**МИНИСТЕРСТВО ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ   
РОСТОВСКОЙ ОБЛАСТИ**

**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССТОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ РОСТОВСКОЙ ОБЛАСТИ**

**«РОСТОВСКИЙ-НА-ДОНУ КОЛЛЕДЖ СВЯЗИ И ИНФОРМАТИКИ»**

|  |  |
| --- | --- |
|  |  |

**ФОНД ОЦЕНОЧНЫХ СРЕДСТВ**

**по профессиональному модулю**

**ПМ.03 «Проектирование и разработка информационных систем»**

**по специальности**

**09.02.11 «Разработка и управление программным обеспечением»**

**(базовой подготовки)**

г. Ростов-на-Дону

2025 г.

|  |  |
| --- | --- |
| **РАССМОТРЕНО**  На заседании цикловой комиссии  программирования  Протокол № 1 от 1.09.2025 года  Председатель ЦК \_\_\_\_\_\_\_\_\_\_\_ А.С.Сулавко | **УТВЕРЖДАЮ**  Зам. директора по УМР  \_\_\_\_\_\_\_\_\_\_\_\_И.В.Подцатова  02 сентября 2025 г. |

Комплект контрольно-оценочных средств по ПМ.03 «Разработка приложений для мобильных платформ» разработан в соответствии с рабочей программой по ПМ, разработанной по специальности 09.02.11 «Разработка и управление программным обеспечением».

Организация-разработчик: Государственное бюджетное профессиональное образовательное учреждение Ростовской области «Ростовский-на-Дону колледж связи и информатики»

Разработчик:

Малая М.А. преподаватель государственного бюджетного профессионального образовательного учреждения Ростовской области «Ростовский-на-Дону колледж связи и информатики».

# СОДЕРЖАНИЕ

1.[Общие положения ………………………………………………………………..4](#_Toc167656260)

[2. Результаты освоения модуля, подлежащие проверке …………………………5](#_Toc167656261)

[3. Оценка освоения теоретического курса профессионального модуля ПМ.03 «Разработка, администрирование и защита баз данных»………..……………..10](#_Toc167656263)

[4. Контроль приобретения практического опыта. Оценка по учебной и (или) производственной практике (по профилю специальности) …………………..136](#_Toc167656286)

[5. Контрольно-оценочные материалы для ПМ.03 «Разработка, администрирование и защита баз данных» …………………………………….138](#_Toc167656287)

# **1.Общие положения**

Результатом освоения профессионального модуля является готовность обучающегося к выполнению вида профессиональной деятельности: «Разработка приложений для мобильных платформ» и сформированность профессиональных и общих компетенций.

Формой аттестации по профессиональному модулю является экзамен по модулю.

**1.Формы промежуточной аттестации по профессиональному модулю**

Таблица 1

|  |  |
| --- | --- |
| **Элементы модуля, профессиональный модуль** | **Формы промежуточной аттестации** |
| **1** | **2** |
| МДК.03.01 «Проектирование информационных систем» | Экзамен |
| МДК.03.02 «Разработка кода информационных систем» | Зачет с оценкой |
| МДК.03.03 «Сопровождение информационных систем» | Зачет с оценкой |
| УП.03.01 Учебная практика | Зачет |
| ПП.03.01 | Зачет |
| ПM.03.ЭК | Квалификационный экзамен |

# **2. Результаты освоения модуля, подлежащие проверке**

# **2.1. Профессиональные и общие компетенции**

В результате контроля и оценки по профессиональному модулю осуществляется комплексная проверка следующих профессиональных и общих компетенций:

Таблица 1

| **Профессиональные компетенции** | **Показатели оценки результата** |
| --- | --- |
| ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ. | * использование современныхcase-средств проектирования баз данных; * выполнение проектирования логической и физической схемы базы данных; * создание хранимых процедур и триггеров на базах данных; * применение стандартных методов для защиты объектов базы данных; * выполнение стандартных процедур резервного копирования и мониторинга выполнения этой процедуры; * выполнение процедур восстановления базы данных и ведение мониторинга выполнения этой процедуры; * организация информационной безопасности на уровне базы данных * *выполнение нормализации баз данных;* * *разработка схемы баз данных в различных СУБД;* * *выполнение репликации БД;* |
| ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт. |
| ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ. |
| ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ. |
| ПК 3.5 Выполнять тестирование и отладку программного обеспечения. |
| ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами. |
| ПК 3.7 Осуществлять защиту данных в мобильных приложения. |

Таблица 2

| **Общие компетенции** | **Показатели оценки результата** |
| --- | --- |
| ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам | * выполнение постановки цели, выбора и применения методов и способов решения профессиональных задач; * выполнение адекватной оценки и самооценки эффективности и качества выполнения профессиональных задач |
| ОК 2 Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности; | * выбор и использование различных источников, включая электронные ресурсы, медиаресурсы, Интернет-ресурсы, периодические издания по специальности для решения профессиональных задач |
| ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях; | * демонстрация ответственности за принятые решения; * выполнение самоанализа и коррекция результатов собственной работы; |
| ОК 4 Эффективно взаимодействовать и работать в коллективе и команде; | * взаимодействовать с обучающимися, преподавателями и мастерами в ходе обучения, с руководителями учебной и производственной практик; * обоснованность анализа работы членов команды (подчиненных) |
| ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста; | * анализ эффективности использования в профессиональной деятельности необходимой технической документации, в том числе на английском языке. |
| ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках. | * выполнение постановки цели, выбора и применения методов и способов решения профессиональных задач; * анализ адекватной оценки и самооценки эффективности и качества выполнения профессиональных задач |

**2.2. Требования к портфолио**

**Тип портфолио:** портфолио работ

**Проверяемые результаты обучения:**

**Основные требования**

Требования к структуре и оформлению портфолио:

1. В портфолио входят практические работы, входящие в МДК.03.01 «Разработка мобильных приложений».
2. Практические работы оформлены в виде отчета.
3. Оформление отчета соответствует требованиям документа «Правила оформления текстовых документов в учебном процессе»

**Показатели оценки портфолио**

Таблица

|  |  |  |
| --- | --- | --- |
| **Коды и наименования проверяемых компетенций или их сочетаний** | **Показатели оценки результата** | **Оценка**  **(да / нет)** |
| ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.  ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.  ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.  ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.  ПК 3.5 Выполнять тестирование и отладку программного обеспечения.  ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.  ПК 3.7 Осуществлять защиту данных в мобильных приложения. | - использование современных case-средств проектирования баз данных;  - выполнение проектирования логической и физической схемы базы данных;  - создание хранимых процедур и триггеров на базах данных;  - применение стандартных методов для защиты объектов базы данных;  - выполнение стандартных процедур резервного копирования и мониторинга выполнения этой процедуры;  - выполнение процедур восстановления базы данных и ведение мониторинга выполнения этой процедуры;  - организация информационной безопасности на уровне базы данных  выполнение нормализации баз данных;   * разработка схемы баз данных в различных СУБД; * выполнение репликации БД; |  |
| ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам  ОК 2 Использовать современные средства поиска, анализа и интерпретации информации, и информационные технологии для выполнения задач профессиональной деятельности;  ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по правовой и финансовой грамотности в различных жизненных ситуациях;  ОК 4 Эффективно взаимодействовать и работать в коллективе и команде;  ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста;  ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках. | * выполнение постановки цели, выбора и применения методов и способов решения профессиональных задач; * выполнение адекватной оценки и самооценки эффективности и качества выполнения профессиональных задач * выбор и использование различных источников, включая электронные ресурсы, медиаресурсы, Интернет-ресурсы, периодические издания по специальности для решения профессиональных задач * демонстрация ответственности за принятые решения; * выполнение самоанализа и коррекция результатов собственной работы; * взаимодействовать с обучающимися, преподавателями и мастерами в ходе обучения, с руководителями учебной и производственной практик; * обоснованность анализа работы членов команды (подчиненных) * анализ эффективности использования в профессиональной деятельности необходимой технической документации, в том числе на английском языке. * выполнение постановки цели, выбора и применения методов и способов решения профессиональных задач; * анализ адекватной оценки и самооценки эффективности и качества выполнения профессиональных задач |  |

**3.Оценка освоения теоретического курса профессионального модуля ПМ.03«Разработка приложений для мобильных платформ»**

**3.1. Задания для оценки освоения МДК**

**3.3.1. Задания для оценки освоения МДК.03.01 «Разработка мобильных приложений»**

***Тема 1.3.1* Основные платформы и языки разработки мобильных приложений**

Ответить на вопросы.

1. Каковы преимущества ОС IOS и Mac OS?

2. Каковы недостатки ОС IOS?

3. Каковы особенности и возможности IOS?

4. Каковы языки используются для программирования под IOS?

5. Каковы основные компоненты приложения под IOS?

6. Какие существуют средства разработки IOS-приложений?

7. Каково назначение Unwind segue?

8. Из чего состоит архитектура IOS-приложения?

9. Для чего используется класс UINavigationController?

10. Для чего используется класс View?

11. Что такое Unwind segue в интерфейсе IOS-приложения?

12. Что такое outlet?

13. Какие кнопоки используются в интерфейсе IOS-приложения?

14. Какие типы входят в коллекции Swift?

15. Как можно обратиться к элементу экрана из кода?

16. Что такое action в IOS-приложении?

17. Объектом какого класса сцена?

18. Как создается переход между сценами?

19. Как добавить контроллер навигации и указать точку входа?

20. Какие визуальные элементы используются в IOS-приложениях?

21. Перечислите основные компоненты IOS-приложения?

22. Что такое Acton?

23. Какие методы вызываются системой для управления жизненным циклом?

24. Каковы возможности и отличия классов и структур в Swift?

25. Как добавить View, и зачем оно нужно?

Критерий оценки:

Каждый полный правильный ответ оценивается 1 баллом.

0-2 правильных ответа — «2»

3 правильных ответа — «3»

4 правильных ответа — «4»

5 правильных ответов — «5»

**Практическое занятие № 1** Работа в playground, Swift

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы:**

1. Научиться создавать исходные модули Swift-приложений.

2. Приобрести навыки запуска Swift-приложений в режиме playground.

**Краткие теоретические сведения:**

Особенности языка Swift

Swift – это новый язык программирования, разработанный Apple Inc для iOS и OS X. Swift принимает лучшее из C и Objective-C, без ограничений совместимости с C. Swift 5 использует ту же среду выполнения, что и существующая система Obj-C в Mac OS и iOS, что позволяет программам Swift работать на многих существующих платформах iOS 6 и OS X 10.8.

Загрузка и установка Swift

Для того чтобы программировать на языке Swift, необходимо:

1. Создать учетную запись на сайте Apple Developer;

2. Загрузить и установить XCode;

Пример 1

import UIKit

var str = "Hello, playground"

print (str)

Разбор программы

1-я строка:

Подключение библиотеки:

import UIKit

Подключаемая библиотека UIKit служит для создания и управления графическим, управляемым событиями пользовательским интерфейсом для приложения iOS или tvOS.

2-я строка:

Создание и инициализация переменной:

var str = "Hello, playground"

var – создание переменной;

str – имя переменной;

“Hello, playground” – инициализация переменной (в данном случае приведен пример с созданием переменной типа String).

print() – печатает сообщения в консоли Xcode при отладке приложений.

Комментарии представляют собой пояснения, которые програм¬мист может включить в программу. Они не являются частью кода и не влияют на работу программы.

В языке Swift используется два вида комментариев (как и в языке С++):

// – однострочный комментарий (до конца строки)

/\* \*/ – многострочный комментарий

Задание для самостоятельного выполнения

1. Создать и выполнить Swift-приложение из примера 1 в режиме playground.

2. Изменить программу из задания 1 таким образом, чтобы она печатала на выходе имя, фамилию, отчество, возраст и группу учащегося.

**Ход работы:**

1. Создание и выполнение Swift-приложения из примера 1 в режиме playground:

1. Откройте Xcode.
2. Выберите "Get started with a playground".
3. Назовите playground, выберите платформу iOS и нажмите "Create".
4. В открывшемся playground введите следующий код:

import UIKit

var str = "Hello, playground"

print(str)

1. Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код. В консоли вы увидите вывод "Hello, playground".

2. Изменение программы из задания 1:

1. Откройте тот же playground или создайте новый.

2. Введите следующий код, чтобы программа печатала на выходе имя, фамилию, отчество, возраст и группу учащегося:

import UIKit

var firstName = "Иван"

var lastName = "Иванов"

var middleName = "Иванович"

var age = 20

var group = "Группа 101"

print("Имя: \(firstName)")

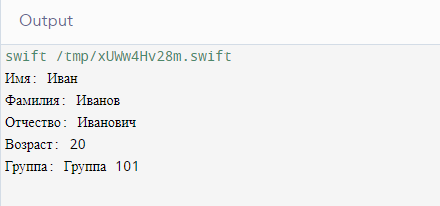
print("Фамилия: \(lastName)")

print("Отчество: \(middleName)")

print("Возраст: \(age)")

print("Группа: \(group)")

3.Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код. В консоли вы увидите вывод:



**Контрольные вопросы:**

* + - Какие шаги нужно выполнить для создания и запуска Swift-приложения в режиме playground?
    - Какая библиотека подключается в примере 1 и для чего она используется?
    - Как объявляются и инициализируются переменные в языке Swift? Приведите пример.
    - Какой тип комментариев поддерживается в языке Swift? Приведите примеры.
    - Напишите код Swift, который выводит на консоль следующие данные: Имя, Фамилия, Отчество, Возраст, Группа учащегося.

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 2** Встроенные типы данных. Объявление и использование переменных. Псевдонимы.

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы**: изучить тему использование переменных.

**Краткие теоретические сведения:**

Встроенные типы данных

Swift предлагает программисту богатый ассортимент как встроенных, так и определяемых пользователем типов данных. Следующие типы базовых типов данных чаще всего используются при объявлении переменных:

– Int или UInt – используется для целых чисел. Более конкретно, вы можете использовать Int32, Int64 для определения 32- или 64-разрядного целого числа со знаком, тогда как UInt32 или UInt64 для определения 32- или 64-разрядных целочисленных переменных без знака. Например: 42 и -23.

– Float – используется для представления 32-разрядного числа с плавающей запятой и чисел с меньшими десятичными точками. Например, 3,14159, 0,1 и -273,158.

– Double – используется для представления 64-битного числа с плавающей запятой и используется, когда значения с плавающей запятой должны быть очень большими.Например, 3,14159, 0,1 и -273,158.

– Bool – Это представляет логическое значение, которое является или истиной или ложью.

– String – это упорядоченная коллекция символов. Например, «Привет, мир!»

– Char – это односимвольный строковый литерал. Например, “С”

– Otional – представляет переменную, которая может содержать либо значение, либо отсутствие значения.

– Cartage – это используется для группировки нескольких значений в одно составное значение.

Введение псевдонимов

Вы можете создать новое имя для существующего типа, используя typealias . Вот простой синтаксис для определения нового типа с помощью typealias –

typealias newname = type

Например, следующая строка указывает компилятору, что Feet – это другое имя для Int –

typealias Feet = Int

Теперь следующее объявление является абсолютно допустимым и создает целочисленную переменную с именем distance –

Пример 1:

typealias Feet = Int

var distance: Feet = 100

print(distance)

Когда мы запускаем вышеуказанную программу, используя ппесочницу, мы получаем следующий результат.

100

Переменные в Swift

Переменная предоставляет нам именованное хранилище, которым наши программы могут манипулировать. Каждая переменная в Swift имеет определенный тип, который определяет размер и расположение памяти переменной; диапазон значений, которые могут быть сохранены в этой памяти; и набор операций, которые могут быть применены к переменной.

Объявление переменной

Объявление переменной сообщает компилятору, где и какого размера нужно создать хранилище для переменной. Прежде чем использовать переменные, вы должны объявить их, используя ключевое слово var следующим образом:

var variableName = <initial value>

В следующем примере показано, как объявить переменную в Swift –

var varA = 42

print(varA)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

42

Именование переменных

Имя переменной может состоять из букв, цифр и символа подчеркивания. Оно должно начинаться либо с буквы, либо с подчеркивания. Прописные и строчные буквы различны, потому что Swift является регистрозависимым языком программирования.

Вы можете использовать простые символы или символы Юникода для именования ваших переменных. В следующих примерах показано, как вы можете назвать переменные –

Пример 2:

var \_var = "Hello, Swift !"

print(\_var)

var переменная = "Привет, Свифт!"

print(переменная)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат.

Hello, Swift !

Привет, Свифт!

Переменные печати

Вы можете распечатать текущее значение константы или переменной с помощью функции печати. Вы можете интерполировать значение переменной, заключив имя в круглые скобки и экранируя его обратной косой чертой перед открывающей скобкой: В примере приведены допустимые примеры –

Пример 3:

var varA = "Godzilla"

var varB = 1000.00

print("Value of \(varA) is more than \(varB) millions")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат.

Value of Godzilla is more than 1000.00 millions

Задание 1

Создать консольное приложение из примера 1.

Задание 2

Создать консольное приложение из примера 3.

Задание 3

Создать консольное приложение выводящее информацию в соответствии с вариантами.

Варианты

1. Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)

2. Книга (название книги, автор, издательство, количество страниц)

3. Файл (имя, время создания, дата создания, размер)

4. Спортсмен (ФИО, вид спорта, сколько лет занимается)

5. Студент (ФИО, группа, средний балл зачетки)

6. Фирма (название, вид услуг, годовой оборот)

7. Компьютер (производитель, частота процессора, количество ядер, объем памяти)

8. Спорт (вид спорта, количество занимающихся в городе, средняя стоимость комплекта инвентаря)

9. Автомобиль (марка, фирма-производитель, год выпуска, пробег)

10. Статья (название, автор, название журнала, год и месяц издания)

**Ход работы:**

Задание 1: Создание консольного приложения из примера 1

1. Откройте Xcode.
2. Создайте новый проект, выбрав "macOS" и "Command Line Tool".
3. Назовите проект и выберите язык Swift.
4. В файле main.swift введите следующий код:

import Foundation

typealias Feet = Int

var distance: Feet = 100

print(distance)

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.

Задание 2: Создание консольного приложения из примера 3

В том же проекте или создайте новый проект.

В файле main.swift введите следующий код:

import Foundation

var varA = "Godzilla"

var varB = 1000.00

print("Value of \(varA) is more than \(varB) millions")

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.

Задание 3: Создание консольного приложения с вариантами

Выберите один из вариантов и создайте соответствующий код.

Пример для Варианта 5 (Студент):

В том же проекте или создайте новый проект.

В файле main.swift введите следующий код:

import Foundation

var firstName = "Иван"

var lastName = "Иванов"

var group = "Группа 101"

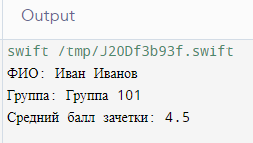
var averageScore: Float = 4.5

print("ФИО: \(firstName) \(lastName)")

print("Группа: \(group)")

print("Средний балл зачетки: \(averageScore)")

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.



**Контрольные вопросы**

1. Какие встроенные типы данных существуют в Swift? Приведите примеры.
2. Как создать новый тип данных с помощью typealias? Приведите пример.
3. Как объявить и инициализировать переменную в Swift? Приведите пример.
4. Как вывести значение переменной на консоль с использованием интерполяции строк? Приведите пример.
5. Что такое опциональные значения в Swift и как их использовать? Приведите пример.

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 3** Кортежи, константы, операторы

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы**: Изучить и научиться работать с кортежами.

Краткие теоретические сведения

Кортежи

Swift также вводит тип кортежей, которые используются для группировки нескольких значений в одно составное.

Значения в кортеже могут быть любого типа и не обязательно должны быть одного типа.

Например, («Tutorials Point», 123) – это кортеж с двумя значениями, одно из которых имеет строковый тип, а другое – целочисленный тип.

let PracticeError = (501, «Не реализовано») – это ошибка, когда что-то на сервере не реализовано. Возвращает два значения. Код ошибки и описание.

Вы можете создавать кортежи из любого количества значений и любого количества различных типов данных.

Вот синтаксис объявления Tuple –

var TupleName = (Value1, value2,… any number of values)

Вот объявление Tuple –

var error501 = (501, “Not implemented”)

Вы можете получить доступ к значениям кортежа, используя номера индекса, начинающиеся с 0.

Вот пример доступа к Значениям кортежа –

print(“The code is\(error501.0)”)

print(“The definition of error is\(error501.1)”)

Вы можете называть переменные кортежа при объявлении, и вы можете вызывать их, используя их имена

var error501 = (errorCode: 501, description: “Not Implemented”)

print(error501.errorCode) // prints 501.

Кортежи помогают возвращать несколько значений из функции. Например, веб-приложение может вернуть кортеж типа («String», Int), чтобы показать, была ли загрузка успешной или неудачной.

Возвращая разные значения в кортеже, мы можем принимать решения в зависимости от разных типов кортежей.

Константы

Константы относятся к фиксированным значениям, которые программа не может изменить во время своего выполнения. Константы могут быть любого из основных типов данных, таких как целочисленная константа, плавающая константа, символьная константа.

Константы обрабатываются как обычные переменные, за исключением того факта, что их значения не могут быть изменены после их определения.

Декларация констант

Прежде чем использовать константы, вы должны объявить их с помощью ключевого слова let следующим образом:

let constantName = <initial value>

Ниже приведен простой пример того, как объявить константу в Swift 4.

let constA = 42

print(constA)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –42

Операторы

Операция – это символ, который указывает компилятору выполнять определенные математические или логические манипуляции. Objective-C богат встроенными операторами и предоставляет следующие типы операторов –

• Арифметические Операторы

• Операции сравнения

• Логические Операторы

• Битовые операции

• Операции присваивания

• Операции диапазона

• Разные Операторы

Это задание объяснит один за другим арифметические, реляционные, логические, побитовые, присваивания и другие операторы.

Арифметические Операторы

В следующей таблице приведены все арифметические операции, поддерживаемые языком Swift. Предположим, что переменная A содержит 10, а переменная B содержит 20, тогда –

Оператор Описание Пример

+ Добавляет два операнда А + Б даст 30

– Вычитает второй операнд из первого A – B даст -10

\* Умножает оба операнда А \* Б даст 200

/ Делит числитель на знаменатель Б / у даст 2

% Оператор модуля и остаток от целочисленного / делительного числа B% A даст 0

Операции сравнения

В следующей таблице показаны все реляционные операции, поддерживаемые языком Swift. Предположим, что переменная A содержит 10, а переменная B содержит 20, тогда –

Оператор Описание Пример

== Проверяет, равны ли значения двух операндов или нет; если да, то условие становится истинным. (A == B) не соответствует действительности.

знак равно Проверяет, равны ли значения двух операндов или нет; если значения не равны, то условие становится истинным. (A! = B) верно.

> Проверяет, больше ли значение левого операнда, чем значение правого операнда; если да, то условие становится истинным. (A> B) не соответствует действительности.

< Проверяет, меньше ли значение левого операнда, чем значение правого операнда; если да, то условие становится истинным. (A <B) верно.

> = Проверяет, больше ли значение левого операнда или равно значению правого операнда; если да, то условие становится истинным. (A> = B) не соответствует действительности.

<= Проверяет, меньше ли значение левого операнда или равно значению правого операнда; если да, то условие становится истинным. (A <= B) верно.

Логические Операторы

Оператор Описание Пример

&& Называется логический оператор И. Если оба операнда отличны от нуля, условие становится истинным. (A && B) неверно.

|| Вызывается логическим оператором ИЛИ. Если любой из двух операндов отличен от нуля, условие становится истинным. (A || B) верно.

! Вызывается логическим оператором НЕ. Используйте для изменения логического состояния своего операнда. Если условие истинно, то оператор Logical NOT сделает его ложным. ! (A && B) верно.

В следующей таблице приведены все логические операции, поддерживаемые языком Swift. Предположим, что переменная A содержит 1, а переменная B содержит 0, тогда –

Операции диапазона

Swift включает в себя два операции диапазона, которые являются ярлыками для выражения диапазона значений. Следующая таблица объясняет эти два оператора.

Оператор Описание Пример

Закрытый диапазон (a … b) определяет диапазон от a до b и включает значения a и b. 1 … 5 дает 1, 2, 3, 4 и 5

Полуоткрытый диапазон (a .. <b) определяет диапазон от a до b, но не включает b. 1 .. <5 дает 1, 2, 3 и 4

Односторонний диапазон a…, определяет диапазон, который проходит от a до конца элементов

… A, определяет диапазон, начиная от начала до 1… дает 1, 2,3… конец элементов

… 2 дает начало… 1,2

Задание для самостоятельного выполнения

1. Создать и вывести в консоль кортеж состоящий из имени, фамилии, отчества, возраста, роста, веса, места обучения и группы.

2. Создать два кортежа в виде 2 учеников (как в задании 1) и сравнить их возраст, рост и вес. Вывести полученный результат в консоль.

3. Создать в соответствии с вариантами 3 кортежа, сравнивая между собой численные параметры.

**Ход работы:**

Задание 1: Создание и вывод в консоль кортежа

1. Откройте Xcode и создайте новый проект, выбрав "macOS" и "Command Line Tool".
2. Назовите проект и выберите язык Swift.
3. В файле main.swift введите следующий код для создания и вывода в консоль кортежа, состоящего из имени, фамилии, отчества, возраста, роста, веса, места обучения и группы:

import Foundation

let studentInfo = (firstName: "Иван", lastName: "Иванов", middleName: "Иванович", age: 20, height: 180, weight: 75, placeOfStudy: "Университет", group: "Группа 101")

print("Имя: \(studentInfo.firstName)")

print("Фамилия: \(studentInfo.lastName)")

print("Отчество: \(studentInfo.middleName)")

print("Возраст: \(studentInfo.age)")

print("Рост: \(studentInfo.height) см")

print("Вес: \(studentInfo.weight) кг")

print("Место обучения: \(studentInfo.placeOfStudy)")

print("Группа: \(studentInfo.group)")

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.

Задание 2: Создание и сравнение двух кортежей учеников

В том же проекте или создайте новый проект.

В файле main.swift введите следующий код для создания двух кортежей и сравнения их возрастов, роста и веса:

import Foundation

let student1 = (firstName: "Иван", lastName: "Иванов", middleName: "Иванович", age: 20, height: 180, weight: 75, placeOfStudy: "Университет", group: "Группа 101")

let student2 = (firstName: "Петр", lastName: "Петров", middleName: "Петрович", age: 22, height: 175, weight: 80, placeOfStudy: "Университет", group: "Группа 102")

if student1.age == student2.age {

print("Оба студента одного возраста")

} else {

print("Студенты разного возраста")

}

if student1.height == student2.height {

print("Оба студента одного роста")

} else {

print("Студенты разного роста")

}

if student1.weight == student2.weight {

print("Оба студента одного веса")

} else {

print("Студенты разного веса")

}

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.

Задание 3: Создание и сравнение трех кортежей в соответствии с вариантами

Выберите один из вариантов и создайте соответствующий код.

Пример для Варианта 9 (Автомобиль):

В том же проекте или создайте новый проект.

В файле main.swift введите следующий код для создания трех кортежей автомобилей и сравнения их пробега:

swift

Копировать код

import Foundation

let car1 = (brand: "Toyota", manufacturer: "Toyota Motor Corporation", year: 2015, mileage: 50000)

let car2 = (brand: "Honda", manufacturer: "Honda Motor Co., Ltd.", year: 2018, mileage: 30000)

let car3 = (brand: "Ford", manufacturer: "Ford Motor Company", year: 2020, mileage: 20000)

if car1.mileage == car2.mileage && car2.mileage == car3.mileage {

print("Все автомобили имеют одинаковый пробег")

} else {

print("Автомобили имеют разный пробег")

}

if car1.mileage < car2.mileage && car1.mileage < car3.mileage {

print("Первый автомобиль имеет наименьший пробег")

} else if car2.mileage < car1.mileage && car2.mileage < car3.mileage {

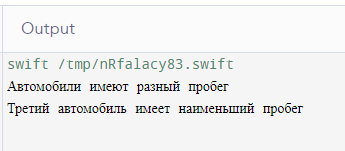
print("Второй автомобиль имеет наименьший пробег")

} else {

print("Третий автомобиль имеет наименьший пробег")

}

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.



**Контрольные вопросы:**

1. Что такое кортеж в Swift и для чего он используется?
2. Как объявить и инициализировать кортеж в Swift? Приведите пример.
3. Как получить доступ к значениям кортежа по индексу? Приведите пример.
4. Как создать и использовать константы в Swift? Приведите пример.
5. Какие виды операторов существуют в Swift? Приведите примеры.

**Варианты**

1. Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)

2. Книга (название книги, автор, издательство, количество страниц)

3. Файл (имя, время создания, дата создания, размер)

4. Спортсмен (ФИО, вид спорта, сколько лет занимается)

5. Студент (ФИО, группа, средний балл зачетки)

6. Фирма (название, вид услуг, годовой оборот)

7. Компьютер (производитель, частота процессора, количество ядер, объем памяти)

8. Спорт (вид спорта, количество занимающихся в городе, средняя стоимость комплекта инвентаря)

9. Автомобиль (марка, фирма-производитель, год выпуска, пробег)

10. Статья (название, автор, название журнала, год и месяц издания)

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 4** Условия и циклы.

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы**: Изучить работу условий, научиться применять на практике.

**Краткие теоретические сведения**

Структуры принятия решений требуют, чтобы программист определял одно или несколько условий, которые должны быть оценены или протестированы программой, вместе с оператором или инструкциями, которые должны быть выполнены, если условие определено как истинное, и необязательно, другие операторы, которые должны быть выполнены, если условие определяется как ложное.

Ниже приводится общее описание типичной структуры принятия решений, встречающейся в большинстве языков программирования.

Swift предоставляет следующие типы решений для принятия решений.

Номер Заявление и описание

1 если,условие

Оператор if состоит из логического выражения, за которым следует один или несколько операторов.

2 если … то,условие

За оператором if может следовать необязательный оператор else, который выполняется, когда логическое выражение имеет значение false.

3 if … else if … else условие

За оператором if может следовать необязательный оператор elseif … else, который очень полезен для проверки различных условий, используя единственный оператор if … elseif.

4 вложенные операторы if

Вы можете использовать один оператор if или elseif внутри другого оператора if или elseif.

5 условие о переключении

Оператор switch позволяет проверять переменную на соответствие списку значений.

Тернарная операция? :

Как вы уже знаете тернарная операция может быть использована для замены операторов if … else . Он имеет следующую общую форму –

Exp1 ? Exp2 : Exp3;

Где Exp1, Exp2 и Exp3 являются выражениями. Обратите внимание на использование и размещение вопросительного знака.

Выражение определяется следующим образом: Exp1 оценивается. Если это правда, то Exp2 оценивается и становится значением выражение. Если Exp1 имеет значение false, то Exp3 оценивается, и его значение становится значением выражения.

Может возникнуть ситуация, когда вам нужно выполнить блок кода несколько раз. В общем случае операторы выполняются последовательно: первый оператор в функции выполняется первым, затем второй и так далее.

Языки программирования предоставляют различные управляющие структуры, которые допускают более сложные пути выполнения.

Оператор цикла позволяет нам выполнять оператор или группу операторов несколько раз. Ниже приведено общее из оператора цикла в большинстве языков программирования:

Язык программирования Swift предоставляет следующие виды циклов для обработки требований циклов.С помощью цикла for-in мы можем перебрать элементы коллекции (массивы, множества, словари) или последовательности. Он имеет следующую форму:

for объект\_последовательности in последовательность {

// действия

}

Например, переберем элементы массива:

for item in 1...5 {

print(item)}

Выражение 1...5 образует последовательность из пяти чисел от 1 до 5. И цикл проходит по всем элементам последовательности. При каждом проходе он извлекает одно число и передает его в переменную item. Таким образом, цикл сработает пять раз.

С помощью оператора where можно задавать условия выборки из последовательности элементов:

for i in 0...10 where i % 2 == 0 {

print(i) // 0, 2, 4, 6, 8, 10

}

Здесь из последовательности 0...10 извлекаются только те элементы, которые соответствуют условию после оператора where - i % 2 == 0, то есть четные числа.

Оператор while проверяет некоторое условие, и если оно возвращает true, то выполняет блок кода. Этот цикл имеет следующую форму:

while условие {

// действия

}

Например:

var i = 10

while i > 0 {

print(i)

i-=1

}

При этом надо внимательно подходить к условию. Если оно всегда будет возвращать true, то мы получим бесконечный цикл, и приложение может зависнуть.

Цикл repeat-while сначала выполняет один раз цикл, и если некоторое условие возвращает true, то продолжает выполнение цикла. Он имеет следующую форму:

repeat{

// действия

} while условие

Например, перепишем предыдущий цикл while:

var i = 10

repeat{

print(i)

i-=1

} while i > 0

Здесь цикл также выполнится 10 раз, пока значение переменной i не станет равно 0.

Но рассмотрим другую ситуацию:

vari = -1

repeat{

print(i)

i-=1

} while i > 0

Несмотря на то, что переменная i меньше 0, но цикл выполнится один раз.

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае мы можем воспользоваться оператором break.

for i in 0...10 {

if i == 5{

break

}

print(i) // 0, 1, 2, 3, 4

}

Поскольку в цикле идет проверка, равно ли значение переменной i числу 5, то когда перебор дойдет до числа 5, сработает оператор break, и цикл завершится.

Теперь поставим себе другую задачу. А что если мы хотим, чтобы при проверке цикл не завершался, а просто переходил к следующему элементу. Для этого мы можем воспользоваться оператором continue:

fori in 0...10 {

if i == 5{

continue

}

print(i) // 0, 1, 2, 3, 4, 6, 7, 8, 9, 10

}

В этом случае цикл, когда дойдет до числа 5, которое не удовлетворяет условию проверки, просто пропустит это число и перейдет к следующему элементу последовательности.

Задание для самостоятельного выполнения

1. Создать и вывести в консоль кортеж содержащий ошибку подключения к серверу (текст сообщения) и ее номер (код), проверить через if код ошибки и вывести сообщение.

2. Создать два кортежа (как в задании 1) и переменную error ей присвоить значение. Вывести полученное сообщение в консоль.

3. Создать в соответствии с вариантами 3 кортежа, использовать цикл, сравнивая собой численные коды и значение переменной error, вывести сообщение.

**Ход работы:**

Задание 1: Создание и вывод кортежа, содержащего ошибку подключения к серверу

Откройте Xcode и создайте новый проект, выбрав "macOS" и "Command Line Tool".

Назовите проект и выберите язык Swift.

В файле main.swift введите следующий код для создания кортежа с ошибкой подключения к серверу и проверки кода ошибки:

import Foundation

let connectionError = (errorCode: 404, errorMessage: "Not Found")

if connectionError.errorCode == 404 {

print("Ошибка: \(connectionError.errorMessage)")

} else {

print("Ошибка не определена.")

}

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.

Задание 2: Создание двух кортежей и переменной error

В том же проекте или создайте новый проект.

В файле main.swift введите следующий код для создания двух кортежей с ошибками подключения к серверу и проверки кода ошибки через переменную error:

import Foundation

let connectionError1 = (errorCode: 404, errorMessage: "Not Found")

let connectionError2 = (errorCode: 500, errorMessage: "Internal Server Error")

let error = connectionError1

if error.errorCode == 404 {

print("Ошибка: \(error.errorMessage)")

} else if error.errorCode == 500 {

print("Ошибка: \(error.errorMessage)")

} else {

print("Ошибка не определена.")

}

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.

Задание 3: Создание трех кортежей и использование цикла

В том же проекте или создайте новый проект.

В файле main.swift введите следующий код для создания трех кортежей в соответствии с вариантом (например, Вариант 21: Автобус) и использования цикла для сравнения численных кодов:

import Foundation

// Вариант 21: Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)

let bus1 = (routeNumber: 1, startStop: "A", endStop: "B", stopsCount: 10)

let bus2 = (routeNumber: 2, startStop: "C", endStop: "D", stopsCount: 15)

let bus3 = (routeNumber: 3, startStop: "E", endStop: "F", stopsCount: 8)

let buses = [bus1, bus2, bus3]

let error = (errorCode: 15, errorMessage: "Too many stops")

for bus in buses {

if bus.stopsCount == error.errorCode {

print("Ошибка: \(error.errorMessage) на маршруте \(bus.routeNumber)")

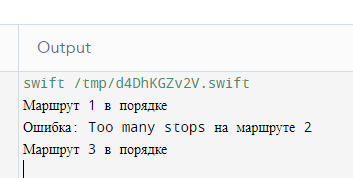
} else {

print("Маршрут \(bus.routeNumber) в порядке")

}

}

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.



Варианты

1. Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)

2. Книга (название книги, автор, издательство, количество страниц)

3. Файл (имя, время создания, дата создания, размер)

4. Спортсмен (ФИО, вид спорта, сколько лет занимается)

5. Студент (ФИО, группа, средний балл зачетки)

6. Фирма (название, вид услуг, годовой оборот)

7. Компьютер (производитель, частота процессора, количество ядер, объем памяти)

8. Спорт (вид спорта, количество занимающихся им в городе, средняя стоимость комплекта инвентаря)

9. Автомобиль (марка, фирма-производитель,год выпуска, пробег)

10. Статья (название, автор, название журнала, год и месяц издания)

**Контрольные вопросы:**

1. Что такое условия в программировании и как они используются в Swift?
2. Какие виды циклов существуют в Swift и в чем их особенности?
3. Как создавать и использовать кортежи в Swift? Приведите примеры.
4. Как использовать операторы if, else, else if и switch для принятия решений?
5. Что такое тернарный оператор и как его использовать в Swift? Приведите пример.

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 5** Массивы и словари

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы:** изучить устройство и механизм работы массивов, научиться применять их на практике.

**Краткие теоретические сведения**

Массивы Swift используются для хранения упорядоченных списков значений одного типа. Swift устанавливает строгую проверку, которая не позволяет вводить неверный тип в массив, даже по ошибке.

Если вы назначаете созданный массив переменной, то он всегда изменчив, что означает, что вы можете изменить его, добавив, удалив или изменив его элементы; но если вы назначите массив константе, то этот массив является неизменным, и его размер и содержимое не могут быть изменены.

Создание массивов

Вы можете создать пустой массив определенного типа, используя следующий синтаксис инициализатора –

Var someArray = [SomeType]()

Вот синтаксис для создания массива заданного размера a \* и инициализации его значением –

var someArray = [SomeType](count: NumbeOfElements, repeatedValue: InitialValue)

Вы можете использовать следующую инструкцию, чтобы создать пустой массив типа Int,имеющий 3 элемента и начальное значение как ноль –

var someInts = [Int](count: 3, repeatedValue: 0)

Ниже приведен еще один пример создания массива из трех элементов и назначения трех значений этому массиву:

var someInts:[Int] = [10, 20, 30]

Создание массива случайных чисел размера range

static func random(in range: Range<Int>) -> Int

Генерация массива случайных чисел

for \_ in 1...3 {

print(Int.random(in: 1..<100))

}

// Prints "53"

// Prints "64"

// Prints "5"

Вы можете извлечь значение из массива, используя синтаксис индекса, передавая индекс значением, которое вы хотите получить, в квадратных скобках сразу после имени массива следующим образом:

Var someVar = someArray[index]

Здесь индекс начинается с 0, что означает, что к первому элементу можно получить доступ, используя индекс как 0, ко второму элементу можно получить доступ, используя индекс как 1, и так далее. В следующем примере показано, как создавать, инициализировать и обращаться к массивам:

Var someInts=[Int](count:3,repeatedValue:10)

Var someVar=someInts[0]

print("Value of first element is \(someVar)")

print("Value of second element is \(someInts[1])")

print("Value of third element is \(someInts[2])")

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Value of first element is 10

Value of second element is 10

Value of third element is 10

Вы можете использовать метод append () или оператор присваивания (+ =), чтобы добавить новый элемент в конец массива. Посмотрите на следующий пример. Здесь мы изначально создаем пустой массив, а затем добавляем новые элементы в тот же массив –

Var someInts=[Int]()

someInts.append(20)

someInts.append(30)

someInts+=[40]

var someVar=someInts[0]

print("Value of first element is \(someVar)")

print("Value of second element is \(someInts[1])")

print("Value of third element is \(someInts[2])")

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Value of first element is 20

Value of second element is 30

Value of third element is 40

Вы можете изменить существующий элемент массива, назначив новое значение по заданному индексу, как показано в следующем примере:

Var someInts=[Int]()

someInts.append(20)

someInts.append(30)

someInts+=[40]

// Modify last element

someInts[2]=50

var someVar=someInts[0]

print("Value of first element is \(someVar)")

print("Value of second element is \(someInts[1])")

print("Value of third element is \(someInts[2])")

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Value of first element is 20

Value of second element is 30

Value of third element is 50

Вы можете использовать цикл for-in для итерации по всему набору значений в массиве, как показано в следующем примере:

Var someStrs=[String]()

someStrs.append("Apple")

someStrs.append("Amazon")

someStrs+=["Google"]

for item insomeStrs{

print(item)

}

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Apple

Amazon

Google

Вы можете использовать функцию enumerate (), которая возвращает индекс элемента вместе с его значением, как показано ниже в следующем примере –

Var someStrs=[String]()

someStrs.append("Apple")

someStrs.append("Amazon")

someStrs+=["Google"]

for(index, item)insomeStrs.enumerated(){

print("Value at index = \(index) is \(item)")

}

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Value at index = 0 is Apple

Value at index = 1 is Amazon

Valueatindex = 2 isGoogle

Сложение двух массивов

Вы можете использовать оператор сложения (+), чтобы сложить два массива одного типа, которые приведут к новому массиву с комбинацией значений из двух массивов следующим образом:

Var intsA=[Int](count:2,repeatedValue:2)

Var intsB=[Int](count:3,repeatedValue:1)

Var intsC=intsA+intsB

for item inintsC{

print(item)

}

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

2

2

1

1

1

Вы можете использовать свойство count только для чтения массива, чтобы узнать количество элементов в массиве, показанном ниже –

Var intsA = [Int](count:2, repeatedValue: 2)

Var intsB = [Int](count:3, repeatedValue: 1)

Var intsC = intsA + intsB

print("Total items in intsA = \(intsA.count)")

print("Total items in intsB = \(intsB.count)")

print("Total items in intsC = \(intsC.count)")

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Total items in intsA = 2

Total items in intsB = 3

Total items in intsC = 5

Словари Swift используются для хранения неупорядоченных списков значений одного типа.Swift устанавливает строгую проверку, которая не позволяет вводить неверный тип в словарь даже по ошибке.

В словарях Swift используется уникальный идентификатор, известный как ключ, для хранения значения, на которое позже можно ссылаться и искать через тот же ключ. В отличие от элементов в массиве, элементы в словаре не имеют определенного порядка. Вы можете использовать словарь, когда вам нужно искать значения на основе их идентификаторов.

Ключ словаря может быть целым числом или строкой без ограничения, но он должен быть уникальным в словаре.

Если вы назначаете созданный словарь переменной, то он всегда изменчив, что означает, что вы можете изменить его, добавив, удалив или изменив его элементы. Но если вы назначаете словарь постоянной, то этот словарь является неизменным, и его размер и содержание не могут быть изменены.

Создание словаря

Вы можете создать пустой словарь определенного типа, используя следующий синтаксис инициализатора –

varsomeDict = [KeyType:ValueType]()

Вы можете использовать следующий простой синтаксис для создания пустого словаря, ключ которого будет иметь тип Int, а связанные значения будут строками –

Var someDict = [Int:String]()

Вот пример для создания словаря из набора заданных значений –

Var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]

Инициализация на основе последовательности

Swift позволяет вам создавать словарь из массивов (Key-ValuePairs.)

var cities = [“Delhi”,”Bangalore”,”Hyderabad”]

Вы можете использовать следующий простой синтаксис для создания пустого словаря, ключ которого будет иметь тип Int, а связанные значения будут строками –

Var Distance = [2000,10, 620]

Вот пример для создания словаря из набора заданных значений –

Let cityDistanceDict = Dictionary(uniqueKeysWithValues: zip(cities, Distance))

Приведенные выше строки кода создадут словарь с городами в качестве ключа и расстоянием в качестве значения –

Фильтрация

Swift 4 позволяет фильтровать значения из словаря.

Var closeCities = cityDistanceDict.filter { $0.value < 1000 }

Если мы запустим приведенный выше код, наш словарь closeCities будет.

["Bangalore" : 10 , "Hyderabad" : 620]

Группировка словаря

Swift позволяет создавать группы значений словаря.

varcities = ["Delhi","Bangalore","Hyderabad","Dehradun","Bihar"]

Вы можете использовать следующий простой синтаксис для группировки значений словаря по первому алфавиту.

Var GroupedCities = Dictionary(grouping: cities ) { $0.first! }

Результатприведенноговышекодабудет

["D" :["Delhi","Dehradun"], "B" : ["Bengaluru","Bihar"], "H" : ["Hyderabad"]]

Вы можете извлечь значение из словаря, используя синтаксис нижнего индекса, передавая ключ значения, которое вы хотите получить, в квадратных скобках сразу после имени словаря следующим образом:

Var someVar = someDict[key]

Давайте проверим следующий пример для создания, инициализации и доступа к значениям из словаря –

Var someDict:[Int:String]=[1:"One",2:"Two",3:"Three"]

Var someVar=someDict[1]

print("Value of key = 1 is \(someVar)")

print("Value of key = 2 is \(someDict[2])")

print("Value of key = 3 is \(someDict[3])")

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Value of key = 1 is Optional("One")

Value of key = 2 is Optional("Two")

Value of key = 3 is Optional("Three")

Модификация словарей

Вы можете использовать метод updateValue (forKey :), чтобы добавить существующее значение к заданному ключу словаря. Этот метод возвращает необязательное значение типа значения словаря. Вот простой пример –

Var someDict:[Int:String]=[1:"One",2:"Two",3:"Three"]

Var oldVal=someDict.updateValue("New value of one",forKey:1)

Var someVar=someDict[1]

print("Old value of key = 1 is \(oldVal)")

print("Value of key = 1 is \(someVar)")

print("Value of key = 2 is \(someDict[2])")

print("Value of key = 3 is \(someDict[3])")

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Old value of key = 1 is Optional("One")

Value of key = 1 is Optional("New value of one")

Value of key = 2 is Optional("Two")

Value of key = 3 is Optional("Three")

Вы можете изменить существующий элемент словаря, назначив новое значение для данного ключа, как показано в следующем примере –

Var someDict:[Int:String]=[1:"One",2:"Two",3:"Three"]

Var oldVal=someDict[1]

someDict[1]="New value of one"

var someVar=someDict[1]

print("Old value of key = 1 is \(oldVal)")

print("Value of key = 1 is \(someVar)")

print("Value of key = 2 is \(someDict[2])")

print("Value of key = 3 is \(someDict[3])")

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Old value of key = 1 is Optional("One")

Value of key = 1 is Optional("New value of one")

Value of key = 2 is Optional("Two")

Value of key = 3 is Optional("Three")

Удаление пары ключ-значение

Вы можете использовать метод removeValueForKey () для удаления пары ключ-значение из словаря. Этот метод удаляет пару ключ-значение, если она существует, и возвращает удаленное значение или возвращает nil, если значение не существовало. Вотпростойпример –

varsomeDict:[Int:String]=[1:"One",2:"Two",3:"Three"]

varremovedValue=someDict.removeValue(forKey:2)

print("Value of key = 1 is \(someDict[1])")

print("Value of key = 2 is \(someDict[2])")

print("Value of key = 3 is \(someDict[3])")

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Value of key = 1 is Optional("One")

Value of key = 2 is nil

Value of key = 3 is Optional("Three")

Вы также можете использовать синтаксис нижнего индекса для удаления пары ключ-значение из словаря, присваивая значение nil для этого ключа. Вот простой пример –

Var someDict:[Int:String]=[1:"One",2:"Two",3:"Three"]

someDict[2]=nil

print("Value of key = 1 is \(someDict[1])")

print("Value of key = 2 is \(someDict[2])")

print("Value of key = 3 is \(someDict[3])")

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Value of key = 1 is Optional("One")

Value of key = 2 is nil

Value of key = 3 is Optional("Three")

Перебор словаря

Вы можете использовать цикл for-in для итерации по всему набору пар ключ-значение в Словаре, как показано в следующем примере:

Var someDict:[Int:String]=[1:"One",2:"Two",3:"Three"]

for(index,keyValue)insomeDict.enumerated(){

print("Dictionary key \(index) - Dictionary value \(keyValue)")

}

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Dictionary key 2 - Dictionary value Two

Dictionary key 3 - Dictionary value Three

Dictionarykey 1 - DictionaryvalueOne

Вы можете использовать функцию enumerate (), которая возвращает индекс элемента вместе с его парой (ключ, значение), как показано ниже в примере –

Var someDict:[Int:String]=[1:"One",2:"Two",3:"Three"]

for(key, value)insomeDict.enumerated(){

print("Dictionary key \(key) - Dictionary value \(value)")

}

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Dictionary key 0 - Dictionary value (key: 2, value: "Two")

Dictionary key 1 - Dictionary value (key: 3, value: "Three")

Dictionary key 2 - Dictionary value (key: 1, value: "One")

Преобразовать в массивы

Вы можете извлечь список пар ключ-значение из данного словаря, чтобы построить отдельные массивы для ключей и значений. Вотпример –

Var someDict:[Int:String]=[1:"One",2:"Two",3:"Three"]

Let dictKeys=[Int](someDict.keys)

Let dictValues=[String](someDict.values)

print("Print Dictionary Keys")

for(key)in dictKeys{

print("\(key)")

}

print("Print Dictionary Values")

for(value)indictValues{

print("\(value)")

}

Когда приведенный выше код компилируется и выполняется, он дает следующий результат –

Print Dictionary Keys

2

3

1

Print Dictionary Values

Two

Three

One

Задание для самостоятельного выполнения

1. Создать массив, заполнить его и вывести в консоль максимальный элемент.

2. Создать словарь из 2 учеников (состоящий из имени, фамилии, отчества, возраста, роста, веса, места обучения и группы) и сравнить их возраст, рост и вес. Вывести самого высокого студента. Вывести полученный результат в консоль.

3. Создать в соответствии с вариантами словарь, сравнивая между собой численные параметры. Вывести максимальное целочисленное значение и всю соответствующую ему информацию.

**Ход работы:**

Задание 1: Создание массива и вывод максимального элемента

Откройте Xcode и создайте новый проект, выбрав "macOS" и "Command Line Tool".

Назовите проект и выберите язык Swift.

В файле main.swift введите следующий код для создания массива, заполнения его элементами и вывода максимального элемента:

import Foundation

// Создание массива

let numbers = [23, 45, 67, 89, 12, 34, 56]

// Поиск максимального элемента

if let maxNumber = numbers.max() {

print("Максимальный элемент в массиве: \(maxNumber)")

} else {

print("Массив пуст.")

}

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.

Задание 2: Создание словаря учеников и сравнение их параметров

В том же проекте или создайте новый проект.

В файле main.swift введите следующий код для создания словаря из двух учеников, сравнения их возраста, роста и веса, и вывода самого высокого студента:

import Foundation

// Создание словаря учеников

let students: [String: (firstName: String, lastName: String, patronymic: String, age: Int, height: Int, weight: Int, school: String, group: String)] = [

"student1": ("Ivan", "Ivanov", "Ivanovich", 20, 180, 75, "University A", "Group 101"),

"student2": ("Petr", "Petrov", "Petrovich", 22, 185, 80, "University B", "Group 102")

]

// Сравнение параметров

let student1 = students["student1"]!

let student2 = students["student2"]!

let olderStudent = student1.age > student2.age ? student1 : student2

let tallerStudent = student1.height > student2.height ? student1 : student2

let heavierStudent = student1.weight > student2.weight ? student1 : student2

print("Самый высокий студент: \(tallerStudent.firstName) \(tallerStudent.lastName), Рост: \(tallerStudent.height) см")

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.

Задание 3: Создание словаря и вывод максимального целочисленного значения

В том же проекте или создайте новый проект.

В файле main.swift введите следующий код для создания словаря в соответствии с вариантом (например, Вариант 21: Автобус) и вывода максимального целочисленного значения и всей соответствующей ему информации:

import Foundation

// Вариант 21: Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)

let buses: [String: (routeNumber: Int, startStop: String, endStop: String, stopsCount: Int)] = [

"bus1": (1, "Stop A", "Stop B", 10),

"bus2": (2, "Stop C", "Stop D", 15),

"bus3": (3, "Stop E", "Stop F", 8)

]

// Поиск максимального количества остановок

var maxStopsBus: (key: String, value: (routeNumber: Int, startStop: String, endStop: String, stopsCount: Int))? = nil

for bus in buses {

if maxStopsBus == nil || bus.value.stopsCount > maxStopsBus!.value.stopsCount {

maxStopsBus = bus

}

}

if let maxBus = maxStopsBus {

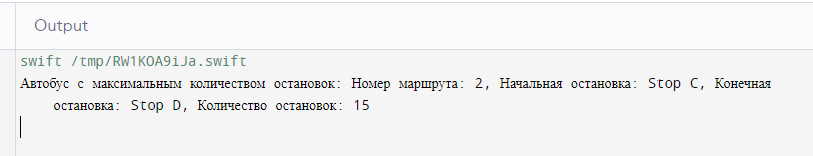
print("Автобус с максимальным количеством остановок: Номер маршрута: \(maxBus.value.routeNumber), Начальная остановка: \(maxBus.value.startStop), Конечная остановка: \(maxBus.value.endStop), Количество остановок: \(maxBus.value.stopsCount)")

} else {

print("Словарь пуст.")

}

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.



**Контрольные вопросы:**

1. Что такое массивы и как они используются в Swift?
2. Что такое словари и как они используются в Swift?
3. Как найти максимальное значение в массиве в Swift?
4. Как создавать и использовать словари в Swift? Приведите примеры.
5. Как сравнивать элементы в словаре и находить максимальное значение?

Варианты

1. Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)

2. Книга (название книги, автор, издательство, количество страниц)

3. Файл (имя, время создания, дата создания, размер)

4. Спортсмен (ФИО, вид спорта, сколько лет занимается)

5. Студент (ФИО, группа, средний балл зачетки)

6. Фирма (название, вид услуг, годовой оборот)

7. Компьютер (производитель, частота процессора, количество ядер, объем памяти)

8. Спорт (вид спорта, количество занимающихся в городе, средняя стоимость комплекта инвентаря)

9. Автомобиль (марка, фирма-производитель, год выпуска, пробег)

10. Статья (название, автор, название журнала, год и месяц издания)

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 6** Функции и перечисления.

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы:** Изучить тему функций, научиться создавать и применять их.

Краткие теоретические сведения

Функция – это набор операторов, организованных вместе для выполнения определенной задачи. Функция Swift может быть такой же простой, как простая функция C, и такой же сложной, как функция языка Objective-C. Это позволяет нам передавать локальные и глобальные значения параметров внутри вызовов функций.

• Объявление функции – сообщает компилятору имя функции, тип возвращаемого значения и параметры.

• Определение функции – предоставляет фактическое тело функции.

Объявление функции – сообщает компилятору имя функции, тип возвращаемого значения и параметры.

Определение функции – предоставляет фактическое тело функции.

Функции Swift содержат тип параметра и его возвращаемые типы.

Определение функции

В Swift функция определяется ключевым словом «func». Когда функция определена, она может принять одно или несколько значений в качестве входных «параметров» для функции, и она обработает функции в основном теле и передаст значения в качестве выходных «возвращаемых типов».

Каждая функция имеет имя, которое описывает задачу, выполняемую функцией. Чтобы использовать функцию, ее нужно вызвать по имени и передать входные значения (известные как аргументы), которые соответствуют типам параметров функции. Параметры функции также называются «кортежами».

Аргументы функции должны всегда указываться в том же порядке, что и список параметров функции, а за возвращаемыми значениями следует →.

Синтаксис

func funcname(Parameters) -> returntype {

Statement1

Statement2

---

Statement N

return parameters

}

Посмотрите на следующий код. Имя студента объявляется как строковый тип данных, объявленный внутри функции ‘student’, и когда функция вызывается, она возвращает имя студента.

func student(name: String) -> String {

return name

}

print(student(name: "First Program"))

print(student(name: "About Functions"))

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

First Program

About Functions

Вызов функции

Давайте предположим, что мы определили функцию с именем display для рассмотрения. Например, для отображения чисел функция с именем функции display отображается сначала с аргументом no1, который содержит целочисленный тип данных. Затем аргумент ‘no1’ присваивается аргументу ‘a’, который в дальнейшем будет указывать на тот же тип данных integer. Теперь аргумент «а» возвращается в функцию. Здесь функция display () будет содержать целочисленное значение и возвращать целочисленные значения при каждом вызове функции.

func display(no1: Int) -> Int {

let a = no1

return a

}

print(display(no1: 100))

print(display(no1: 200))

Когда мы запускаем программу, описанную выше, используя площадку, мы получаем следующий результат –

100

200

Параметры и возвращаемые значения

Swift предоставляет гибкие параметры функции и возвращаемые значения от простых до сложных значений. Как и в C и Objective-C, функции в Swift также могут иметь несколько форм.

Функции с параметрами

Доступ к функции осуществляется путем передачи значений ее параметров в тело функции. Мы можем передать одно или несколько значений параметров в виде кортежей внутри функции.

func mult(no1: Int, no2: Int) -> Int {

return no1\*no2

}

print(mult(no1: 2, no2: 20))

print(mult(no1: 3, no2: 15))

print(mult(no1: 4, no2: 30))

Когда мы запускаем программу выше, используя площадку, мы получаем следующий результат –

40

45

120

Функции без параметров

У нас также могут быть функции без каких-либо параметров.

Синтаксис

func funcname() -> datatype {

return datatype

}

Ниже приведен пример с функцией без параметра:

func votersname() -> String {

return "Alice"

}

print(votersname())

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Alice

Функции с возвращаемыми значениями

Функции также используются для возврата значений типа string, integer и float в качестве типов возврата. Чтобы узнать наибольшее и наименьшее число в данном массиве, функция ‘ls’ объявляется с длинными и короткими целочисленными типами данных.

Массив инициализируется для хранения целочисленных значений. Затем массив обрабатывается, и каждое значение в массиве читается и сравнивается для его предыдущими значениями. Когда следующее значение меньше предыдущего, оно сохраняется в аргументе ‘small’, в противном случае оно сохраняется в аргументе ‘large’, а значения возвращаются при вызове функции.

func ls(array: [Int]) -> (large: Int, small: Int) {

var lar = array[0]

var sma = array[0]

for i in array[1..<array.count] {

if i < sma {

sma = i

} else if i > lar {

lar = i

}

}

return (lar, sma)

}

let num = ls(array: [40,12,-5,78,98])

print("Largest number is: \(num.large) and smallest number is: \(num.small)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Largest number is: 98 and smallest number is: -5

Функции без возвращаемых значений

Некоторые функции могут только аргументы, объявленные внутри функции и не иметь возвращаемых значений. Следующая программа объявляет a и b в качестве аргументов функции sum (), внутрь самой функции значения аргументов a и b передаются путем вызова вызова функции sum (), и ее значения выводятся на печать, тем самым исключается необходимость возврата значений.

func sum(a: Int, b: Int) {

let a = a + b

let b = a - b

print(a, b)

}

sum(a: 20, b: 10)

sum(a: 40, b: 10)

sum(a: 24, b: 6)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

30 20

50 40

30 24

Перечисление – это определяемый пользователем тип данных, который состоит из набора связанных значений. Ключевое слово enum используется для определения перечислимого типа данных.

Функциональность перечисления

Перечисление в Swift также напоминает структуру C и Objective C.

• Он объявлен в классе, и его значения доступны через экземпляр этого класса.

• Начальное значение члена определяется с помощью инициализаторов enum.

• Его функциональность также расширена за счет обеспечения стандартных функций протокола.

Он объявлен в классе, и его значения доступны через экземпляр этого класса.

Начальное значение члена определяется с помощью инициализаторов enum.

Его функциональность также расширена за счет обеспечения стандартных функций протокола.

Синтаксис

Перечисления вводятся с ключевым словом enum и помещают все их определения в пару фигурных скобок –

enum enumname {

// enumeration values are described here

}

Например, вы можете определить перечисление для дней недели следующим образом:

enum DaysofaWeek {

case Sunday

case Monday

---

case Saturday

}

Пример:

enum names {

case Swift

case Closures

}

var lang = names.Closures

lang = .Closures

switch lang {

case .Swift:

print("Welcome to Swift")

case .Closures:

print("Welcome to Closures")

default:

print("Introduction")

}

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Welcome to Closures

Перечисление Swift не присваивает своим членам значение по умолчанию, такое как C и Objective C. Вместо этого члены явно определяются своими именами перечисления. Имя перечисления должно начинаться с заглавной буквы (например, enum DaysofaWeek).

var weekDay = DaysofaWeek.Sunday

Здесь имя перечисления ‘DaysofaWeek’ присваивается переменной дней недели. Воскресенье. Он сообщает компилятору, что тип данных, относящийся к Sunday, будет присвоен последующим членам enum этого конкретного класса. Как только тип данных элемента enum определен, к членам можно получить доступ путем передачи значений и дальнейших вычислений.

Перечисление с оператором Switch

Оператор Swift «Switch» также следует за множественным выбором. Доступ к одной переменной возможен в определенное время на основе указанного условия. Регистр по умолчанию в операторе switch используется для отлова неуказанных случаев.

enum Climate {

case India

case America

case Africa

case Australia

}

var season = Climate.America

season = .America

switch season {

case .India:

print("Climate is Hot")

case .America:

print("Climate is Cold")

case .Africa:

print("Climate is Moderate")

case .Australia:

print("Climate is Rainy")

}

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Climate is Cold

Сначала программа определяет Climate в качестве имени перечисления. Затем его члены, такие как «Индия», «Америка», «Африка» и «Австралия», объявляются принадлежащими к классу «Климат».Теперь член America назначен переменной сезона. Кроме того, Switch case увидит значения, соответствующие .America, и перейдет к этому конкретному оператору. Вывод будет отображаться как «Климат холодный». Точно так же все члены могут быть доступны через операторы switch. Если условие не выполняется, по умолчанию выводится «Климат не предсказуем».

Задание для самостоятельного выполнения

1. Написать функцию, которая увеличивает числовое значение бонуса на заданное число для покупателя при его участии в той или иной акции, результат присвоить в переменную и вывести на экран, предварительно показать исходное значение; список акций и начисляемых за них бонусов сформировать отдельно и вывести на экран.

2. Написать функцию перевода бонусов в рубли и показать сумму скидки в рублях. (Формула произвольна, на Ваше усмотрение).

3. Написать функцию, показывающую количество получаемых наклеек в зависимости от числа имеющихся бонусов и возможные призы в акции, а также сколько еще наклеек надо собрать, чтобы получить каждый из них.

**Ход работы:**

Задание 1: Функция увеличения бонуса для покупателя при участии в акции

Создадим список акций и начисляемых за них бонусов:

// Список акций и начисляемых бонусов

enum Promotion {

case discount(code: String, bonus: Int)

case freeProduct(code: String, productName: String)

}

let promotions: [Promotion] = [

.discount(code: "SUMMER", bonus: 50),

.discount(code: "FALL", bonus: 30),

.freeProduct(code: "WINTER", productName: "Sticker"),

.freeProduct(code: "SPRING", productName: "Keychain")

]

// Вывод списка акций и начисляемых бонусов

print("Список акций и начисляемых бонусов:")

for promotion in promotions {

switch promotion {

case .discount(let code, let bonus):

print("Акция \"\(code)\": Дополнительные бонусы \(bonus)")

case .freeProduct(let code, let productName):

print("Акция \"\(code)\": Бесплатный подарок - \(productName)")

}

}

Напишем функцию, которая будет увеличивать числовое значение бонуса на заданное число для покупателя при его участии в акции:

// Функция для увеличения бонуса при участии в акции

func increaseBonus(currentBonus: inout Int, by amount: Int) {

currentBonus += amount

}

// Пример использования функции

var customerBonus = 100

print("Исходное значение бонуса: \(customerBonus)")

increaseBonus(currentBonus: &customerBonus, by: 50)

print("Значение бонуса после участия в акции: \(customerBonus)")

Нажмите кнопку "Run" (или используйте комбинацию клавиш Command+R), чтобы выполнить код.

Задание 2: Функция перевода бонусов в рубли

Напишем функцию, которая будет переводить бонусы в рубли и показывать сумму скидки в рублях:

// Функция для перевода бонусов в рубли

func convertBonusToRubles(bonus: Int, conversionRate: Double) -> Double {

return Double(bonus) \* conversionRate

}

// Пример использования функции

let bonusAmount = 100

let rubles = convertBonusToRubles(bonus: bonusAmount, conversionRate: 0.5)

print("Сумма скидки в рублях: \(rubles)")

Задание 3: Функция для расчета количества получаемых наклеек и призов в зависимости от бонусов

Напишем функцию, которая будет показывать количество получаемых наклеек в зависимости от числа имеющихся бонусов и возможные призы в акции, а также сколько еще наклеек надо собрать, чтобы получить каждый из них:

// Функция для расчета количества получаемых наклеек и призов

func calculateStickersAndPrizes(bonus: Int) {

let stickers = bonus / 10

let remainingBonus = bonus % 10

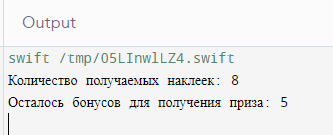
print("Количество получаемых наклеек: \(stickers)")

print("Осталось бонусов для получения приза: \(10 - remainingBonus)")

}

// Пример использования функции

calculateStickersAndPrizes(bonus: 85)



**Контрольные вопросы:**

1. Какие типы акций могут быть в списке, и какие данные они содержат?
2. Каким образом можно использовать перечисления для описания списка акций?
3. Как можно увеличить значение переменной внутри функции в Swift?
4. Каким образом можно определить функцию для перевода бонусов в рубли?
5. Как можно разделить количество получаемых наклеек и призов при заданном количестве бонусов?

Варианты

1. Магазин «Магнит»

2. Магазин «Перекресток»

3. Магазин «Marmalato»

4. Магазин «1000 и 1 бутылка»

5. Авиалинии, бонусы за километры

6. Служба «Такси»

7. Служба ремонта «Почини», скидки при последующих заказах

8. Кафе «У друзей»

9. Турфирма «Пегас»

10. Кинотеатр «Рассвет»

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 7** Структуры и классы. Создание, инициализация

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы:** изучить создание и инициализацию классов и структур, научиться применять на практике.

**Краткие теоретические сведения**

Классы в Swift являются строительными блоками гибких конструкций. Подобно константам, переменным и функциям пользователь может определять свойства и методы класса. Swift предоставляет нам функциональность, с помощью которой, при объявлении классов пользователям не нужно создавать интерфейсы или файлы реализации. Swift позволяет нам создавать классы в виде одного файла, и внешние интерфейсы будут создаваться по умолчанию после инициализации классов.

Преимущества:

• Наследование передает свойства одного класса другому классу

• Приведение типов позволяет пользователю проверять тип класса во время выполнения

• Деинициализаторы заботятся об освобождении ресурсов памяти

• Подсчет ссылок позволяет экземпляру класса иметь более одной ссылки

Общие характеристики классов и структур

• Свойства определены для хранения значений

• Индексы определены для обеспечения доступа к значениям

• Методы инициализируются для улучшения функциональности

• Начальное состояние определяется инициализаторами

• Функциональность расширена за пределы значений по умолчанию

• Подтверждение стандартов функциональности протокола

Синтаксис

Class classname {

Definition 1

Definition 2

---

Definition N

}

Определение класса

class student {

var studname: String

var mark: Int

var mark2: Int

}

Синтаксис для создания экземпляров

let studrecord = student()

Пример:

class MarksStruct {

var mark: Int

init(mark: Int) {

self.mark = mark

}

}

class studentMarks {

var mark = 300

}

let marks = studentMarks()

print("Mark is \(marks.mark)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Mark is 300

Доступ к свойствам класса в качестве ссылочных типов

Свойства класса могут быть доступны с помощью «.» синтаксис. Имя свойства отделяется знаком «.» после имени экземпляра.

class MarksStruct {

var mark: Int

init(mark: Int) {

self.mark = mark

}

}

class studentMarks {

var mark1 = 300

var mark2 = 400

var mark3 = 900

}

let marks = studentMarks()

print("Mark1 is \(marks.mark1)")

print("Mark2 is \(marks.mark2)")

print("Mark3 is \(marks.mark3)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Mark1 is 300

Mark2 is 400

Mark3 is 900

Swift предоставляет гибкие конструкции для использования структур. Применяя структуры, можно определить методы и свойства конструкций.

В отличие от C и Objective C

• Структура не требует файлов реализации и интерфейса.

• Структура позволяет нам создавать один файл и автоматически расширять его интерфейс для других блоков.

В Структуре значения переменных копируются и передаются в последующих кодах путем возврата копии старых значений, чтобы значения не могли быть изменены.

Структуры описываются с помощью ключевого слова 'Struct'

Синтаксис

struct nameStruct {

Definition 1

Definition 2

---

Definition N

}

Определение структуры

Рассмотрим пример, предположим, что мы должны получить доступ к записям студентов, содержащим оценки трех предметов, и узнать три предмета. Здесь markStruct используется для инициализации структуры с тремя отметками как тип данных «Int».

struct MarkStruct {

var mark1: Int

var mark2: Int

var mark3: Int

}

Доступ к структуре и ее свойствам

Члены структуры доступны по имени структуры. Экземпляры структуры инициализируются ключевым словом let.

struct studentMarks {

var mark1 = 100

var mark2 = 200

var mark3 = 300

}

let marks = studentMarks()

print("Mark1 is \(marks.mark1)")

print("Mark2 is \(marks.mark2)")

print("Mark3 is \(marks.mark3)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Mark1 is 100

Mark2 is 200

Mark3 is 300

Доступ к оценкам учащихся осуществляется по структуре «studentMarks». Элементы структуры инициализируются как mark1, mark2, mark3 со значениями целочисленного типа. Затем структура studentMarks () передается метке «mark» с ключевым словом «let». В дальнейшем «отметки» будут содержать значения элементов структуры. Теперь значения печатаются путем доступа к значениям элементов структуры с помощью «.» с его инициализированными именами.

struct MarksStruct {

var mark: Int

init(mark: Int) {

self.mark = mark

}

}

var aStruct = MarksStruct(mark: 98)

var bStruct = aStruct // aStruct and bStruct are two structs with the same value!

bStruct.mark = 97

print(aStruct.mark) // 98

print(bStruct.mark) // 97

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

98

97

Лучшие практики использования конструкций

Язык Swift обеспечивает функциональность для определения структур как пользовательских типов данных для построения функциональных блоков. Экземпляры структуры передаются по значению в определенные блоки для дальнейших манипуляций.

Нужно иметь структуры

• Инкапсулировать простые значения данных.

• Копировать инкапсулированные данные и связанные с ними свойства по «значениям», а не «ссылкам».

• Структура для «Копировать» и «Ссылка».

Структуры в Swift передают свои члены со своими значениями, а не ссылки.

struct markStruct {

var mark1: Int

var mark2: Int

var mark3: Int

init(mark1: Int, mark2: Int, mark3: Int) {

self.mark1 = mark1

self.mark2 = mark2

self.mark3 = mark3

}

}

var marks = markStruct(mark1: 98, mark2: 96, mark3:100)

print(marks.mark1)

print(marks.mark2)

print(marks.mark3)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

98

96

100

Другой пример

struct markStruct {

var mark1: Int

var mark2: Int

var mark3: Int

init(mark1: Int, mark2: Int, mark3: Int) {

self.mark1 = mark1

self.mark2 = mark2

self.mark3 = mark3

}

}

var fail = markStruct(mark1: 34, mark2: 42, mark3: 13)

print(fail.mark1)

print(fail.mark2)

print(fail.mark3)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

34

42

13

Задание для самостоятельного выполнения

1. Создать, инициализировать и вывести на экран классы в соответствии с вариантом задания

2. Создать, инициализировать и вывести на экран структуры в соответствии с вариантом задания

3. Описать структуры на языке Swift, инициализировать поля, создать массив значений из структур, ввести значение для поиска и вывести прочие поля найденного элемента значения

Ход работы

**Ход работы:**

Задание 1: Создание и инициализация классов

Создадим класс "Автомобиль" с соответствующими свойствами:

class Car {

var brand: String

var model: String

var year: Int

init(brand: String, model: String, year: Int) {

self.brand = brand

self.model = model

self.year = year

}

func displayInfo() {

print("Марка: \(brand), Модель: \(model), Год выпуска: \(year)")

}

}

// Создание и инициализация объекта класса "Автомобиль"

let myCar = Car(brand: "Toyota", model: "Camry", year: 2022)

// Вывод информации об автомобиле на экран

print("Информация о моем автомобиле:")

myCar.displayInfo()

Задание 2: Создание и инициализация структур

Создадим структуру "Книга" с соответствующими свойствами:

struct Book {

var title: String

var author: String

var pages: Int

init(title: String, author: String, pages: Int) {

self.title = title

self.author = author

self.pages = pages

}

func displayInfo() {

print("Название: \(title), Автор: \(author), Количество страниц: \(pages)")

}

}

// Создание и инициализация объекта структуры "Книга"

let myBook = Book(title: "Swift Programming", author: "John Doe", pages: 300)

// Вывод информации о книге на экран

print("Информация о книге:")

myBook.displayInfo()

Задание 3: Описание структур и поиск элемента

Опишем структуры "Студент" и "Группа" на языке Swift, инициализируем поля, создадим массив значений из структур, введем значение для поиска и выведем прочие поля найденного элемента значения:

struct Student {

var name: String

var age: Int

var group: Group

init(name: String, age: Int, group: Group) {

self.name = name

self.age = age

self.group = group

}

}

struct Group {

var name: String

var studentsCount: Int

init(name: String, studentsCount: Int) {

self.name = name

self.studentsCount = studentsCount

}

}

// Создание объектов структур "Группа"

let group1 = Group(name: "Group A", studentsCount: 25)

let group2 = Group(name: "Group B", studentsCount: 30)

let group3 = Group(name: "Group C", studentsCount: 28)

// Создание объектов структур "Студент"

let students: [Student] = [

Student(name: "Alice", age: 20, group: group1),

Student(name: "Bob", age: 22, group: group2),

Student(name: "Charlie", age: 21, group: group3)

]

// Ввод значения для поиска

let searchName = "Bob"

// Поиск студента по имени

if let foundStudent = students.first(where: { $0.name == searchName }) {

print("Имя: \(foundStudent.name)")

print("Возраст: \(foundStudent.age)")

print("Группа: \(foundStudent.group.name)")

} else {

print("Студент с именем \(searchName) не найден.")

}

**Контрольные вопросы:**

1. Чем отличаются классы и структуры в Swift?

2. Что такое инициализаторы (иниты) и зачем они нужны?

3. Как создать экземпляр класса или структуры в Swift?

4. Какие основные свойства и методы может содержать класс или структура?

5. Как осуществить инициализацию объекта класса или структуры с помощью конструктора?

Варианты

1. Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту) Ввести номер маршрута

2. Книга (название книги, автор, издательство, количество страниц) Ввести название

3. Файл (имя, время создания, дата создания, размер) Ввести имя

4. Спортсмен (ФИО, вид спорта, сколько лет занимается) Ввести ФИО

5. Студент (ФИО, группа, средний балл зачетки) Ввести ФИО

6. Фирма (название, вид услуг, годовой оборот) Ввести название

7. Компьютер (модель, производитель, частота процессора, количество ядер, объем памяти) Ввести модель

8. Спорт (вид спорта, количество занимающихся в городе, средняя стоимость комплекта инвентаря) Ввести вид спорта

9. Автомобиль (марка, фирма-производитель, год выпуска, пробег) Ввести марку

10. Статья (название, автор, название журнала, год и месяц издания) Ввести название и автора

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 8** Методы экземпляров. Синтаксис работы с классами, self, имя внешнего параметра с символом # и \_, методы мутации структур.

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы**: изучить работу методов, научиться применять на практике.

Краткие теоретические сведения

В языке Swift функции, связанные с определенными типами, называются методами. В Objective C классы используются для определения методов, в то время как гибкий язык Swift предоставляет пользователю возможность иметь методы для классов, структур и перечислений.

Методы экземпляра

В языке Swift доступ к экземплярам Classes, Structures и Enumeration осуществляется через методы экземпляра.

Методы экземпляра обеспечивают функциональность

• Чтобы получить доступ и изменить свойства экземпляра

• функциональность, связанная с потребностью экземпляра

Метод экземпляра может быть записан в фигурных скобках {}. Он имеет неявный доступ к методам и свойствам экземпляра типа. Когда вызывается конкретный экземпляр типа, он получает доступ к этому конкретному экземпляру.

Синтаксис

func funcname(Parameters) -> returntype {

Statement1

Statement2

---

Statement N

return parameters

}

class calculations {

let a: Int

let b: Int

let res: Int

init(a: Int, b: Int) {

self.a = a

self.b = b

res = a + b

}

func tot(c: Int) -> Int {

return res - c

}

func result() {

print("Result is: \(tot(c: 20))")

print("Result is: \(tot(c: 50))")

}

}

let pri = calculations(a: 600, b: 300)

pri.result()

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Result is: 880

Result is: 850

Вычисления класса определяют два метода экземпляра –

• init () определен, чтобы сложить два числа a и b и сохранить их в результате ‘res’

• tot () используется, чтобы вычесть ‘res’ из переданного значения ‘c’

Наконец, для печати расчетов используются методы со значениями для a и b. Методы экземпляра доступны с помощью «.» точечный синтаксис

Локальные и внешние имена параметров

Функции Swift описывают как локальные, так и глобальные объявления для своих переменных. Точно так же соглашения об именах Swift методы также напоминают Objective C. Но характеристики локальных и глобальных объявлений имен параметров различаются для функций и методов. Первый параметр в Swift именуется предлогами как «with», «for» и «by» для простого доступа к соглашениям об именах.

Swift обеспечивает гибкость методов, объявляя имя первого параметра в качестве локальных имен параметров, а оставшиеся имена параметров – как глобальные имена параметров. Здесь ‘no1’ объявляется методами Swift как локальные имена параметров. «no2» используется для глобальных объявлений и доступен через программу.

class division {

var count: Int = 0

func incrementBy(no1: Int, no2: Int) {

count = no1 / no2

print(count)

}

}

let counter = division()

counter.incrementBy(no1: 1800, no2: 3)

counter.incrementBy(no1: 1600, no2: 5)

counter.incrementBy(no1: 11000, no2: 3)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

600

320

3666

Имя внешнего параметра с символом # и \_

Несмотря на то, что методы Swift предоставляют первые имена параметров для локальных объявлений, пользователь может изменить имена параметров с локальных на глобальные объявления. Это можно сделать, добавив префикс «#» к имени первого параметра. Таким образом, к первому параметру можно получить глобальный доступ по всем модулям.

Когда пользователю необходимо получить доступ к последующим именам параметров с помощью внешнего имени, имя метода переопределяется с помощью символа «\_».

class multiplication {

var count: Int = 0

func incrementBy(no1: Int, no2: Int) {

count = no1 \* no2

print(count)

}

}

let counter = multiplication()

counter.incrementBy(no1: 800, no2: 3)

counter.incrementBy(no1: 100, no2: 5)

counter.incrementBy(no1: 15000, no2: 3)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

2400

500

45000

Ссылка на текущий объект в методах

Методы имеют неявное свойство, известное как «self», для всех экземпляров определенного типа. Свойство «Self» используется для ссылки на текущие экземпляры для его определенных методов.

class calculations {

let a: Int

let b: Int

let res: Int

init(a: Int, b: Int) {

self.a = a

self.b = b

res = a + b

print("Inside Self Block: \(res)")

}

func tot(c: Int) -> Int {

return res - c

}

func result() {

print("Result is: \(tot(c: 20))")

print("Result is: \(tot(c: 50))")

}

}

let pri = calculations(a: 600, b: 300)

let sum = calculations(a: 1200, b: 300)

pri.result()

sum.result()

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Inside Self Block: 900

Inside Self Block: 1500

Result is: 880

Result is: 850

Result is: 1480

Result is: 1450

Изменение типов значений из методов экземпляра

В Swift языковые структуры и перечисления относятся к типам значений, которые не могут быть изменены его методами экземпляра. Тем не менее, язык Swift обеспечивает гибкость для изменения типов значений путем «мутации» поведения. Mutate внесет любые изменения в методы экземпляра и вернется к исходной форме после выполнения метода. Также с помощью свойства «self» создается новый экземпляр для его неявной функции, который заменит существующий метод после его выполнения.

struct area {

var length = 1

var breadth = 1

func area() -> Int {

return length \* breadth

}

mutating func scaleBy(res: Int) {

length \*= res

breadth \*= res

print(length)

print(breadth)

}

}

var val = area(length: 3, breadth: 5)

val.scaleBy(res: 3)

val.scaleBy(res: 30)

val.scaleBy(res: 300)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

9

15

270

450

81000

135000

Ссылка на текущий объект для метода мутации

Методы мутации в сочетании со свойством self присваивают новый экземпляр определенному методу.

struct area {

var length = 1

var breadth = 1

func area() -> Int {

return length \* breadth

}

mutating func scaleBy(res: Int) {

self.length \*= res

self.breadth \*= res

print(length)

print(breadth)

}

}

var val = area(length: 3, breadth: 5)

val.scaleBy(res: 13)

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат. –

39

65

Задание для самостоятельного выполнения

1. В соответствии со своим вариантом задания создать класс, инициализировать его, реализовать методы мутации для численного значения, которое может быть увеличено, без потери здравого смысла объекта.

2. Реализовать метод вывода на печать экземпляра класса

3. Создать массив из 5 элементов типа созданного класса.

4. Реализовать поиск нужного объекта в массиве и вывести его на экран

**Ход работы:**

Создание класса и инициализация: Создайте класс с нужными свойствами и методами. Инициализируйте экземпляр класса.

swift

Копировать код

class MyClass {

var value: Int

init(value: Int) {

self.value = value

}

func increase(by amount: Int) {

// Реализация метода мутации для увеличения значения

self.value += amount

}

func printValue() {

print("Значение: \(self.value)")

}

}

Реализация метода вывода на печать: Добавьте метод, который будет выводить значения свойств экземпляра класса на печать.

swift

Копировать код

class MyClass {

var value: Int

init(value: Int) {

self.value = value

}

func increase(by amount: Int) {

self.value += amount

}

func printValue() {

print("Значение: \(self.value)")

}

}

Создание массива из экземпляров класса: Создайте массив из нескольких экземпляров класса.

var myObjects: [MyClass] = []

for \_ in 1...5 {

let object = MyClass(value: Int.random(in: 1...100))

myObjects.append(object)

}

Поиск объекта в массиве и вывод на экран: Реализуйте поиск нужного объекта в массиве и вывод его на экран.

swift

Копировать код

let searchValue = 42 // Значение, которое мы ищем

if let foundObject = myObjects.first(where: { $0.value == searchValue }) {

foundObject.printValue()

} else {

print("Объект с заданным значением не найден.")

}

****

**Контрольные вопросы:**

1. Что такое методы экземпляров в Swift?
2. Какие аргументы могут принимать методы класса?
3. В чем отличие мутационных методов от обычных методов?
4. Как можно вызвать метод экземпляра класса?
5. Можно ли вызывать метод экземпляра класса без создания экземпляра?
6. Как реализовать метод, который изменяет значения свойств экземпляра класса?
7. Как можно использовать методы экземпляров для управления экземплярами класса в массиве?

Варианты

1. Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)

2. Книга (название книги, автор, издательство, количество страниц)

3. Файл (имя, время создания, дата создания, размер)

4. Спортсмен (ФИО, вид спорта, сколько лет занимается)

5. Студент (ФИО, группа, средний балл зачетки)

6. Фирма (название, вид услуг, годовой оборот)

7. Компьютер (производитель, частота процессора, количество ядер, объем памяти)

8. Спорт (вид спорта, количество занимающихся в городе, средняя стоимость комплекта инвентаря)

9. Автомобиль (марка, фирма-производитель, год выпуска, пробег)

10. Статья (название, автор, название журнала, год и месяц издания)

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 9** Инициализация и деинициализация, наследование, делегирование, назначенные и удобные инициализаторы

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы**: изучить работу инциализаторов, научиться применять на практике.

Краткие теоретические сведения

Классы, структуры и перечисления, однажды объявленные в Swift , инициализируются для подготовки экземпляра класса. Начальное значение инициализируется для сохраненного свойства, а также для новых экземпляров. Инициализируются значения, чтобы продолжить работу. Ключевое слово для создания функции инициализации выполняется методом init ().Инициализатор Swift отличается от Objective-C тем, что он не возвращает никаких значений. Его функция заключается в проверке инициализации вновь созданных экземпляров перед их обработкой. Swift также предоставляет процесс «деинициализации» для выполнения операций управления памятью после освобождения экземпляров.

Роль инициализатора для сохраненных свойств

Сохраненное свойство должно инициализировать экземпляры для своих классов и структур перед обработкой экземпляров. Хранимые свойства используют инициализатор для назначения и инициализации значений, тем самым устраняя необходимость вызова наблюдателей свойств.Инициализатор используется в хранимом свойстве

• Создать начальное значение.

• Чтобы назначить значение свойства по умолчанию в определении свойства.

• Для инициализации экземпляра для определенного типа данных используется init (). Внутри функции init () аргументы не передаются.

Синтаксис

init() {

//New Instance initialization goes here

}

Пример

struct rectangle {

var length:Double

var breadth:Double

init(){

length =6

breadth =12

}

}

var area = rectangle()

print("area of rectangle is \(area.length\*area.breadth)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

area of rectangle is 72.0

Здесь структура ‘rectangle’ инициализируется с длиной и шириной элементов как ‘Double’ типы данных. Метод Init () используется для инициализации значений для вновь создаваемых элементов длины и двойника. Площадь прямоугольника вычисляется и возвращается путем вызова функции прямоугольника.

Установка значений свойств по умолчанию

Язык Swift предоставляет функцию Init () для инициализации значений сохраненных свойств.Кроме того, пользователь может инициализировать значения свойств по умолчанию при объявлении членов класса или структуры. Когда свойство принимает одно и то же значение во всей программе, мы можем объявить его только в разделе объявления, а не инициализировать его в init (). Установка значений свойств по умолчанию позволяет пользователю, когда наследование определено для классов или структур.

struct rectangle {

var length =6

var breadth =12

}

var area = rectangle()

print("area of rectangle is \(area.length\*area.breadth)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

area of rectangle is 72

Здесь вместо объявления длины и ширины в init () значения инициализируются в самом объявлении.

Инициализация параметров

В языке Swift пользователь имеет возможность инициализировать параметры как часть определения инициализатора с помощью init ().

structRectangle{

var length:Double

var breadth:Double

var area:Double

init(fromLength length:Double, fromBreadth breadth:Double){

self.length = length

self.breadth = breadth

area = length \* breadth

}

init(fromLeng leng:Double, fromBread bread:Double){

self.length = leng

self.breadth = bread

area = leng \* bread

}

}

let ar =Rectangle(fromLength:6, fromBreadth:12)

print("area is: \(ar.area)")

let are =Rectangle(fromLeng:36, fromBread:12)

print("area is: \(are.area)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

area is: 72.0

area is: 432.0

Локальные и внешние параметры

Параметры инициализации имеют как локальные, так и глобальные имена параметров, аналогичные параметрам функций и методов. Объявление локального параметра используется для доступа в теле инициализации, а объявление внешнего параметра используется для вызова инициализатора. Инициализаторы Swift отличаются от инициализатора функций и методов тем, что не определяют, какой инициализатор используется для вызова каких функций.

Чтобы преодолеть это, Swift вводит автоматическое внешнее имя для каждого параметра в init ().Это автоматическое внешнее имя эквивалентно локальному имени, записанному перед каждым параметром инициализации.

structDays{

let sunday, monday, tuesday:Int

init(sunday:Int, monday:Int, tuesday:Int){

self.sunday = sunday

self.monday = monday

self.tuesday = tuesday

}

init(daysofaweek:Int){

sunday = daysofaweek

monday = daysofaweek

tuesday = daysofaweek

}

}

let week =Days(sunday:1, monday:2, tuesday:3)

print("Days of a Week is: \(week.sunday)")

print("Days of a Week is: \(week.monday)")

print("Days of a Week is: \(week.tuesday)")

let weekdays =Days(daysofaweek:4)

print("Days of a Week is: \(weekdays.sunday)")

print("Days of a Week is: \(weekdays.monday)")

print("Days of a Week is: \(weekdays.tuesday)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

Days of a Week is: 1

Days of a Week is: 2

Days of a Week is: 3

Days of a Week is: 4

Days of a Week is: 4

Days of a Week is: 4

Параметры без внешних имен

Когда внешнее имя не требуется для инициализации, подчеркивание ‘\_’ используется для переопределения поведения по умолчанию.

structRectangle{

var length:Double

init(frombreadth breadth:Double){

length = breadth \*10

}

init(frombre bre:Double){

length = bre \*30

}

init(\_ area:Double){

length = area

}

}

let rectarea =Rectangle(180.0)

print("area is: \(rectarea.length)")

let rearea =Rectangle(370.0)

print("area is: \(rearea.length)")

let recarea =Rectangle(110.0)

print("area is: \(recarea.length)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

area is: 180.0

area is: 370.0

area is: 110.0

Необязательные типы недвижимости

Когда хранимое свойство в каком-то экземпляре не возвращает никакого значения, это свойство объявляется с необязательным типом, указывающим, что для этого конкретного типа не возвращается «никакого значения». Когда сохраненное свойство объявляется как «необязательное», оно автоматически инициализирует значение «nil» во время самой инициализации.

structRectangle{

var length:Double?

init(frombreadth breadth:Double){

length = breadth \*10

}

init(frombre bre:Double){

length = bre \*30

}

init(\_ area:Double){

length = area

}

}

let rectarea =Rectangle(180.0)

print("area is: \(rectarea.length)")

let rearea =Rectangle(370.0)

print("area is: \(rearea.length)")

let recarea =Rectangle(110.0)

print("area is: \(recarea.length)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

area is: Optional(180.0)

area is: Optional(370.0)

area is: Optional(110.0)

Изменение свойств константы во время инициализации

Инициализация также позволяет пользователю изменять значение постоянного свойства. Во время инициализации свойство класса позволяет его экземплярам класса изменяться суперклассом, а не подклассом. Рассмотрим, к примеру, в предыдущей программе «длина» объявлена как «переменная» в основном классе. Приведенная ниже программная переменная «length» модифицируется как «постоянная» переменная.

structRectangle{

let length:Double?

init(frombreadth breadth:Double){

length = breadth \*10

}

init(frombre bre:Double){

length = bre \*30

}

init(\_ area:Double){

length = area

}

}

let rectarea =Rectangle(180.0)

print("area is: \(rectarea.length)")

let rearea =Rectangle(370.0)

print("area is: \(rearea.length)")

let recarea =Rectangle(110.0)

print("area is: \(recarea.length)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

area is: Optional(180.0)

area is: Optional(370.0)

area is: Optional(110.0)

Инициализаторы по умолчанию

Инициализаторы по умолчанию предоставляют новый экземпляр всем его объявленным свойствам базового класса или структуры со значениями по умолчанию.

class defaultexample {

var studname:String?

var stmark =98

varpass=true

}

var result = defaultexample()

print("result is: \(result.studname)")

print("result is: \(result.stmark)")

print("result is: \(result.pass)")

Когда мы запускаем вышеуказанную программу, используя площадку, мы получаем следующий результат –

result is: nil

result is: 98

result is: true

Вышеуказанная программа определена с именем класса как «defaultexample». Три функции-члена по умолчанию инициализируются как ‘studname?’ хранить значения ‘nil’, ‘stmark’ как 98 и ‘pass’ как логическое значение ‘true’. Аналогично значения членов в классе могут быть инициализированы как значения по умолчанию перед обработкой типов членов класса.

Назначенные и вспомогательные инициализаторы в действии

Основной (базовый) класс называется Food, который имеет одно простое свойство типа String, называемое name, и обеспечивает два инициализатора для создания экземпляров класса Food:

class Food {

var name: String

init(name: String) {

self.name = name

}

convenience init() {

self.init(name: "[Unnamed]")

}

}

Схема ниже показывает цепочку работы инициализаторов в классе Food :

Классы по умолчанию не имеют почленного инициализатора, так что класс Food предоставляет назначенный инициализатор, который принимает единственный аргумент name. Этот инициализатор может быть использован для создания экземпляра Food со специфичным именем:

let namedMeat = Food(name: "Бекон")

//имя namedMeat является "Бекон"

Инициализатор init(name: String) из класса Food, представлен в виде назначенного инициализатора, потому что он проверяет, что все хранимые свойства нового экземпляра Food полностью инициализированы. Класс Food не имеет суперкласса, так что инициализатор init(name: String) не имеет вызова super.init() для завершения своей инициализации.

Класс Food так же обеспечивает вспомогательный инициализатор init() без аргументов. Инициализатор init() предоставляет имя плейсхолдера для новой еды, делегируя к параметру name инициализатора init(name: String), давая ему значение [Unnamed] :

let mysteryMeat = Food()

//mysteryMeat называется "[Unnamed]"

Второй класс в иерархии - это подкласс RecipeIngredient класса Food. Класс RecipeIngredient создает модель ингредиентов в рецепте. Он представляет свойство quantity типа Int (в дополнение к свойству name, унаследованное от Food) и определяет два инициализатора для создания экземпляров RecipeIngredient :

class RecipeIngredient: Food{

var quantity: Int

init(name: String, quantity: Int) {

self.quantity = quantity

super.init(name: name)

}

override convenience init(name: String) {

self.init(name: name, quantity: 1)

}

}

Схема ниже показывает цепочку инициализаторов для класса RecipeIngredient:

Все три инициализатора могут быть использованы для создания новых RecipeIngredient экземпляров:

let oneMysteryItem = RecipeIngredient()

let oneBacon = RecipeIngredient(name: "Bacon")

let sixEggs = RecipeIngredient(name: "Eggs", quantity: 6)

Пример1

Class User {

Var age: Int

Var name: String

init(){

age = 22

name = "Tom"

}

init(name: String, age: Int){

self.age = age

self.name = name

}

func getUserInfo(){

print("Имя: \(self.name); возраст: \(self.age)")

}

}

var bob: User = User(name: "Bob", age: 34)

bob.getUserInfo() // Имя: Bob; возраст: 34

Второй инициализатор принимает два параметра name и age для установки свойств класса. Так как параметры и свойства класса называются одинаково, то для их разграничения вместе с названиями свойств используется ключевое слово self.

И при создании объекта в данном случае используется второй инициализатор: var bob: User = User(name: "Bob", age: 34)

Инициализаторы могут определять значения по умолчанию для параметров. Так, пример выше мы могли бы сократить следующим образом:

Class User {

Var age: Int

Var name: String

Init (name: String= "Tom", age: Int= 22){

self.age = age

self.name = name

}

Func getUserInfo(){

print("Имя: \(self.name); возраст: \(self.age)")

}

}

Var tom = User()

tom.getUserInfo() // Имя: Tom; возраст: 22

Задание для самостоятельного выполнения:

Создайте класс или структуру "Person" с сохраненными свойствами "name" (имя) и "age" (возраст).

Реализуйте инициализатор, который принимает имя и возраст и присваивает их соответствующим свойствам.

Создайте экземпляр класса или структуры "Person" с вашим именем и возрастом.

Реализуйте метод вывода информации о персоне на экран.

**Ход работы:**

Создание класса или структуры "Person":

class Person {

var name: String

var age: Int

init(name: String, age: Int) {

self.name = name

self.age = age

}

func printInfo() {

print("Имя: \(name), Возраст: \(age)")

}

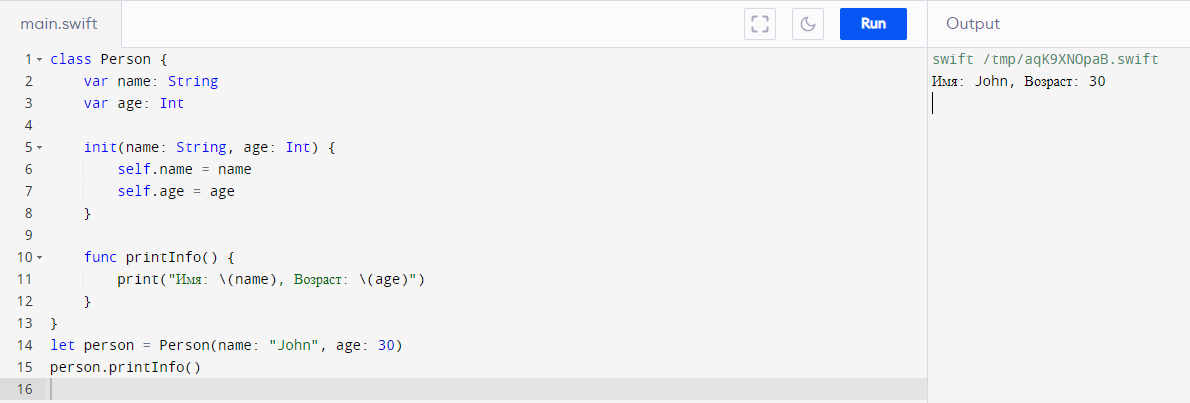
}

Создание экземпляра класса "Person":

let person = Person(name: "John", age: 30)

Вызов метода вывода информации о персоне:

person.printInfo()



**Контрольные вопросы:**

1. Что такое инициализатор?
2. Какие свойства должны быть инициализированы в инициализаторе?
3. Можно ли создать несколько инициализаторов для одного класса или структуры?
4. Как передаются аргументы в инициализатор?
5. Что такое хранимые свойства?
6. Какие ключевые слова используются при объявлении и реализации инициализатора?
7. Как можно вызвать инициализатор при создании экземпляра класса или структуры?
8. Как использовать инициализатор для задания значений свойств по умолчанию?
9. Чем отличается инициализатор класса от инициализатора структуры?
10. Как вызвать метод объекта в Swift?

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическая работа № 10** Замыкания. Функции замыкания, захват значений, удобные инициализаторы

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы**: изучить работу замыканий, научиться применять их на практике.

Краткие теоретические сведения

Замыкания — это самодостаточные блоки с определенным функционалом, которые могут быть переданы и использованы в вашем коде. Замыкания в Swift похожи на блоки в C и Objective-C, и лямбды в других языках программирования.

Замыкания могут захватывать и хранить ссылки на любые константы и переменные из контекста, в котором они объявлены. Эта процедура известна как заключение этих констант и переменных, отсюда и название "замыкание". Swift выполняет всю работу с управлением памятью при захвате за вас.

Глобальные и вложенные функцииявляются частным случаем замыканий. Замыкания принимают одну из трех форм:

• Глобальные функции являются замыканиями, у которых есть имя и которые не захватывают никакие значения.

• Вложенные функции являются замыканиями, у которых есть имя и которые могут захватывать значения из включающей их функции.

• Выражения замыкания являются безымянными замыканиями, написанные в облегченном синтаксисе, которые могут захватывать значения из их окружающего контекста.

Выражения замыкания в Swift имеют четкий, ясный, оптимизированный синтаксис в распространенных сценариях. Эти оптимизации включают:

• Вывод типа параметра и возврат типа значения из контекста

• Неявные возвращающиеся значения однострочных замыканий

• Сокращенные имена параметров

• Синтаксис последующих замыканий

Замыкания могут захватывать константы и переменные из окружающего контекста, в котором они объявлены. После захвата замыкание может ссылаться или модифицировать значения этих констант и переменных внутри своего тела, даже если область, в которой были объявлены эти константы и переменные уже больше не существует.

В Swift самая простая форма замыкания может захватывать значения из вложенных функций, написанных внутри тела других функций. Вложенная функция может захватить любые значения из аргументов окружающей ее функции, а так же константы и переменные, объявленные внутри тела внешней функции.

Вот пример функции makeIncrementer, которая содержит вложенную функцию incrementer. Вложенная функция incrementer() захватывает два значения runningTotal и amount из окружающего контекста. После захвата этих значений incrementer возвращается функцией makeIncrementer как замыкание, которое увеличивает runningTotal на amount каждый раз как вызывается.

func makeIncrementer(forIncrement amount: Int) -> () ->Int {

var runningTotal = 0

func incrementer() ->Int {

runningTotal += amount

return runningTotal

}

return incrementer

}

Возвращаемый тип makeIncrementer Void -> Int. Это значит, что он возвращает функцию, а не простое значение. Возвращенная функция не имеет параметров и возвращает Int каждый раз как ее вызывают.

Функция incrementer() не имеет ни одного параметра и она ссылается на runningTotal и amount внутри тела функции. Она делает это, захватывая существующие значения от runningTotal и amount из окружающей функции и используя их внутри. Захват ссылки дает гарантию того, что runningTotal не исчезнет при окончании вызова makeIncrementer и гарантирует, что runningTotal останется переменной в следующий раз, когда будет вызвана функция incrementer().

let incrementByTen = makeIncrementer(forIncrement: 10)

Этот пример заставляет константу incrementByTen ссылаться на функцию инкрементора, которая добавляет 10 к значению переменной runningTotal каждый раз как вызывается. Многократный вызов функции показывает ее в действии:

incrementByTen()

// возвращает 10

incrementByTen()

// возвращает 20

incrementByTen()

// возвращает 30

Если вы создаете второй инкрементор, он будет иметь свою собственную ссылку на новую отдельную переменную runningTotal :

let incrementBySeven = makeIncrementer(forIncrement: 7)

incrementBySeven()

//возвращает значение 7

Повторный вызов первоначального инкрементора( incrementByTen ) заставит увеличиваться его собственную переменную runningTotal и никак не повлияет на переменную, захваченную в incrementBySeven :

incrementByTen()

//возвращает 40

Заметка

Если вы присваиваете замыкание свойству экземпляра класса, и замыкание захватывает этот экземпляр по ссылке на него или его члены, вы создаете сильные обратные связи между экземпляром и замыканием. Swift использует списки захвата, для разрыва этих сильных обратных связей.

Замыкающие выражения

Вложенные функции являются удобным способом для обозначения и объявления самоорганизованных блоков кода, которые являются частью более крупной функции. Тем не менее, иногда полезно писать короткие версии функциональных конструкций, без полного объявления и указания имени. Это особенно верно, когда вы работаете с функциями, которые принимают другие функции в виде одного из своих параметров.

Замыкающие выражения, являются способом написания встроенных замыканий через краткий и специализированный синтаксис. Замыкающие выражения обеспечивают несколько синтаксических оптимизаций для написания замыканий в краткой форме, без потери ясности и намерений. Примеры замыкающих выражений ниже, показывают эти оптимизации путем рассмотрения метода sorted(by:) при нескольких итерациях, каждая из которых изображает ту же функциональность в более сжатой форме.

Метод sorted

В стандартной библиотеке Swift есть метод sorted(by:), который сортирует массив значений определенного типа, основываясь на результате сортирующего замыкания, которые вы ему передадите. После завершения процесса сортировки, метод sorted(by:) возвращает новый массив того же типа и размера как старый, с элементами в правильном порядке сортировки. Исходный массив не изменяется методом sorted(by:).

Примеры замыкающих выражений ниже используют метод sorted(by:)для сортировки массива из String значений в обратном алфавитном порядке. Вот исходный массив для сортировки:

let names = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]

Замыкание метода sorted(by:) принимает два аргумента одного и того же типа, что и содержимое массива, и возвращает Bool значение, которое решает поставить ли первое значение перед вторым, или после второго. Замыкание сортировки должно вернуть true, если первое значение должно быть до второго значения, и false в противном случае.

Этот пример сортирует массив из String значений, так что сортирующее замыкание должно быть функцией с типом

(String, String) ->Bool.

Один из способов обеспечить сортирующее замыкание, это написать нормальную функцию нужного типа, и передать его в качестве аргумента метода sorted(by:):

func backward(\_ s1: String, \_ s2: String) ->Bool {

return s1 > s2

}

var reversedNames = names.sorted(by: backward)

// reversedNamesравен ["Ewa", "Daniella", "Chris", "Barry", "Alex"]

Если первая строка (s1) больше чем вторая строка (s2), функция backward(\_:\_:) возвращает true, что указывает, что s1 должна быть перед s2 в сортированном массиве. Для символов в строках, "больше чем" означает "появляется в алфавите позже, чем". Это означает что буква "B" "больше чем" буква "А", а строка "Tom" больше чем строка "Tim". Это делает обратную алфавитную сортировку, с "Barry" поставленным перед "Alex", и так далее.

Синтаксис замыкающего выражения имеет следующую общую форму:

1. { (параметры) -> тип результата in

2. выражения

3. }

Синтаксис замыкающего выражения может использовать сквозные параметры. Значения по умолчанию не могут быть переданы. Вариативные параметры могут быть использованы в любом месте в списке параметров. Кортежи также могут быть использованы как типы параметров и как типы возвращаемого значения.

Пример ниже показывает версию функции backward(\_:\_:) с использованием замыкающего выражения:

reversedNames = names.sorted(by: { (s1: String, s2: String) ->Boolin

return s1 > s2

})

Обратите внимание, что объявление типов параметров и типа возвращаемого значения для этого однострочного замыкания идентично объявлению из функции backward(\_:\_:). В обоих случаях, оно пишется в виде (s1:String, s2: String) ->Bool. Тем не менее, для однострочных замыкающих выражений, параметры и тип возвращаемого значения пишутся внутри фигурных скобок, а не вне их.

Начало тела замыкания содержит ключевое слово in. Это ключевое слово указывает, что объявление параметров и возвращаемого значения замыкания закончено, и тело замыкания вот-вот начнется.

Поскольку тело замыкания настолько короткое, оно может быть записано в одну строку:

reversedNames = names.sorted(by: { (s1: String, s2: String) ->Boolinreturn s1 > s2 })

Определение типа из контекста

Поскольку сортирующее замыкание передается как аргумент метода, Swift может вывести типы его параметров и тип возвращаемого значения, через тип параметра метода sorted(by:). Этот параметр ожидает функцию имеющую тип (String, String) ->Bool. Это означает что типы (String, String) и Bool не нужно писать в объявлении замыкающего выражения. Поскольку все типы могут быть выведены, стрелка результата ( -> ) и скобки вокруг имен параметров также могут быть опущены:

reversedNames = names.sorted(by: { s1, s2 inreturn s1 > s2 })

Тем не менее, вы всё равно можете явно указать типы, если хотите. И делать это предполагается, если это поможет избежать двусмысленности для читателей вашего кода.

Неявные возвращаемые значения из замыканий с одним выражением

Замыкания с одним выражением могут неявно возвращать результат своего выражения через опускание ключевого слова return из их объявления, как показано в этой версии предыдущего примера:

reversedNames = names.sorted(by: { s1, s2 in s1 > s2 })

Здесь, функциональный тип аргумента метода sorted(by:)дает понять, что замыкание вернет Bool значение. Поскольку тело замыкания содержит одно выражение (s1 > s2), которое возвращает Bool значение, то нет никакой двусмысленности, и ключевое слово return можно опустить.

Сокращенные имена параметров

Swift автоматически предоставляет сокращённые имена для однострочных замыканий, которые могут быть использованы для обращения к значениям параметров замыкания через имена $0, $1, $2, и так далее.

Если вы используете эти сокращенные имена параметров с вашим замыкающим выражением, вы можете пропустить список параметров замыкания из его объявления, а количество и тип сокращенных имен параметров будет выведено из ожидаемого типа метода. Ключевое слово in также может быть опущено, поскольку замыкающее выражение полностью состоит из его тела:

reversedNames = names.sorted(by: { $0> $1 })

Здесь, $0 и $1 обращаются к первому и второму String параметру замыкания.

Операторные функции

Здесь есть на самом деле более короткий способ написать замыкающее выражение выше. Тип String в Swift определяет свою специфичную для строк реализацию оператора больше ( > ) как функции, имеющей два строковых параметра и возвращающей значение типа Bool. Это точно соответствует типу метода, для параметра метода sorted(by:). Таким образом, вы можете просто написать оператор больше, а Swift будет считать, что вы хотите использовать специфичную для строк реализацию:

reversedNames = names.sorted(by: >)

Автозамыкания (autoclosures)

Автозамыкания - замыкания, которые автоматически создаются для заключения выражения, которое было передано в качестве аргумента функции. Такие замыкания не принимают никаких аргументов при вызове и возвращают значение выражения, которое заключено внутри нее. Синтаксически вы можете опустить круглые скобки функции вокруг параметров функции, просто записав обычное выражение вместо явного замыкания.

Нет ничего необычного в вызове функций, которые принимают автозамыкания, но необычным является реализовывать такие функции. Например, функция assert(condition:message:file:line:) принимает автозамыкания на место condition и message параметров. Ее параметр condition вычисляется только в сборке дебаггера, а параметр message вычисляется, если только condition равен false.

Автозамыкания позволяют вам откладывать вычисления, потому как код внутри них не исполняется, пока вы сами его не запустите. Это полезно для кода, который может иметь сторонние эффекты или просто является дорогим в вычислительном отношении, потому что вы можете контролировать время исполнения этого кода. Пример ниже отображает как замыкания откладывают вычисления:

var customersInLine = ["Chris", "Alex", "Ewa", "Barry", "Daniella"]

print(customersInLine.count)

// Выведет "5"

let customerProvider = { customersInLine.remove(at: 0) }

print(customersInLine.count)

// Выведет "5"

print("Now serving \(customerProvider())!")

// Выведет "Now serving Chris!"

print(customersInLine.count)

// Выведет "4"

Даже если первый элемент массива customersInLine удаляется кодом внутри замыкания, элемент массива фактически не удаляется до тех пор пока само замыкание не будет вызвано. Если замыкание так и не вызывается, то выражение внутри него никогда не выполнится и, соответственно, элемент не будет удален из массива. Обратите внимание, что customerProvider является не String, а () ->String, то есть функция не принимает аргументов, но возвращает строку. Вы получите то же самое поведение, когда сделаете это внутри функции:

// customersInLineравен ["Alex", "Ewa", "Barry", "Daniella"]

func serve(customer customerProvider: () ->String) {

print("Now serving \(customerProvider())!")

}

serve(customer: { customersInLine.remove(at: 0) } )

// Выведет "NowservingAlex!"

Функция serve(customer:) описанная выше принимает явное замыкание, которое возвращает имя клиента. Версия функции serve(customer:) ниже выполняет ту же самую операцию, но вместо использования явного замыкания, она использует автозамыкание, поставив маркировку при помощи атрибута @autoclosure. Теперь вы можете вызывать функцию, как будто бы она принимает аргумент String вместо замыкания. Аргумент автоматически преобразуется в замыкание, потому что тип параметра customerProvider имеет атрибут @autoclosure.

// customersInLineравен ["Ewa", "Barry", "Daniella"]

func serve(customer customerProvider: @autoclosure () ->String) {

print("Now serving \(customerProvider())!")

}

serve(customer: customersInLine.remove(at: 0))

// Выведет "NowservingEwa!"

Заметка

Слишком частое использование автозамыканий может сделать ваш код сложным для чтения. Контекст и имя функции должны обеспечивать ясность отложенности исполнения кода.

Если вы хотите чтобы автозамыкание могло сбежать, то вам нужно использовать оба атрибута и @autoclosure, и @escaping.

// customersInLine равен ["Barry", "Daniella"]

var customerProviders: [() ->String] = []

func collectCustomerProviders(\_customerProvider: @autoclosure @escaping () ->String) {

customerProviders.append(customerProvider)

}

collectCustomerProviders(customersInLine.remove(at: 0))

collectCustomerProviders(customersInLine.remove(at: 0))

print("Collected \(customerProviders.count) closures.")

// Выведет "Collected 2 closures."

for customerProviderincustomerProviders {

print("Now serving \(customerProvider())!")

}

// Выведет "Now serving Barry!"

// Выведет "Now serving Daniella!"

Вкодевыше, вместотого, чтобывызыватьпереданноезамыканиевкачествеаргумента customer, функция collectCustomerProviders(\_:) добавляетзамыканиекмассиву customerProviders. Массив объявлен за пределами функции, что означает, что замыкание в массиве может быть исполнено после того, как функция вернет значение. В результате значение аргумента customerProvider должен иметь “разрешение” на “побег” из зоны видимости функции.

Для начала укрепимся с мыслью, что в Swift функции являются носителями гордого статуса объектов первого класса. Это значит, что функцию можно хранить в переменной, передавать как параметр, возвращать в качестве результата работы другой функции. Вводится понятие «типа функции». Этот тип описывает не только тип возвращаемого значения, но и типы входных аргументов.

Допустим, у нас есть две похожие функции, которые описывают две математические операции сложения и вычитания:

func add(op1: Double, op2: Double) ->Double {

return op1 + op2

}

func subtract(op1: Double, op2: Double) ->Double {

return op1 - op2

}

Их тип будет описываться следующим образом:

(Double, Double) ->Double

Прочесть это можно так: «Перед нами тип функции с двумя входными параметрами типа Double и возвращаемым значением типа Double.»

Мы можем создать переменную такого типа:

// Описываем переменную

varoperation: (Double, Double) ->Double

// Смело присваиваем этой переменной значение

// нужной нам функции, в зависимости от каких-либо условий:

foriin0..<2 {

ifi == 0 {

operation = add

} else {

operation = subtract

}

let result = operation(1.0, 2.0) // "Вызываем" переменную

println(result)

}

Код, описанный выше, выведет в консоли:

3.0

-1.0

Используем еще одну привилегию объекта первого класса. Возвращаясь к предыдущему примеру, мы могли бы создать такую новую функцию, которая бы принимала одну из наших старых функций типа (Double, Double) ->Double в качестве последнего параметра. Воттаконабудетвыглядеть:

// (1)

funcperformOperation(op1: Double, op2: Double, operation: (Double, Double) ->Double) ->Double { // (2)

returnoperation(op1, op2) // (3)

}

Разберем запутанный синтаксис на составляющие. Функция performOperation принимает три параметра:

• op1 типа Double (op — сокращенное от «операнд»)

• op2 типа Double

• operation типа (Double, Double) ->Double

В своем теле performOperation просто возвращает результат выполнения функции, хранимой в параметре operation, передавая в него первых два своих параметра.

Пока что выглядит запутанно, и, возможно, даже не понятно. Немного терпения, господа.

Давайте теперь передадим в качестве третьего аргумента не переменную, а анонимную функцию, заключив ее в фигурные {} скобки. Переданный таким образом параметр и будет называться замыканием:

letresult = performOperation(1.0, 2.0, {(op1:Double, op2: Double) ->Doublein

return op1 + op2 // (5)

}) // (4)

println(result) // Выводит 3.0 вконсоли

Отрывок кода (op1:Double, op2: Double) ->Doublein — это, так сказать, «заголовок» замыкания. Состоит он из:

• псевдонимов op1, op2 типа Double для использования внутри замыкания

• возвращаемого значения замыкания ->Double

• ключевого слова in

Еще раз о том, что сейчас произошло, по пунктам:

(1) Объявлена функция performOperation

(2) Эта функция принимает три параметра. Два первых — операнды. Последний — функция, которая будет выполнена над этими операндами.

(3) performOperation возвращает результат выполнения операции.

(4) В качестве последнего параметра в performOperation была передана функция, описанная замыканием.

(5) В теле замыкания указывается, какая операция будет выполняться над операндами.

Авторы Swift приложили немало усилий, чтобы пользователи языка могли писать как можно меньше кода и как можно больше тратить свое драгоценное время на размышления об архитектуре проекта. Взяв за основу наш пример с арифметическими операциями, посмотрим, до какого состояния мы сможем его «раскрутить».

Во-первых, можно не указывать типы входных параметров в замыкании явно, так как компилятор уже знает о них. Вызовфункциитеперьвыглядиттак:

performOperation(1.0, 2.0, {(op1, op2) ->Doublein

return op1 + op2

})

Во-вторых, если замыкание передается в качестве последнего параметра в функцию, то синтаксис позволяет сократить запись, и код замыкания просто прикрепляется к хвосту вызова:

performOperation(1.0, 2.0) {(op1, op2) ->Doublein

return op1 + op2

}

Приятная (в некоторых случаях) особенность языка заключается в том, что если код замыкания умещается в одну строку, то результат выполнения этой строки автоматичеси будет возвращен. Таким образом ключевое слово «return» можно не писать:

performOperation(1.0, 2.0) {(op1, op2) ->Doublein

op1 + op2

}

Идем дальше. Интересно, что Swift позволяет использовать так называемые стенографические (англ. shorthand) имена для входных параметров в замыкании. Т.е. каждому параметру по умолчанию присваивается псевдоним в формате $n, где n — порядковый номер параметра, начиная с нуля. Таким образом, нам, оказывается, даже не нужно придумывать имена для аргументов. В таком случае весь «заголовок» замыкания уже не несет в себе никакой смысловой нагрузки, и его можно опустить:

performOperation(1.0, 2.0) { $0 + $1 }

Согласитесь, эта запись уже совсем не похожа на ту, которая была в самом начале.

Давайте посмотрим на предыдущую запись и зададимся вопросом, что уже знает компилятор о замыкании? Он знает количество параметров (2) и их типы (Double и Double). Знает тип возвращаемого значения (Double). Так как в коде замыкания выполняется всего одна строка, он знает, что ему нужно возвращать в качестве результата его выполнения. Можно ли упростить эту запись как-то еще?

Оказывается, можно. Если замыкание работает только с двумя входными аргументами, в качестве замыкания разрешается передать операторную функцию, которая будет выполняться над этими аргументами (операндами). Теперь наш вызов будет выглядеть следующим образом:

performOperation(1.0, 2.0, +)

Теперь можно производить элементарные операции над нашими операндами в зависимости от некоторых условий, написав при этом минимум кода.

Кстати, Swift также позволяет использовать операции сравнения в качестве операторной фуниции. Выглядетьэтобудетпримернотак:

funcperformComparisonOperation(op1: Double, op2: Double, operation: (Double, Double) ->Bool) ->Bool {

returnoperation(op1, op2)

}

println(performComparisonOperation(1.0, 1.0, >=)) // Выведет "true"

println(performComparisonOperation(1.0, 1.0, <)) // Выведет "false"

Илибитовыеоперации:

funcperformBitwiseOperation(op1: Bool, op2: Bool, operation: (Bool, Bool) ->Bool) ->Bool {

returnoperation(op1, op2)

}

println(performBitwiseOperation(true, true, ^)) // Выведет "false"

println(performBitwiseOperation(true, false, |)) // Выведет "true"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Если вам нужно создать очередь с приоритетом, можно использовать двоичную кучу (binaryheap). Как известно, это может быть как MinHeap, так и MaxHeap, т.е. кучи, где в корне дерева находится минимальный или максимальный элемент соотвественно. Базовая реализация MinHeap от MaxHeap будет отличаться по сути только проверочными сравнениями при восстановлении инварианта двоичной кучи после добавления/удаления элемента.

Таким образом, мы могли создать базовый класс BinaryHeap, который будет содержать свойство comparison типа (T, T) ->Bool. А конструктор этого класса будет принимать способ сравнения и затем использовать его в методах heapify. Прототип базового класса выглядел бы так:

classBinaryHeap<T: Comparable>: DebugPrintable{

privatevar array: Array<T?>

privatevar comparison: (T, T) ->Bool

privatevar used: Int = 0

// Бла-бла-бла

// Internal Methods

internalfuncremoveTop() ->T? { //... }

internalfuncgetTop() ->T? { //... }

// Public Methods:

funcaddValue(value: T) {

if used == self.array.count {

self.grow()

}

self.array[used] = value

heapifyToTop(used, comparison) // Одно из мест, где используется функция сравнения

self.used++

}

init(size newSize: Int, comparison newComparison: (T, T) ->Bool) {

array = [T?](count: newSize, repeatedValue: nil)

comparison = newComparison

}

}

Теперь для того, чтобы создать классы MinHeap и MaxHeap нам достаточно унаследоваться от BinaryHeap, а в их конструкторах просто явно указать, какое сравнение применять. Вот так будет выглядеть наши классы:

classMaxHeap<T: Comparable>: BinaryHeap<T>{

funcgetMax() ->T? {

returnself.getTop()

}

funcremoveMax() ->T? {

returnself.removeTop()

}

init(sizenewSize:Int) {

super.init(size: newSize, {$0> $1})

}

}

classMinHeap<T: Comparable>: BinaryHeap<T>{

funcgetMin() ->T? {

returnself.getTop()

}

funcremoveMin() ->T?{

returnself.removeTop()

}

init(size newSize: Int) {

super.init(size: newSize, {$0<= $1})

}

}

Задание для самостоятельного выполнения

Выполнить с учетом варианта

1. Создать массив из 5-ти элементов «бонусные акции», содержащий названия акций точки продаж(магазина, кафе и т.п.), отсортировать его, используя замыкания

2. Создать функцию с замыканием «увеличить бонус на», для каждой бонусной акции задать свое значение инкремента.

3. Создать класс «бонусные акции» с полями название и бонус, описать функции замыкания для начисления бонусов клиенту.

4. Изменить свой код так, чтобы при выборе любой из 5-ти акций изменялся накопленный бонус клиента в соответствие с выбранной акцией, акции и бонусы хранятся и обрабатываются с использованием класса и его методов, примените замыкание для захвата значения бонуса клиента.

**Ход работы:**

Создание массива бонусных акций и сортировка его с использованием замыканий:

var bonusActions = ["Скидка на кофе", "Скидка на пиццу", "Бесплатный чай", "Скидка на десерт", "Подарок за покупку"]

bonusActions.sort(by: { $0 < $1 })

Создание функции с замыканием "увеличить бонус на":

func increaseBonus(by amount: Int, completion: (Int) -> Void) {

let currentBonus = 100 // текущий бонус клиента

let newBonus = currentBonus + amount

completion(newBonus)

}

Создание класса "Бонусные акции" с полями "название" и "бонус" и описание функции замыкания для начисления бонусов клиенту:

class BonusAction {

var name: String

var bonus: Int

init(name: String, bonus: Int) {

self.name = name

self.bonus = bonus

}

func applyBonus(to client: inout Int, completion: () -> Void) {

client += bonus

completion()

}

}

Изменение кода для обработки выбора акции и изменения накопленного бонуса клиента с использованием класса и его методов, а также применение замыкания для захвата значения бонуса клиента:

var clientBonus = 1000 // начальный бонус клиента

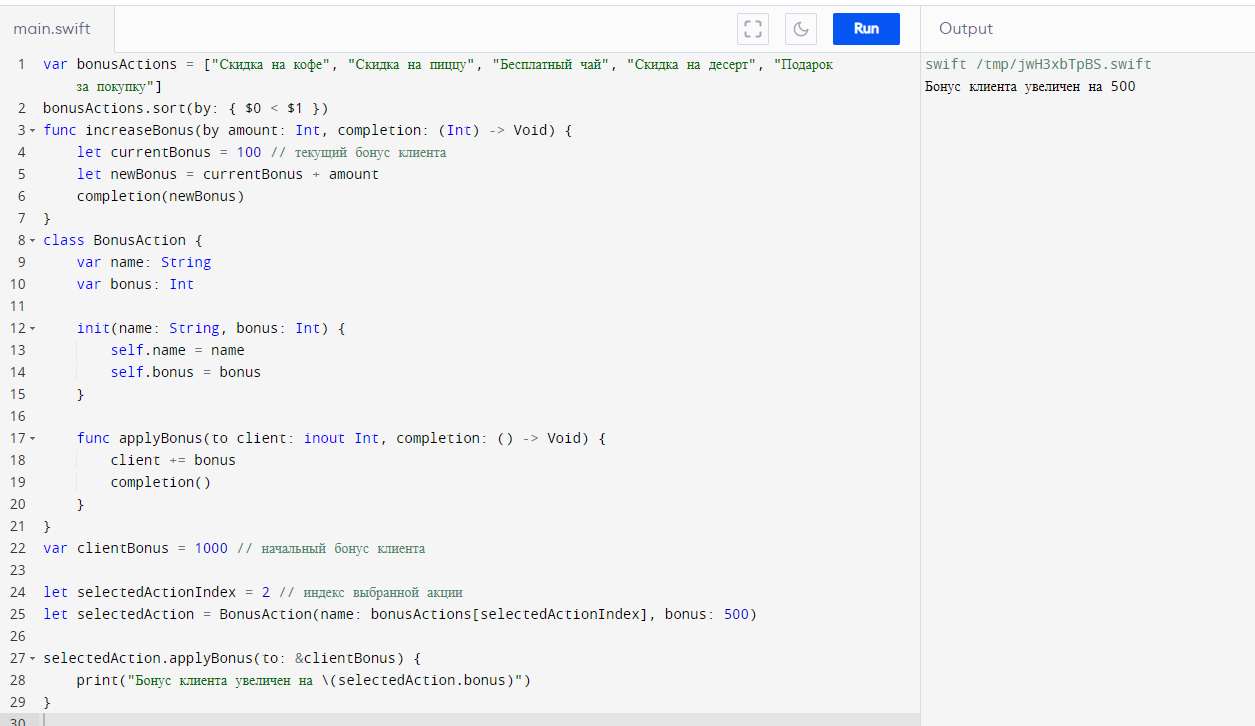
let selectedActionIndex = 2 // индекс выбранной акции

let selectedAction = BonusAction(name: bonusActions[selectedActionIndex], bonus: 500)

selectedAction.applyBonus(to: &clientBonus) {

print("Бонус клиента увеличен на \(selectedAction.bonus)")

}



**Контрольные вопросы:**

1. Что такое замыкание?
2. Какие два вида замыканий существуют в Swift?
3. Как передать замыкание в функцию в Swift?
4. Что такое захват значений в замыкании?
5. Какие возможности предоставляют замыкания для работы с коллекциями?
6. Как использовать замыкания для сортировки массива?
7. Как использовать замыкания для обработки асинхронных операций?
8. Как можно создать функцию с замыканием в Swift?
9. Как возвращать значение из замыкания в Swift?
10. Какие преимущества предоставляет использование замыканий в разработке программного обеспечения?

Варианты

1. Магазин «Магнит»

2. Магазин «Перекресток»

3. Магазин «Marmalato»

4. Магазин «1000 и 1 бутылка»

5. Авиалинии, бонусы за километры

6. Служба «Такси»

7. Служба ремонта «Почини», скидки при последующих заказах

8. Кафе «У друзей»

9. Турфирма «Пегас»

10. Кинотеатр «Рассвет»

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическая работа № 11** Опциональные типы

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы:** изучить опциональные типы, научиться применять их на практике.

Краткие теоретические сведения

Что такое Optionals?

Optionals (опционалы) — это удобный механизм обработки ситуаций, когда значение переменной может отсутствовать. Значение будет использовано, только если оно есть.

Зачем нужныOptionals, когда есть проверка на nil?

Во-первых, проверка на равенство/неравенство nil применима только к nullable-типам и не применима к примитивным типам, структурам и перечислениям. Для обозначения отсутсвия значения у переменной примитивного типа приходится вводить спецзначения, такие как NSNotFound.

Опциональные типы представляют объекты, которые могут иметь, а могут и не иметь значение. Опциональные типы выступают двойниками базовых типов. Все они имеют в конце вопросительный знак: Int?, String? и т.д. Вопросительный знак как раз указывает, что это опциональный тип.

Например, рассмотрим следующую ситуацию:

letsomeString = "123"

letsomeNumber = Int(someString)

Здесь инициализатор Int(someString) преобразует строку someString в число. В данном случае у нас все нормально, так как строка "123" действительно содержит число 123. Однако, что, если бы переменная someString представляла бы строку "hello"? В этом случае инициализатор не смог бы преобразовать строку в число. Поэтому инициализатор возвращает не просто объект Int, а Int?, то есть объект, который может иметь, а может не иметь значения.

По факту, если объект не имеет значения, то ему присваивается специальное значение nil. В коде мы также можем установить явным образом это значение:

Var number: Int? = 12

number = nil // теперь переменная number не имеет значения

Значение nil может применяться только к объектам опциональных типов.

Фактически запись типа Int? является сокращением от Optional<Int>. То есть мы также можем определить переменную следующим образом:

varnumber: Optional<Int> = 12

Несмотря на то, что в примере выше переменной number присваивается число 12, но фактически переменная будет иметь в качестве значения Optional(12), то есть мы могли бы написать следующим образом:

varnumber : Optional<Int>= Optional(12)

// илитак

varnumber2 = Optional(12)

При этом опять же стоит понимать, что Optional<Int>, это не то же самое, что и Optional<String> или Optional<Doublegt;, например:

varnumber = Optional(12)

number = Optional("12") // Ошибка number представляеттип Optional<Int>, ане Optional<String>

Получение значения из Optional

При работе с объектами опциональных типов следует помнить, что они не эквивалентны объектам обычных типов. То есть следующий пример у нас работать не будет:

vara: Int? = 12

varb: Int= 10

varc = a + b // ошибка - разные типы

a и b здесь переменные разных типов, хотя казалось бы обе переменных хранят целые числа. И чтобы полноценно работать с объектами опциональных типов, следует извлечь из них значение. Для извлечения значения используется оператор ! - восклицательный знак после названия объекта опционального типа. Данный оператор еще называют unwrapoperator или forcedunwrapoperator:

vara: Int? = 12

varb: Int= 10

varc = a! + b // с = 22

Другой пример:

varb: Int= 10

vara: Int? = Int("123")

b = a! + b

print(a!) // 123

print(b) // 133

Неявное получение значений Optional

Swift предоставляет еще один способ получения значения подобных типов, который заключается в использовании типов Optional с неявно получаемым значением (implicitlyunwrappedOptional):

varb: Int= 10

vara: Int! = Int("123")

b = a + b

print(a) // 123

print(b) // 133

Здесь переменная a имеет тип Int!, а не Int?. Фактиччески это тот же самый Optional, но теперь нам явным образом не надо применять оператор ! для получения его значения.

Проверка Optional на nil

В то же время если переменная a в примере выше не будет содержать конкретное значение, то программа опять же выбросит ошибку. Например? вслучаеvar a: Int! = Int("abc")илиvar a: Int? = Int("abc"). Поэтому перед использованием объектов опциональных типов желательно проверить, что они имеют какие-либо значение.

Для проверки мы можем использовать условную конструкцию if. Ее общая форма:

If var переменная | let константа = опциональное\_значение {

действия1

} else{

действия2

}

Если опциональное\_значение не равно nil, то оно присваивается создаваемой переменной (или константе), и выполняются действия1. Иначе выполняются действия2.

Например:

Var str: String= "123"

Var b: Int= 10

If var a = Int(str){

a+=b

print(a)

}

else{

print(b)

}

Если выражение Int(str) (которое возвращает объект Int?) успешно преобразует строку в число, то есть будет иметь значение, то создается переменная a, которой присваивается полученное значение, и затем выполняется код:

a+=b

print(a)

Если же преобразование из строки в число завершится с ошибкой, и выражение Int(str) возвратит значение nil, то выполняется код в блоке else:

else{

print(b)

}

Но также в данном случае мы могли и по другому проверить на значение nil:

varstr: String= "123"

varb: Int= 10

vara: Int? = Int(str)

if a != nil{

a+=b

print(a)

}

else{

print(b)

}

Если надо проверить значения нескольких переменных или констант, то все их можно указать в одном выражении if:

leta = Int("123")

letb = Int("456")

ifletaVal = a, letbVal = b{

print(aVal)

print(bVal)

}

else{

print("Error")

}

В данном случае выражение if выполняется, если и a, и b не равныnil. Иначе выполняется блок else.

Сравнение объектов Optional

При сравнении объекта Optional с объектом конкретного типа, Swift преобразует объект конкретного типа к типу Optional:

leta: Int? = 10

ifa == 10{

print("a is equal to 10")

}

else{

print("a is not equal to 10")

}

И таким образом работают операции == и !=. Однако с операциями <, >, <=, >= все будет несколько иначе. Например, следующий код выдаст ошибку:

leta: Int? = 10

ifa > 5{

print("a is greater than 5")

}

И в подобных операциях к объекту Optional необходимо применить оператор !:

leta: Int? = 10

ifa != nil&& a! > 5{

print("a is greater than 5")

}

Optional в switch..case

Если сравниваемое значение в конструкции switch представляет объект Optional, то с помощью операции ? мы можем получить и сравнивать его значение при его наличии:

leti = Int("1")

switch i {

case1?:

print("i is equal to 1")

caseletn?:

print("i is equal to \(n)")

casenil:

print("i is undefined")

}

Оператор nil-объединения

Оператор ?? позволяет проверить значения объекта Optional на nil. Этот оператор принимает два операнда a ?? 10. Если первый операнд не равен nil, то возвращается значение первого операнда. Если первый операнд равен nil, то возвращается второй операнд:

1

2

3 leta = Int("234")

letb = a ?? 10

print(b) // 234

В данном случае поскольку константа a не равна nil, то выражение a ?? 10 возвращает значение этой константы, то есть число 234.

Задание для самостоятельного выполнения

1. Создать пять строковых констант

Одни константы это только цифры, другие содержат еще и буквы

Найти сумму всех этих констант приведя их к Int

(Используйте и optionalbinding и forcedunwrapping)

2. С сервера к нам приходит тюпл с тремя параметрами:

statusCode, message, errorMessage (число, строка и строка) в этом тюплеstatusCode всегда содержит данные, но сама строка приходит только в одном поле если statusCode от 200 до 300 исключительно, то выводите message, в противном случает выводите errorMessage.

После этого проделайте тоже самое только без участия statusCode

3. Создайте 5 тюплов с тремя параметрами:

Имя, номер машины, оценка за контрольную при создании этих тюплов не должно быть никаких данных после создания каждому студенту установите имя, некоторым установите номер машины, некоторым установите результат контрольной, выведите в консоль:

- имена студента;

- есть ли у него машина;

- если да, то какой номер;

- был ли на контрольной;

- если да, то какая оценка.

**Ход работы:**

Создание пяти строковых констант:

let str1 = "123"

let str2 = "456"

let str3 = "abc"

let str4 = "789"

let str5 = "def"

Нахождение суммы всех констант, приведя их к типу Int с использованием optional binding и forced unwrapping:

swift

Копировать код

var sum = 0

if let num1 = Int(str1) {

sum += num1

}

if let num2 = Int(str2) {

sum += num2

}

if let num3 = Int(str3) {

sum += num3

}

if let num4 = Int(str4) {

sum += num4

}

if let num5 = Int(str5) {

sum += num5

}

print("Сумма всех чисел: \(sum)")

Обработка тюпла с тремя параметрами:

let response: (statusCode: Int, message: String?, errorMessage: String?) = (250, "Success", "Error")

if 200..<300 ~= response.statusCode {

if let message = response.message {

print("Message: \(message)")

}

} else {

if let errorMessage = response.errorMessage {

print("Error message: \(errorMessage)")

}

}

Создание и вывод информации о студентах с использованием тюплов:

let student1: (name: String, carNumber: Int?, examResult: Int?)

let student2: (name: String, carNumber: Int?, examResult: Int?)

let student3: (name: String, carNumber: Int?, examResult: Int?)

let student4: (name: String, carNumber: Int?, examResult: Int?)

let student5: (name: String, carNumber: Int?, examResult: Int?)

student1.name = "John"

student1.carNumber = 1234

student1.examResult = nil

student2.name = "Alice"

student2.carNumber = nil

student2.examResult = 85

student3.name = "Bob"

student3.carNumber = 5678

student3.examResult = 90

student4.name = "Emily"

student4.carNumber = nil

student4.examResult = nil

student5.name = "Mike"

student5.carNumber = 4321

student5.examResult = 75

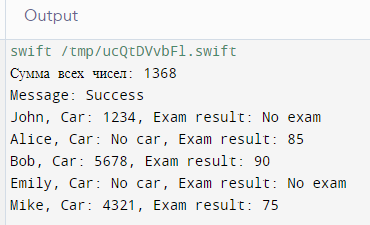
print("\(student1.name), Car: \(student1.carNumber != nil ? "\(student1.carNumber!)" : "No car"), Exam result: \(student1.examResult != nil ? "\(student1.examResult!)" : "No exam")")

print("\(student2.name), Car: \(student2.carNumber != nil ? "\(student2.carNumber!)" : "No car"), Exam result: \(student2.examResult != nil ? "\(student2.examResult!)" : "No exam")")

print("\(student3.name), Car: \(student3.carNumber != nil ? "\(student3.carNumber!)" : "No car"), Exam result: \(student3.examResult != nil ? "\(student3.examResult!)" : "No exam")")

print("\(student4.name), Car: \(student4.carNumber != nil ? "\(student4.carNumber!)" : "No car"), Exam result: \(student4.examResult != nil ? "\(student4.examResult!)" : "No exam")")

print("\(student5.name), Car: \(student5.carNumber != nil ? "\(student5.carNumber!)" : "No car"), Exam result: \(student5.examResult != nil ? "\(student5.examResult!)" : "No exam")")

****

**Контрольные вопросы:**

1. Что такое опциональные типы в Swift?
2. Какие два вида опциональных типов существуют в Swift?
3. Как можно извлечь значение из опционального типа?
4. Чем отличается optional binding от forced unwrapping?
5. Как проверить, содержит ли опциональное значение nil?
6. Какие проблемы могут возникнуть при использовании forced unwrapping?
7. Как обрабатывать опциональные значения при работе с тюплами?
8. Какие методы обработки опциональных значений предоставляет Swift?
9. Какие ситуации могут потребовать использования опциональных типов?

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

***Тема 1.3.2 Создание и тестирование модулей для мобильных приложений***

**Практическая работа № 12** Сабскрипты и Опциональная последовательность

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках

**Цель работы:** научиться создавать и использовать опциональные последовательности.

Краткие теоретические сведения

Классы, структуры и перечисления могут определять сабскрипты (subscripts). Сабскрипты используются для доступа к элементам коллекции или последовательности. В ряде языков программирования есть похожая концепция - индексаторы. Сабскрипты позволяют обращаться с объектом класса или структуры как с отдельной коллекцией.

Для определения сабскрипта используется ключевое слово subscript:

subscript(параметры) ->тип\_возвращаемых\_объектов {

get{

// возвращаем значение

}

set(newValue) {

// устанавливаем новое значение newValue

}

}

После ключевого слова subscript в скобках идут параметры, которые используются для получения элементов. Нередко параметры представляют числовой индекс, по которому надо получить объект. Далее указывается тип элементов, с которыми мы работаем.

Сабскрипт может состоять из двух блоков: get и set. Блок get возвращает элемент, а блок set устанавливает новое значение, которое передается через параметр newValue.

Например, создадим класс библиотеки. Упрощенно библиотека представляет некоторый набор книг. То есть мы можем представить класс библиотеки как коллекцию книг и использовать сабскрипты для получения книг по индексу:

classBook{ // класскниги

varname: String

init(name: String){

self.name = name

}

}

classLibrary{ // классбиблиотеки

varbooks: [Book] = [Book]()

init(){

books.append(Book(name: "Войнаимир"))

books.append(Book(name: "Отцыидети"))

books.append(Book(name: "Чайка"))

}

subscript(index: Int) -> Book{

get{

returnbooks[index]

}

set(newValue){

books[index] = newValue

}

}

}

varmyLibrary: Library = Library()

varfirstBook: Book = myLibrary[0] // получаемэлементпоиндексу 0

print(firstBook.name) // Войнаимир

myLibrary[2] = Book(name:"Мартин Иден") // установка элемента по индексу 2

print(myLibrary[2].name) // МартинИден

Здесь сабскриптпредназначен для работы с типом Book. В блоке get происходит получение объекта Book по индексу из массива books. В блоке set устанавливаем объект Book в массиве books.

В итоге в программе мы сможем обращаться к библиотеке как к массиву по индексу для получения нужной книги:

1 varfirstBook: Book = myLibrary[0]

Есть два типа сабскриптов:

• Сабскрипты, которые поддерживают чтение и запись (то есть с блоками get и set, как в примере выше)

• Сабскрипты только для чтения (только с блоком get)

Изменим класс библиотеки, чтобы применить сабскрипт только для чтения:

Class Library{

Var books: [Book] = [Book]()

init(){

books.append(Book(name: "Войнаимир"))

books.append(Book(name: "Отцыидети"))

books.append(Book(name: "Чайка"))

}

subscript(index: Int) -> Book{

returnbooks[index]

}

}

Опциональная последовательность

Опциональная цепочка (optionalchaining) - процесс запросов и вызовов свойств, методов, сабскриптов (индексов) у опционала, который может быть nil. Если опционал содержит какое-либо значение, то вызов свойства, метода или сабскрипта успешен, и наоборот, если опционал равен nil, то вызов свойства, метода или сабскрипта возвращает nil. Множественные запросы могут быть соединены вместе, и вся цепочка этих запросов не срабатывает, если хотя бы один запрос равен nil.

Опциональная последовательность в Swift - аналог сообщению nil в Objective-C, но только она работает со всеми типами, и может быть проверена на успех или неудачу.

ОП как альтернатива принудительному извлечению

Вы обозначаете опциональную последовательность, когда ставите вопросительный знак (?) опционального значения, свойство, метод или индекс которого вы хотите вызвать, если опционал не nil. Это очень похоже на установку восклицательного знака (!) после опционального значения для принудительного извлечения его значения. Основное отличие в том, что опциональная последовательность не исполняется, если опционал равен nil, в то время как принудительное извлечение приводит к runtime ошибке, когда опционал равен nil.

В частности, результат вызова опциональной последовательности того же типа, что и тип ожидаемого возвращаемого значения, только в завернутом в опционал виде. Свойство, которое обычно возвращало Int, вернет Int?, когда обращаются к нему через опциональную последовательность.

Первые два класса Person, Residence определены как:

classPerson {

var residence: Residence?

}

classResidence {

varnumberOfRooms = 1

}

Экземпляры Residence имеют единственное свойство numberOfRooms типа Int, со значением по умолчанию 1. Экземпляры Person имеют опциональное свойство residence типа Residence?.

Если вы создаете новый экземпляр Person, то его свойство residence по умолчанию имеет значение nil, в силу того, что оно является опционалом. В коде ниже john имеет свойство residence, значение которого nil:

letjohn = Person()

Если вы попытаетесь получить доступ к свойству numberOfRooms свойства residence экземпляра Person, поставив восклицательный знак после residence, для принудительного извлечения, то вы получите ошибку исполнения, потому что residence не имеет значения для извлечения:

letroomCount = john.residence!.numberOfRooms

// ошибка runtime

Код, представленный выше, срабатывает успешно, если john.residence имеет не nil значение и устанавливает корректное значение типа Int для roomCount. Однако этот код всегда будет выдавать ошибку исполнения, когда residence равен nil, что указано выше.

Опциональная последовательность предоставляет альтернативный способ получить доступ к значению numberOfRooms. Для использования опциональной последовательности используйте вопросительный знак, на месте восклицательного знака:

ifletroomCount = john.residence?.numberOfRooms {

print("John's residence has \(roomCount) room(s).")

} else {

print("Unable to retrieve the number of rooms.")

}

// Выведет "Unable to retrieve the number of rooms."

Определение классовых моделей для ОП

Можно использовать опциональную последовательность для вызовов свойств, методов, сабскриптов, которые находятся более чем на один уровень глубже. Это позволяет вам пробираться через подсвойства, внутри сложных моделей вложенных типов, и проверять возможность доступа свойств, методов и сабскриптов этих подсвойств.

Фрагмент кода ниже определяет четыре модели классов для использования в нескольких следующих примерах, включая примеры с многоуровневой опциональной последовательностью. Эти классы расширяют модели Person, Residence приведенные ранее, добавляя классы Room, Address со свойствами, методами и сабскриптами.

Класс Person объявляется так же как и раньше:

classPerson{

var residence: Residence?

}

Класс Residence стал намного сложнее, чем был раньше. В этот раз класс Residence определяет переменное свойство rooms, которое инициализировано пустым массивом [Room]:

classResidence{

var rooms = [Room]()

varnumberOfRooms: Int {

returnrooms.count

}

subscript(i: Int) ->Room {

get {

return rooms[i]

}

set {

rooms[i] = newValue

}

}

funcprintNumberOfRooms() {

print("Общееколичествокомнатравно\(numberOfRooms)")

}

varaddress: Address?

}

Из-за того, что эта версия Residence хранит массив экземпляров Room, его свойство numberOfRooms реализовано как вычисляемое, а не как хранимое свойство. Вычисляемое свойство numberOfRooms просто возвращает значение свойства.

В качестве сокращенного варианта доступа к массиву rooms, эта версия класса Residence предлагает сабскрипт (доступный как для чтения, так и для записи), который предоставляет доступ к комнате под требуемым индексом в массиве rooms.

Эта версия Residence так же обеспечивает метод printNumberOfRooms, который просто выводит на экран количество комнат в резиденции.

И наконец, Residence определяет опциональное свойство address типа Address?. Тип класса Address для этого свойства определен ниже.

Класс Room используется для массива rooms, в качестве простого класса с одним свойством name и инициализатором, в котором устанавливается значение свойства name как подходящее имя комнаты:

classRoom{

let name: String

init(name: String) {

self.name = name

}

}

Последний класс в этой модели Address. Этот класс имеет три опциональных свойства типа String?. Первые два свойства buildingName, buildingNumber являются альтернативным вариантом определения конкретного здания как части адреса. Третье свойство street используется для названия улицы, для этого адреса:

classAddress{

varbuildingName: String?

varbuildingNumber: String?

var street: String?

funcbuildingIdentifier() ->String? {

ifletbuildingNumber = buildingNumber, let street = street {

return"\(buildingNumber)\(street)"

} elseifbuildingName != nil {

returnbuildingName

} else {

returnnil

}

}

}

Класс Address так же предоставляет метод buildingIdentifier(), который возвращает String?. Этот метод проверяет значения свойств экземпляра и возвращает buildingName, если у него есть значение или возвращает buildingNumber, если у него есть значение или nil, если ни у одного из свойств нет значения.

Доступ к свойствам через ОП

Как было показано в разделе ОП как альтернатива принудительному извлечению, вы можете использовать опциональную последовательность для доступа к свойству опционального значения и проверить результат доступа к этому свойству на успешность

Используйте классы, определенные ранее, для создания нового экземпляра Person и попробуйте получить доступ к свойству numberOfRooms, как вы делали ранее:

let john = Person()

ifletroomCount = john.residence?.numberOfRooms {

print("John's residence has \(roomCount) room(s).")

} else {

print("Unable to retrieve the number of rooms.")

}

// Выведет "Unable to retrieve the number of rooms."

Так как john.residence равен nil, этот вызов опциональной последовательности не будет успешен как и ранее.

Вы можете попробовать установить значение свойства через опциональную последовательность:

letsomeAddress = Address()

someAddress.buildingNumber = "29"

someAddress.street = "Acacia Road"

john.residence?.address = someAddress

В этом примере попытка установить значение свойству address опциональному свойству residence? провалится, так как john.residence все еще nil.

Присваивание является частью опциональной цепочки, что означает, что никакой код с правой стороны не вычисляется. В предыдущем примере не так просто заметить, что someAddress никогда не вычисляется, потому что доступ к самой константе не имеет никаких побочных эффектов. Пример ниже делает тоже самое присваивание, но он использует функцию для того, чтобы создать адрес. Функция выводит "Functionwascalled" до того, как вернется значение, что позволяет вам увидеть была ли вычислена правая часть от оператора присваивания.

funccreateAddress() ->Address {

print("Function was called.")

letsomeAddress = Address()

someAddress.buildingNumber = "29"

someAddress.street = "Acacia Road"

returnsomeAddress

}

john.residence?.address = createAddress()

Можно заметить, что функция createAddress() не вызывается, так как ничего в консоли у нас не выводится.

Вызов методов через ОП

Вы можете использовать опциональную последовательность для вызова метода опциональной величины, и проверить сам вызов метода на успешность. Вы можете сделать это, даже если этот метод не возвращает значения.

Метод printNumberOfRooms класса Residence выводит текущее значение numberOfRooms. Вот как выглядит этот метод:

funcprintNumberOfRooms() {

print("Общее количество комнат равно \(numberOfRooms)")

}

Этот метод не определяет возвращаемого значения. Однако функции и методы без возвращаемого значения имеют неявный возвращаемый тип Void. Если вы вызовите этот метод на опциональном значении в опциональной последовательности, то он вернет тип не Void, а Void?, потому что возвращаемые значения всегда опционального типа, когда они вызываются через опциональную последовательность.if john.residence?.printNumberOfRooms() != nil { print("Есть возможность вывести общее количество комнат.") } else { print("Нет возможности вывести общее количество комнат.") } // Выведет "Нет возможности вывести общее количество комнат."

Доступ к сабскриптамчерез ОП

Когда вы получаете доступ к опциональному значению через опциональную последовательность, вы размещаете вопросительный знак до скобок сабскрипта (индекса), а не после. Вопросительный знак опциональной последовательности следует сразу после части выражения, которая является опционалом.

Из-за того, что john.residence является nil, то вызов сабскрипта проваливается:

ifletfirstRoomName = john.residence?[0].name {

print("Название первой комнаты \(firstRoomName).")

} else {

print("Никак не получить название первой комнаты.")

}

// Выведет "Никак не получить название первой комнаты."

Вызов вопросительного знака опциональной последовательности в этом сабскрипте идет сразу после john.residence, но до скобок сабскрипта, потому что john.residence является опциональным значением, на которое применяется опциональная последовательность.

Аналогично вы можете попробовать установить новое значение через сабскрипт с помощью опциональной последовательности:

john.residence?[0] = Room(name: "Bathroom")

Это попытка установки значения через сабскрипт так же не срабатывает, так как residence все еще nil.

Если вы создадите и присвоите действительное значение экземпляру Residence, при помощи одного или нескольких экземпляров Room в массиве rooms, то вы сможете использовать сабскрипт на экземпляре residence для того, чтобы получить доступ к массиву rooms через опциональную последовательность:

letjohnsHouse = Residence()

johnsHouse.rooms.append(Room(name: "Гостиная"))

johnsHouse.rooms.append(Room(name: "Кухня"))

john.residence = johnsHouse

ifletfirstRoomName = john.residence?[0].name {

print("Название первой комнаты \(firstRoomName).")

} else {

print("Никак не получить название первой комнаты.")

}

// Выведет "Название первой комнаты Гостиная."

Получение доступа к сабскрипту (индексу) опционального типа

Если сабскрипт возвращает значение опционального типа, например ключ словаря типа Dictionary в Swift, то мы должны поставить вопросительный знак после закрывающей скобки сабскрипта, для присоединения его опционального возвращаемого значения:

vartestScores = ["Dave": [86, 82, 84], "Bev": [79, 94, 81]]

testScores["Dave"]?[0] = 91

testScores["Bev"]?[0] += 1

testScores["Brian"]?[0] = 72

// массив "Dave" теперьимеетвид [91, 82, 84], массив "Bev" - [80, 94, 81]

Пример выше определяет словарь testScores, который содержит две пары ключ/значение, которые соединяют ключ типа String со значением типа [Int]. Пример использует опциональную последовательность для установки значения первого элемента ключа "Dave" равным 91, для увеличения первого элемента массива под ключом "Bev" на 1 и для попытки установить первое значение несуществующего массива, соответствующего ключу "Brian" равным 72. Первые два вызова завершились успешно, потому что их ключи находятся в testScores. Третий вызов завершился неудачей, так как такого ключа как "Brian" в словаре не оказалось.

Соединение нескольких уровней ОП

Вы можете соединить несколько уровней опциональных последовательностей вместе для того, чтобы пробраться до свойств, методов, сабскриптов, которые находятся глубже в модели. Однако многоуровневые опциональные последовательности не добавляют новых уровней опциональности к возвращаемым значениям:

Скажем другими словами:

• Если тип, который вы пытаетесь получить не опциональный, то он станет опциональным из-за опциональной последовательности.

• Если тип, который вы пытаетесь получить, уже опциональный, то более опциональным он уже не станет, даже по причине опциональной последовательности.

Таким образом:

• Если вы пытаетесь получить значение типа Int через опциональную последовательность, то получите Int?, и это не будет зависеть от того, сколько уровней в опциональной последовательности задействовано.

• Аналогично, если вы попытаетесь получить значение типа Int? через опциональную последовательность, то вы получите Int?, что опять таки не зависит от количества уровней, которые задействованы в опциональной последовательности.

Пример ниже пробует получить доступ к свойству street свойства address свойства residence экземпляра john. Здесь задействовано два уровня опциональной последовательности для того, чтобы соединить свойства residence и address, оба из которых опционального типа:

ifletjohnsStreet = john.residence?.address?.street {

print("John's street name is \(johnsStreet).")

} else {

print("Unable to retrieve the address.")

}

// Выведет "Unable to retrieve the address."

Значение john.residence на данный момент содержит корректный экземпляр класса Residence. Однако значение john.residence.address равно nil. Из-за этого вызов john.residence?.address?.street проваливается.

Обратите внимание, что в примере выше вы пытаетесь получить значение свойства street. Тип этого свойства String?. Возвращаемое значение john.residence?.address?.street так же String?, даже если два уровня опциональной последовательности применены в дополнение к опциональному типу самого свойства.

Если вы установите фактический экземпляр класса Address как значение для john.residence.address и установите фактическое значение для свойства street, то вы можете получить доступ к значению свойства street через многоуровневую опциональную последовательность (цепочку):

letjohnsAddress = Address()

johnsAddress.buildingName = "The Larches"

johnsAddress.street = "Laurel Street"

john.residence?.address = johnsAddress

ifletjohnsStreet = john.residence?.address?.street {

print("John's street name is \(johnsStreet).")

} else {

print("Unable to retrieve the address.")

}

// Выведет "John's street name is Laurel Street."

В этом примере попытка установить свойство address свойства john.residence будет успешной, потому что значение john.residence в настоящее время содержит действующий экземпляр Address.

Дженерики в Swift

Дженерики (универсальные шаблоны) позволяют вам писать гибкие, многократно используемые функции и типы, которые могут работать с любым типом. Вы можете написать код, который избегает дублирования и выражает свое предназначение в ясной, абстрактной манере.

Типы Array и Dictionary в Swift являются универсальными коллекциями (дженериками).

В приведенном ниже коде универсальная функция для свапа двух значений используется для строки и целого числа. Это пример кода многократного использования.

funcswapTwoValues<T>(\_ a: inout T, \_ b: inout T) {

lettemporaryA = a

a = b

b = temporaryA

}

var num1 = 4

var num2 = 5

var str1 = “a”

var str2 = “b”

swapTwoValues(&num1,&num2)

swapTwoValues(&str1,&str2)

print (“num1:”, num1) //output: 5

print (“num2:”, num2) //output: 4

print (“str1:”, str1) //output: b

print (“str2:”, str2) //output: a

Когда их следует использовать опциональные типы?

Опциональным (Optional, “опционал”) в Swift является тип, в котором значение может быть, а может и не быть. Опционалы обозначаются путем добавления «?» к любому типу.

Варианты использования опционала:

1. Фрагменты кода, которые могут претерпеть неудачу (я чего-то ожидал, но ничего не получил).

2. Объекты, которые в данный момент являются пустыми, но могут стать чем-то позже (и наоборот).

Хороший пример опционала:

Свойство, которое может присутствовать или отсутствовать, например отчество или муж/жена в классе Person.

Метод, который может возвращать либо значение, либо ничего, например, поиск соответствия в массиве.

Метод, который может вернуть либо результат, либо получить ошибку и ничего не вернуть, например, пытаться прочитать содержимое файла (в результате чего обычно будут возвращены данные файла), но файл не существует.

Свойства-делегаты, которые не всегда должны быть установлены и обычно устанавливаются после инициализации.

Как слабые ссылки в классах. То, на что они указывают, может быть установлено в nil в любое время.

Если вам нужен способ узнать, когда установлено значение (данные еще не загружены> данные) вместо использования отдельного логической переменной dataLoaded.

Что такое опциональная последовательность (optionalchaining) в Swift?

Процессы запроса, вызова свойств, сабскриптов и методов для опционала, который может иметь значение «nil», определяется как опциональная последовательность (опциональная цепочка).

Опциональная последовательность возвращает два значения —

• если опционал содержит “значение”, то при вызове связанных с ним свойств, методов и сабскриптов возвращается значение

• если опционал содержит “nil”, все связанные с ним свойства, методы и сабскрипты возвращают nil

Опциональная последовательность — это альтернатива принудительной распаковки.

Что такое принудительная распаковка (forcedunwrapping)?

Принудительная распаковка — это способ извлечения значения, содержащегося в опционале. Эта операция опасна, потому что вы, по сути, говорите компилятору: я уверен, что этот опционал содержит реальное значение, извлеки его!

let value: Int? = 1

letnewValue: Int = value! // ТеперьnewValueсодержит 1

letanotherOptionalInt: Int? = nil

letanotherInt = anotherOptionalInt! // Output:fatal error: внезапный nil прираспаковкеопциональногозначения.

Что такое неявная распаковка (implicitunwrapping)?

Неявная распаковка: когда мы определяем неявно распакованный опционал, мы определяем контейнер, который будет автоматически выполнять принудительную распаковку каждый раз, когда мы его считываем.

varname: String! = “Virat”

letstudent = name// В данный момент мы считываем текст

name = nil

letplayer = name//Output:fatalerror: внезапныйnil при распаковке опционального значения.

Если неявно распакованный опционал равен nil и вы пытаетесь получить доступ к его упакованному значению, вы вызовете ошибку времени выполнения. Результат точно такой же, как если бы вы поместили восклицательный знак после обычного опционала, который не содержит значения.

Что такое опциональныйбиндинг (Optionalbinding)?

Вы можете распаковывать опционалы как «безопасным», так и «небезопасным» способом. Безопасный способ — использовать опциональный биндинг.

Опциональныйбиндинг используется для выяснения, содержит ли опционал значение, и если да, то мы сделаем это значение доступным в качестве временной константы или переменной. Таким образом нет необходимости использовать суффикс! для доступа к его значению.

letpossibleString: String? = "Hello"

ifletactualString = possibleString {

//actualString - обычное (не опциональное) строковое значение

//равное значению, хранимому в possibleString

print(actualString)

}

else {

//possibleString не содержат значения, обработайте эту

//ситуацию

}

Задание для самостоятельного выполнения

1. Создать класс по варианту, сделать сабскрипты для чтения и изменения его полей.

2. Создать новый класс на основе исходного с учетом варианта задания (автопарк, библиотека, каталог, команда, группа, реестр фирм, склад компьютеров, спортклуб, автостоянка, каталог журнальных статей), использовать пример со стр 2.

3. Заполнить информацию и вывести на экран

4. Создать класс ошибок (код, сообщение), возвращаемых сервером. На его основе создайте учет ошибок за текущие сутки, добавьте время в качестве, ключа. Используя сабскрипты, верните ответ сервера в заданном временном интервале.

**Ход работы:**

Создание класса Library с использованием сабскриптов:

class Library {

var books: [String]

init(books: [String]) {

self.books = books

}

subscript(index: Int) -> String? {

get {

guard index >= 0 && index < books.count else {

return nil

}

return books[index]

}

set(newValue) {

guard let newValue = newValue, index >= 0 && index < books.count else {

return

}

books[index] = newValue

}

}

}

var library = Library(books: ["Book1", "Book2", "Book3"])

print(library[1] ?? "No book") // Output: Book2

library[1] = "New Book"

print(library[1] ?? "No book") // Output: New Book

Создание нового класса CarPark:

class CarPark {

var cars: [String]

init(cars: [String]) {

self.cars = cars

}

subscript(index: Int) -> String? {

get {

guard index >= 0 && index < cars.count else {

return nil

}

return cars[index]

}

set(newValue) {

guard let newValue = newValue, index >= 0 && index < cars.count else {

return

}

cars[index] = newValue

}

}

}

var carPark = CarPark(cars: ["Car1", "Car2", "Car3"])

print(carPark[2] ?? "No car") // Output: Car3

carPark[2] = "New Car"

print(carPark[2] ?? "No car") // Output: New Car

Заполнение информации и вывод на экран:

swift

Копировать код

print("Cars in the car park:")

for i in 0..<carPark.cars.count {

print("\(i + 1). \(carPark[i] ?? "")")

}

Создание класса ошибок и использование сабскриптов для учета ошибок за текущие сутки:

struct ServerError {

var code: Int

var message: String

}

class ErrorLog {

var errors: [Date: ServerError]

init(errors: [Date: ServerError]) {

self.errors = errors

}

subscript(date: Date) -> ServerError? {

get {

return errors[date]

}

set(newValue) {

errors[date] = newValue

}

}

}

var errorLog = ErrorLog(errors: [:])

let currentDate = Date()

errorLog[currentDate] = ServerError(code: 404, message: "Page not found")

errorLog[currentDate.addingTimeInterval(-3600)] = ServerError(code: 500, message: "Internal server error")

// Retrieving errors within a specific time interval

let startDate = Calendar.current.date(byAdding: .hour, value: -2, to: currentDate)!

let endDate = currentDate

for date in errorLog.errors.keys {

if date >= startDate && date <= endDate {

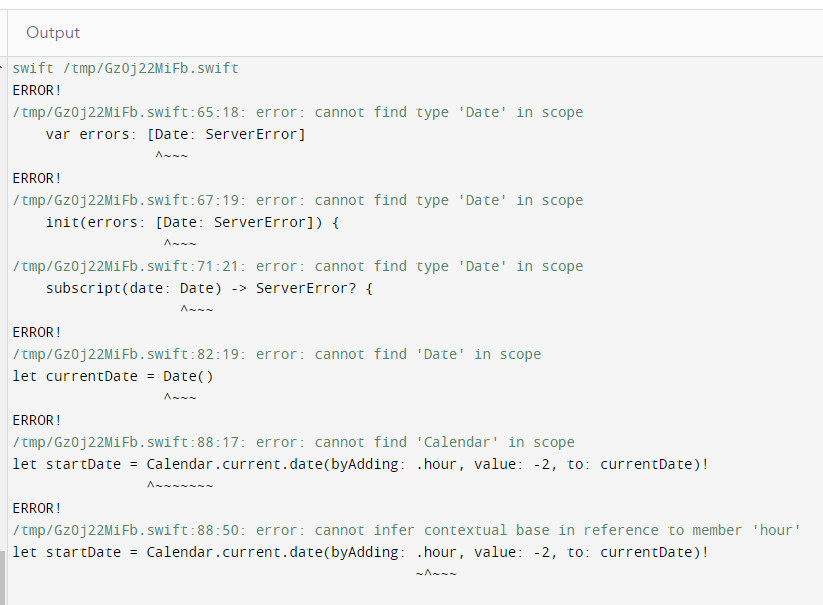
if let error = errorLog[date] {

print("\(date): \(error.code) - \(error.message)")

}

}

}



**Контрольные вопросы:**

1. Что такое сабскрипты в Swift?
2. Какова цель использования сабскриптов?
3. Как объявить сабскрипт в классе или структуре?
4. Как можно использовать сабскрипты для доступа к элементам массива или словаря?
5. Какие типы данных могут использоваться в качестве индекса для сабскриптов?
6. Какие возможности предоставляют сабскрипты для пользовательских типов данных?

Варианты

1. Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)

2. Книга (название книги, автор, издательство, количество страниц)

3. Файл (имя, время создания, дата создания, размер)

4. Спортсмен (ФИО, вид спорта, сколько лет занимается)

5. Студент (ФИО, группа, средний балл зачетки)

6. Фирма (название, вид услуг, годовой оборот)

7. Компьютер (производитель, частота процессора, количество ядер, объем памяти)

8. Спорт (вид спорта, количество занимающихся в городе, средняя стоимость комплекта инвентаря)

9. Автомобиль (марка, фирма-производитель, год выпуска, пробег)

10. Статья (название, автор, название журнала, год и месяц издания)

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическая работа № 13** Приведение типов, Any и AnyObject

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках

**Цель работы:** Изучить и научиться работать с типами Any и AnyObjectв Swift.

Краткие теоретические сведения

Приведение типов - это способ проверить тип экземпляра и/или способ обращения к экземпляру так, как если бы он был экземпляром суперкласса или подкласса откуда-либо из своей собственной классовой иерархии.

Приведение типов в Swift реализуется с помощью операторов is и as. Эти два оператора предоставляют простой и выразительный способ проверки типа значения или преобразование значения к другому типу.

Any и AnyObject

Any и AnyObject - это два специальных типа в Swift, которые используются для работы с неспецифическими типами.

Согласно документации Apple Swift,

1. Any может представлять экземпляр любого типа вообще, включая типы функций и дополнительные типы.

2. AnyObject может представлять экземпляр любого типа класса.

Поработаем с ними playgraund.

Any пример

Any позволяет работать со смесью разных типов, включая функциональные и неклассовые типы, такие как Int, String и Bool. Согласно документации, элементы в этом массиве являются структурами, которые являются типами значений, поэтому теоретически AnyObjec t не должен работать в этих случаях.

Включим Strings и Ints, которые являются типами значений в Swift, используя AnyObject .

Ошибка AnyObjectArray при включении типов структур

Как и ожидалось, компилятор выдает ошибку, говоря, что элементы не соответствуют типу AnyObject в массиве.

Элементы, приведенные в AnyObject

Можно использовать AnyObject для Ints и Strings, явно приведя каждый элемент к AnyObject.

Печать anyObjectArray

Элемент Hi явно выглядит как строка, но в нем нет кавычек, как у обычного значения String в Swift!

Распечатаем каждый элемент, используя цикл for-in, чтобы проверить его фактический тип, а не приведенный тип AnyObject .

Проверка типов элементов в anyObjectArray 1

Это тип String! Тогда как его передать в AnyObject ? Опять же, строки в Swift - это структуры, а не типы классов. Таким образом, теоретически я не могу использовать их как AnyObject .

Проверка типов элементов в anyObjectArray 2

Hi - это тоже NSString, а числовые элементы - это NSNumber! И… это ссылочные типы в Objective-C!

Печать массива NSString и массива строк

Привет, без кавычек в консоли как NSString

Элементы, приведенные к AnyObject в массиве, теперь являются типами классов Objective-C: NSString и NSNumber.

В рамках взаимодействия с Objective-C Swift предлагает удобные и эффективные способы работы с фреймворками Cocoa. Swift автоматически преобразует некоторые типы Objective-C в типы Swift, а некоторые типы Swift - в типы Objective-C.Типы, которые можно преобразовать между Objective-C и Swift, называются мостовые типы.

Везде, где вы можете использовать мостовой ссылочный тип Objective-C, вы можете использовать вместо него тип значения Swift. Это позволяет вам использовать функциональные возможности, доступные в реализации ссылочного типа, естественным образом для кода Swift. По этой причине вам почти никогда не нужно использовать мостовой ссылочный тип непосредственно в вашем собственном коде. Фактически, когда код Swift импортирует API-интерфейсы Objective-C, импортер заменяет ссылочные типы Objective-C их соответствующими типами значений. Аналогичным образом, когда код Objective-C импортирует API Swift, импортер также заменяет типы значений Swift их соответствующими ссылочными типами Objective-C ».

Другими словами, компилятор делает все возможное, чтобы быть гибким в обработке таких типов за счет автоматического преобразования и соединения, предотвращая при этом легкое падение нашего приложения.

Таккогда мы на самом деле используем AnyObject ? Как указано в документации Apple, AnyObject можно использовать для работы с объектами, которые являются производными от Class, но не имеют общего корневого класса.

Но обязательно ли использовать это в нашем коде?нет .

Apple говорит:

В Swift 3 тип id в Objective-C теперь сопоставляется с типом Any в Swift, который описывает значение любого типа, будь то класс, перечисление, структура или любой другой тип Swift. Это изменение делает API-интерфейсы Objective-C более гибкими в Swift, поскольку типы значений, определенные в Swift, могут передаваться в API-интерфейсы Objective-C и извлекаться как типы Swift, устраняя необходимость в ручных «блочных» типах.

Эти преимущества также распространяются на коллекции: типы коллекций Objective-C NSArray, NSDictionary и NSSet, которые ранее принимали только элементы AnyObject, теперь могут содержать элементы любого типа. Для хешированных контейнеров, таких как Dictionary и Set, есть новый тип AnyHashable, который может содержать значение любого типа, соответствующего протоколу Swift Hashable.

Кажется, что один Any отлично работает в соединении этих двух языков в Swift 3 без необходимости использования AnyObject !

Так что же было главной причиной этих изменений?

Своими словами Apple поясняет:

Swift 3 взаимодействует с API Objective-C более мощным способом, чем предыдущие версии. Например, Swift 2 сопоставил тип id в Objective-C с типом AnyObject в Swift, который обычно может содержать только значения типов классов. Swift 2 также предоставил неявные преобразования в AnyObject для некоторых типов значений моста, таких как String, Array, Dictionary, Set и некоторых чисел, в качестве удобства, чтобы собственные типы Swift можно было легко использовать с API-интерфейсами Какао, которые ожидали NSString, NSArray, или другие классы-контейнеры из Foundation. Эти преобразования несовместимы с остальным языком, что затрудняет понимание того, что именно можно использовать в качестве AnyObject, что приводит к ошибкам .

Однако можно было бы настаивать на том, что мы, разработчики iOS, всегда должны быть как можно более конкретными с точки зрения использования типов в коде.

Действительно, Apple рекомендует:

Используйте Any и AnyObject только тогда, когда вам явно необходимы поведение и возможности, которые они предоставляют. Всегда лучше конкретизировать типы, с которыми вы ожидаете работать в своем коде .

Подумайте об этом сценарии: мы работаем с числом 12,5 в Swift. В этом случае мы бы специально заявили, что это тип Double или Float, а не объявляли бы его типа Any . Таким образом, мы можем получить удобный доступ к различным свойствам или методам, доступным для этого конкретного типа. В этом контексте мы бы использовали AnyObject для типов классов, потому что они немного более специфичны, чем Any . Но опять же, использование AnyObject - это всего лишь вариант.

Пример использования Any с различными типами

Задание для самостоятельного выполнения

1. Создайте на основе примера выше массив с информацией об одном экземпляре из Вашего задания

2. Числовое значение увеличьте на 10

3. Название и прочие характеристики напечатайте в консоле

4. Отдельно выведите весь массив

**Ход работы:**

Создание массива с информацией о заданном экземпляре:

var instanceInfo: [Any] = ["Car", "Toyota", 2015, 25000.0]

Увеличение числового значения на 10:

if let year = instanceInfo[2] as? Int {

instanceInfo[2] = year + 10

}

Печать названия и прочих характеристик в консоли:

swift

Копировать код

if let name = instanceInfo[0] as? String, let brand = instanceInfo[1] as? String, let year = instanceInfo[2] as? Int, let price = instanceInfo[3] as? Double {

print("Name: \(name)")

print("Brand: \(brand)")

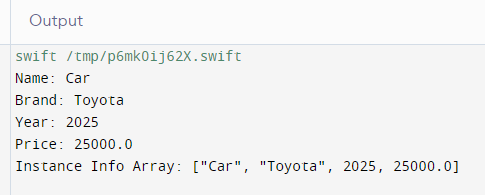
print("Year: \(year)")

print("Price: \(price)")

}

Печать всего массива:

print("Instance Info Array: \(instanceInfo)")



**Контрольные вопросы:**

1. Какие типы данных можно хранить в переменных типа Any и AnyObject?
2. Какие операторы используются для приведения типов в Swift?
3. В чем отличие между Any и AnyObject?
4. Когда следует использовать тип Any или AnyObject в коде?
5. Как можно преобразовать тип Any обратно в конкретный тип данных?

Варианты

1. Автобус (номер маршрута, название начальной и конечной остановки, количество остановок по маршруту)

2. Книга (название книги, автор, издательство, количество страниц)

3. Файл (имя, время создания, дата создания, размер)

4. Спортсмен (ФИО, вид спорта, сколько лет занимается)

5. Студент (ФИО, группа, средний балл зачетки)

6. Фирма (название, вид услуг, годовой оборот)

7. Компьютер (производитель, частота процессора, количество ядер, объем памяти)

8. Спорт (вид спорта, количество занимающихся в городе, средняя стоимость комплекта инвентаря)

9. Автомобиль (марка, фирма-производитель,год выпуска, пробег)

10. Статья (название, автор, название журнала, год и месяц издания)

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 14** Разработка мобильного калькулятора калорий

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

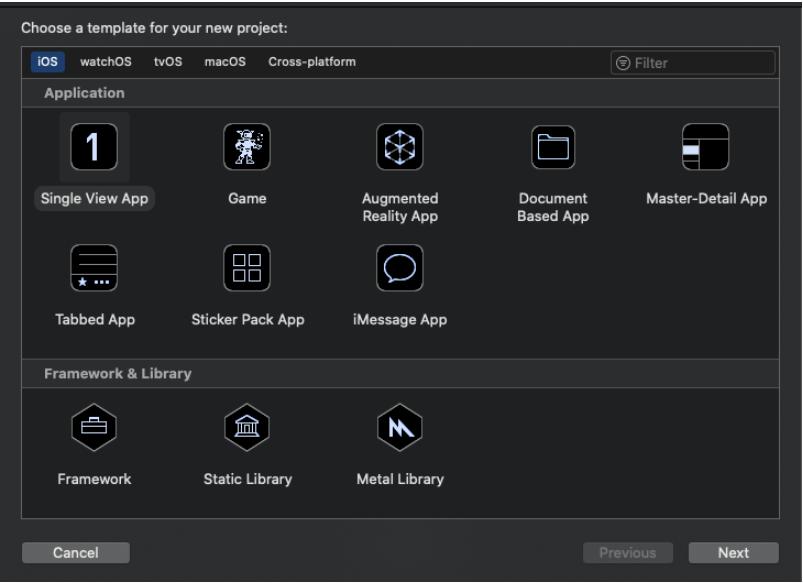
ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы:** научиться использовать storyboard, создавать приложения c визуальным интерфейсом.

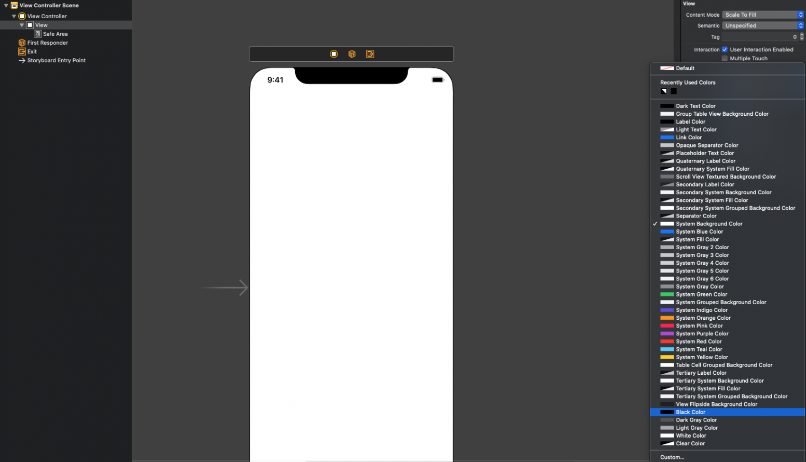
**Ход работы:**

Запускаете новый проект и выбираете Single Page Application:



После того, как вы создали приложение, можно сразу вызвать эмулятор для того, чтобы отслеживать, что получается. Для этого можете просто нажать сочетание клавиш CMD + R. Слева вверху вы можете выбирать устройства, на которых вы запускаете эмулятор, а также можете через кабель подключить ваше настоящее физическое устройство и тестировать сразу же на нем).

Существует много способов создания интерфейса и виджетов в нем, я воспользуюсь нейтральным вариантом — создам основной holder, размещу его на весь экран, а потом программно начну располагать свои виджеты. Поэтому сначала мы переходим в файл Main.storyboard и делаем основной View черным, для того, чтобы удобнее размещать наши элементы:



После этого мы добавим View, с которым в основном и будем работать. Для этого кликаем на плюсик справа сверху, через который можно добавлять виджеты, и добавляем новый виджет, наберем в поиске uiview:

После этого мы перетаскиваем этот виджет на основной экран, и нам нужно расширить его на весь экран. Для этого нам нужно поработать с constaint. Для этого щелкаем по значку слева внизу, который немного похож на кораблик Дарта Вейдера и назначить 0 с каждой из сторон:

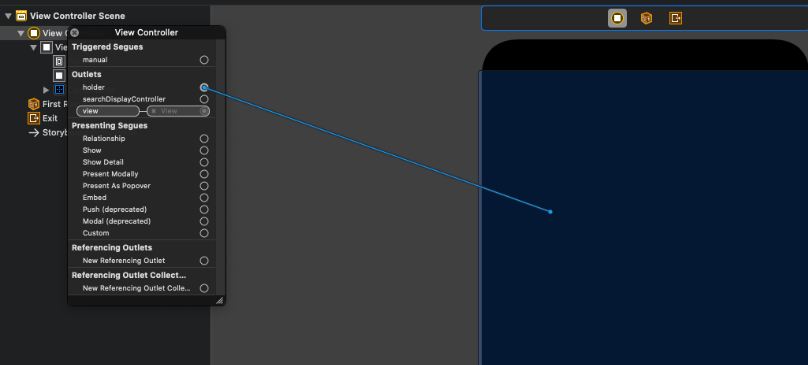


После этого ваш новый виджет расширится до размеров всего экрана. Дальше нужно также, как и в прошлый раз, поменять цвет нашего виджета, только теперь на Clear Color, чтобы он нам не мешался и мы могли размещать виджеты в нем.

Теперь можно переходить в файл ViewController.swift, в котором у нас и будет происходить все остальное программирование. Для начала нам нужно использовать специальный маркер @IBOutlet для того, чтобы иметь возможность связать Storyboard c нашим кодом. Для этого мы должны написать следующую строчку:

@IBOutlet var holder: UIView!

Теперь мы можем связать код с виджетом. Для этого, по сути, нужно соединить наш holder c Holder. Для этого щелкаем правой кнопкой по viewController и перетягиваем holder на наш экран:



Наконец-то мы закончили связывание кода с Storyboard и можно заняться именно программированием.

import UIKit

class ViewController: UIViewController {

@IBOutlet weak var foodTextField: UITextField!

@IBOutlet weak var caloriesTextField: UITextField!

@IBOutlet weak var totalCaloriesLabel: UILabel!

var totalCalories: Int = 0

override func viewDidLoad() {

super.viewDidLoad()

}

@IBAction func addButtonTapped(\_ sender: UIButton) {

guard let food = foodTextField.text, let caloriesStr = caloriesTextField.text, let calories = Int(caloriesStr) else {

return

}

totalCalories += calories

totalCaloriesLabel.text = "Total Calories: \(totalCalories)"

// Очистка полей ввода

foodTextField.text = ""

caloriesTextField.text = ""

}

}

Этот код создает калькулятор калорий, который позволяет пользователю вводить продукты и их калорийное содержание. При нажатии на кнопку "Добавить", введенные продукты и калории добавляются к общему количеству калорий, которое отображается на метке totalCaloriesLabel.

**Контрольные вопросы:**

1. Какие инструменты можно использовать для создания мобильного приложения?
2. Что такое Storyboard и для чего он используется в разработке iOS-приложений?
3. Какие элементы интерфейса могут использоваться в мобильном калькуляторе калорий?
4. Как осуществляется связывание элементов интерфейса с кодом приложения?
5. Какие этапы включает в себя разработка мобильного приложения?

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 15.** Программная работа с элементами экрана (View).

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы:** научиться использовать библиотеку визуальных компонентов

**Ход работы:**

Создание интерфейса пользователя в Storyboard:

Откройте Xcode и создайте новый проект.

Откройте Main.storyboard и создайте пользовательский интерфейс, добавляя различные элементы, такие как кнопки, метки, текстовые поля и изображения.

Настройте внешний вид элементов и их расположение на экране.

Создание связей с элементами интерфейса:

Откройте файл ViewController.swift.

Создайте IBOutlet и IBAction для каждого элемента интерфейса, с которым вы хотите взаимодействовать программно.

Свяжите созданные IBOutlet и IBAction с соответствующими элементами интерфейса в Storyboard.

Пример кода для создания IBOutlet и IBAction:

@IBOutlet weak var myLabel: UILabel!

@IBOutlet weak var myButton: UIButton!

@IBAction func buttonTapped(\_ sender: UIButton) {

// Обработка нажатия на кнопку

myLabel.text = "Button tapped!"

}

Работа с элементами интерфейса в коде:

Используйте созданные IBOutlet для получения доступа к свойствам элементов интерфейса.

Используйте IBAction для реагирования на действия пользователя, такие как нажатие на кнопку.

Манипулируйте свойствами элементов интерфейса, такими как текст метки или изображение кнопки, в соответствии с логикой вашего приложения.

Пример кода для изменения свойств элементов интерфейса:

myLabel.text = "New text"

myButton.setTitle("New Title", for: .normal)

Обработка пользовательского ввода:

Используйте методы делегата или замыкания для обработки ввода пользователя, например, при редактировании текстовых полей.

Реализуйте логику для валидации и обработки введенных данных.

Пример кода для работы с текстовым полем и делегатом:

class ViewController: UIViewController, UITextFieldDelegate {

@IBOutlet weak var textField: UITextField!

override func viewDidLoad() {

super.viewDidLoad()

textField.delegate = self

}

func textFieldShouldReturn(\_ textField: UITextField) -> Bool {

textField.resignFirstResponder() // Скрыть клавиатуру при нажатии на Return

return true

}

func textFieldDidEndEditing(\_ textField: UITextField) {

// Обработка введенных данных после завершения редактирования

}

}

Обновление интерфейса:

Используйте методы жизненного цикла ViewController, такие как viewDidLoad() и viewWillAppear(), для обновления элементов интерфейса перед их отображением на экране.

Используйте метод setNeedsDisplay() для перерисовки элементов интерфейса в случае необходимости.

Пример кода для обновления интерфейса:

override func viewDidLoad() {

super.viewDidLoad()

updateUI()

}

func updateUI() {

// Обновление интерфейса

}

Это основные шаги для программной работы с элементами экрана в iOS приложениях.

Контрольные вопросы:

1. Что такое IBOutlet и IBAction в iOS разработке?
2. Как создать связь между элементами интерфейса и кодом в Swift?
3. Какие методы жизненного цикла ViewController используются для обновления интерфейса?
4. Что такое метод делегата UITextFieldDelegate и как его использовать?
5. Каким образом можно обработать нажатие на кнопку в iOS приложении?

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 16** Создание обработчиков событий

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы:** научиться работать со сценами (экранами)

**Ход работы:**

Пример 1: Последовательные переходы от View к View на несколько уровней в глубь

На первом View мы объявляем NavigationView в который заворачиваем ссылку (NavigationLink) на следующий View.

Здесь единственным отличием первого View от остальных, является объявление NavigationView, так как в остальных View его объявлять уже не требуется, ввиду применения наследования (вспоминаем "коробку с шариками" из начала статьи)

// первый экран

struct View1: View {

var body: some View {

// объявляем NavigationView единожды на первом экране

NavigationView {

NavigationLink {

View1\_1()

} label: {

Text("Переход на View1\_1")

}

.navigationTitle("View1")

}

}

}

// второй экран

struct View1\_1: View {

var body: some View {

NavigationLink {

View1\_2()

} label: {

Text("Переход на View1\_2")

}

.navigationTitle("View1\_1")

}

}

// третий экран

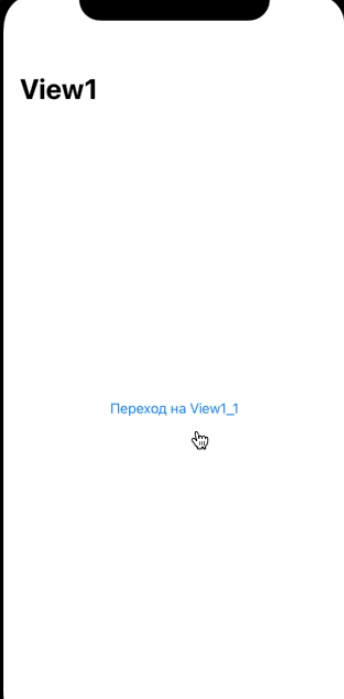
struct View1\_2: View {

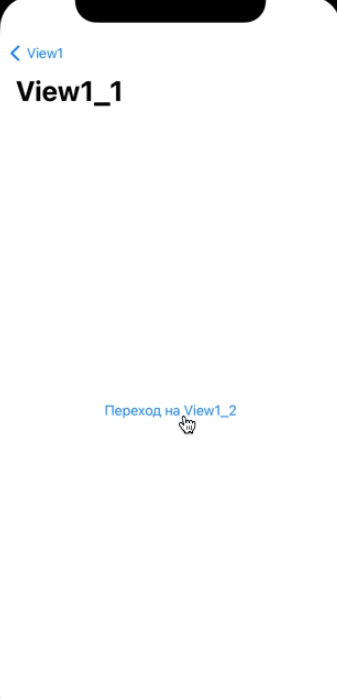
var body: some View {

Text("Последний экран")

}

}







Варианты заданий:

1. Кнопка "Привет": Создайте приложение с одной кнопкой "Привет". При нажатии на кнопку должен появляться всплывающий диалог с сообщением "Привет, мир!".
2. Счетчик касаний: Разместите на экране кнопку "Касание" и метку (UILabel). При каждом нажатии на кнопку увеличивайте значение счетчика на 1 и отображайте его в метке.
3. Изменение цвета: Создайте приложение с двумя кнопками: "Синий" и "Красный". При нажатии на каждую из кнопок фон приложения должен менять цвет на соответствующий: синий или красный.
4. Показать/Скрыть текст: Разместите кнопку "Показать" и метку (UILabel) с каким-либо текстом. При нажатии на кнопку текст в метке должен появляться или исчезать.
5. Простой калькулятор: Создайте простое приложение калькулятора с двумя текстовыми полями для ввода чисел и четырьмя кнопками для операций: "+", "-", "\*" и "/". После ввода чисел и выбора операции приложение должно вычислить результат и отобразить его на экране.

**Контрольные вопросы:**

1. Что такое обработчики событий в контексте разработки мобильных приложений?
2. Какие типы UI-элементов могут иметь обработчики событий в iOS?
3. Как создать обработчик события для кнопки в интерфейсе при помощи Interface Builder (Storyboard)?
4. Как можно создать обработчик события программно в коде?
5. Как передать параметры в обработчик события?
6. Как отключить обработчик события для определенного UI-элемента?

Критерии оценки:

**«5» «отлично»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* проявил активность в освоении новых практических умений;
* содержание отчета по практическим работам полноценно отражает информацию по выполненным видам работ, установленных программой практики;
* оформление отчета по практическим работам полностью соответствует предъявляемым требованиям;
* при выполнении практических работ студент уверенно продемонстрировал обладание необходимыми компонентами общих и профессиональных компетенций;

**«4» «хорошо»**

* студент продемонстрировал готовность и способность к самостоятельному выполнению видов работ в стандартных ситуациях;
* все работы выполнил правильно и уверенно;
* содержание отчета по практическим работам в целом правильно отражает информацию по выполненным видам работ;
* оформление отчета по практическим работам соответствует предъявляемым требованиям, однако имеются замечания (некоторая неаккуратность, неполное описание проделанной работы и освоенных умений);
* при выполнении практических работ студент продемонстрировал обладание компонентами общих и профессиональных компетенций;
* характеристики студента положительные, имеются отдельные замечания и рекомендации;

**«3» «удовлетворительно»**

* при выполнении видов работ студент нуждался во внешнем сопровождении и контроле;
* в отчете по практическим работам содержится описание видов выполняемых работ согласно программе дисциплины, однако, записи неполные и неточные;
* отчет по практическим работам оформлен с отклонениями от установленных требований;
* при выполнении практических работ студент продемонстрировал обладание отдельными компонентами общих и профессиональных компетенций;

«**2» «неудовлетворительно»**

– студент не справился с запланированными видами работ;

– при выполнении практических работ студент продемонстрировал слабое обладание отдельными компонентами общих и профессиональных компетенций;

– отчет по итогам окончания дисциплины студентом не подготовлен.

**Практическое занятие № 17** Создание мобильного приложения для точки продаж. Разработка UI.

**Формируемые ОК, ПК**

ПК 3.1 Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 3.2 Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт.

ПК 3.3 Проектировать и разрабатывать базы данных для мобильных платформ.

ПК 3.4 Осуществлять внедрение мультимедиа в программное обеспечение для мобильнвх платформ.

ПК 3.5 Выполнять тестирование и отладку программного обеспечения.

ПК 3.6 Выполнять интеграцию разработанного приложения с внешними системами и платформами.

ПК 3.7 Осуществлять защиту данных в мобильные приложения.

ОК 1 Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам

ОК 2 Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности

ОК 3 Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях

ОК 4 Эффективно взаимодействовать и работать в коллективе и команде

ОК 5 Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста

ОК 9 Пользоваться профессиональной документацией на государственном и иностранном языках.

**Цель работы :** научиться разрабатывать мобильное приложения для точки продаж

**Ход работы:**

Создание списка преподавателей

Во-первых, необходимо создать представление для отображения списка всех участников команды, включая фотографии их профиля и их описание. Посмотрим, как это можно сделать.

Как мы видим, в сгенерированном коде имеется компонент Text со значением «Hello World». В редакторе кода изменим значение кода на «Simon Ng».

struct ContentView: View {

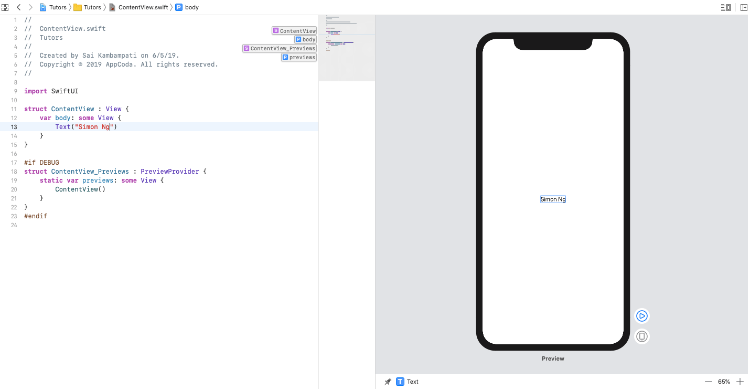
var body: some View {

Text("Simon Ng")

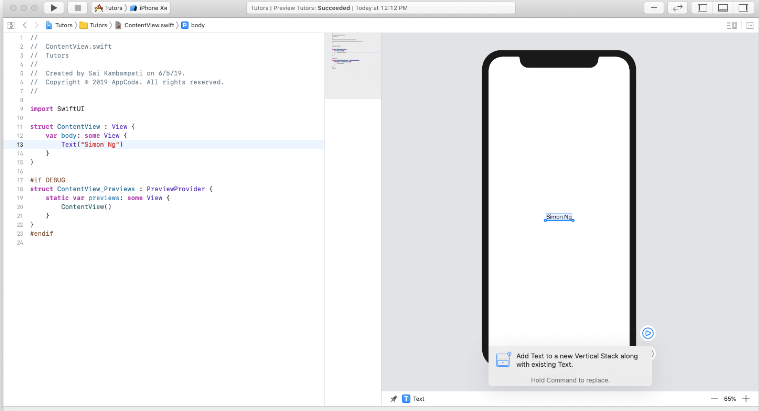
}

}

Если все работает верно, справа вы должны увидеть автоматическое обновление. Это эффект мгновенного просмотра, что мы и ожидали.



Давайте добавим в приложении новый элемент Text. Это будет краткое описание участника. Чтобы в приложении добавить новый элемент интерфейса, необходимо нажать кнопку + в правом верхнем углу. Появится новое окно со списком различных вью. Переместим вью с названием Text и поместим его под первоначальный элемент Text, как показано ниже.



Обратите внимание на код слева:

struct ContentView: View {

var body: some View {

VStack {

Text("Simon Ng")

Text("Placeholder")

}

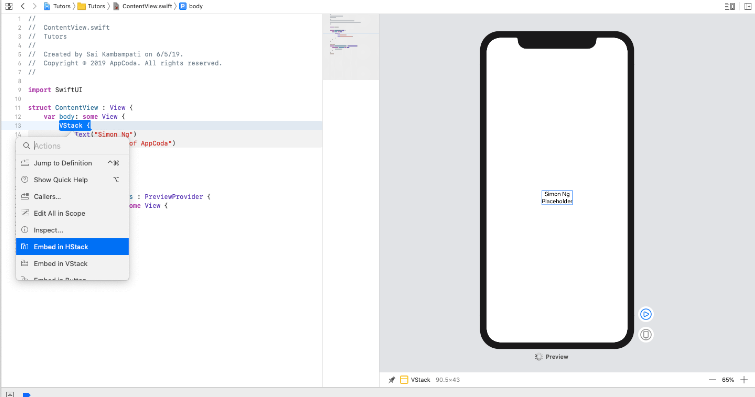
}

}

Можно заметить, что новый элемент Text был добавлен под Text вью с значением Simon Ng. Отличие состоит в том, что теперь это вью, похоже, обернул представление в нечто, называемое VStack. VStack используется для вертикального стека, и он является заменой Auto Layout в SwiftUI. Если у Вас имеется опыт разработки программного обеспечения для watchOS, вы вероятно знаете, что здесь нет никаких ограничений, более того все элементы помещаются в группы. При вертикальном стеке все вью будут расположены вертикально.

Теперь измените текст «Placeholder» на «Founder of AppCoda»

Далее, давайте добавим изображение слева от этого текста. Так как мы хотим расположить представление горизонтально к существующим представлениям, то имеется необходимость обернуть VStack в HStack. Для этого, выполним ⌘+Click на VStack, а затем выберем Embed in HStack. Посмотрим на это ниже:



Данный код должен выглядеть следующим образом:

struct ContentView: View {

var body: some View {

HStack {

VStack {

Text("Simon Ng")

Text("Founder of AppCoda")

}

}

}

}

Значительных изменений пока нету, но сейчас мы добавим изображение. Измените код, чтобы он выглядел следующим образом:

struct ContentView: View {

var body: some View {

HStack {

Image(systemName: "photo")

VStack {

Text("Simon Ng")

Text("Founder of AppCoda")

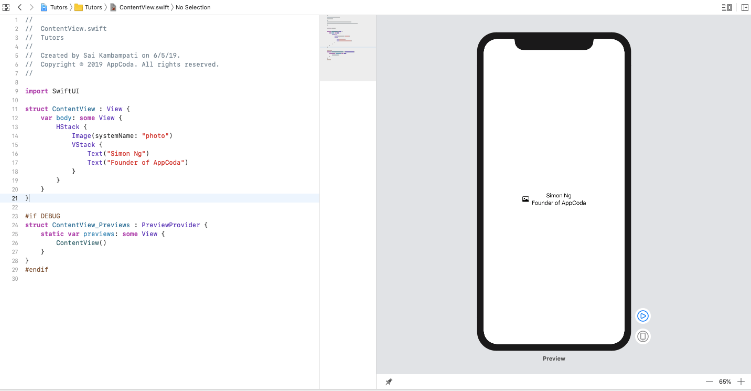
}

}

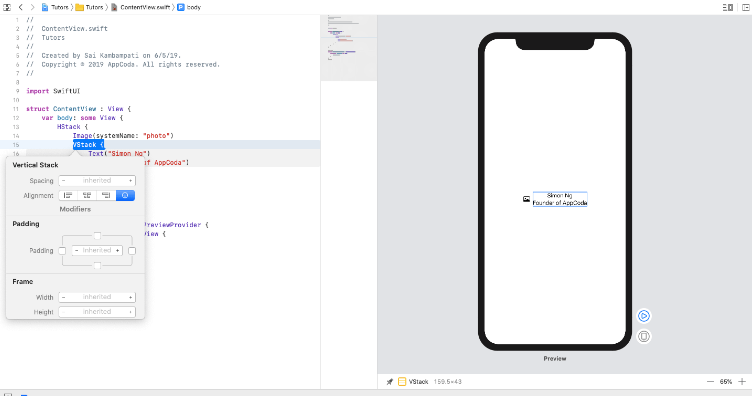
}

}

Начиная с iOS 13, Apple представляет новую функцию под названием SFSymbols. SF Symbols, разработанный компанией Apple, представляет собой набор из более чем 1500 символов, которые можно использовать в приложениях. Поскольку они могут легко интегрироваться с системным шрифтом San Francisco, символы автоматически обеспечивают оптическое вертикальное выравнивание с текстом любого размера. Поскольку у нас пока нет изображений наших преподавателей, будем использовать так называемый placeholder.



Теперь сосредоточимся на некоторых незначительных проблемах дизайна. Поскольку имеется необходимость эмуляции внешнего вида **UITableRow**, давайте выровняем текст по левому краю (т. е. сделаем его главным). Для этого выполним **⌘+Click** на VStack и нажмем **Inspect**. Выберем значок выравнивания по левому краю, как показано ниже:



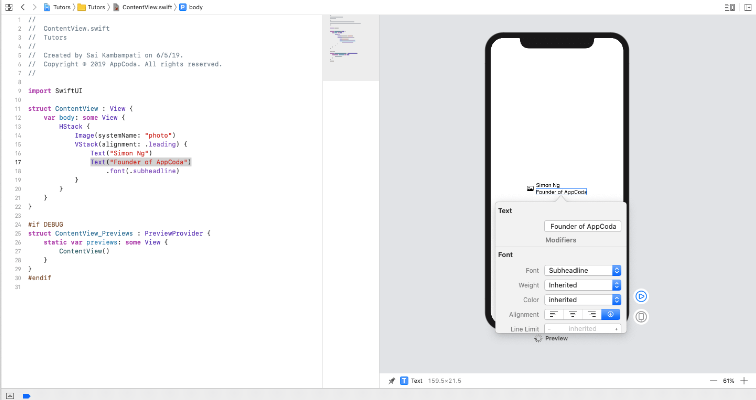
Далее увидим изменение в коде. Также код будет изменен в реальном времени для отображения новых изменений.

VStack(alignment: .leading) {

...

}

Теперь, когда второе текстовое представление является заголовком, давайте изменим шрифт. Как и раньше, ⌘+Click на текстовом представлении «Founder of AppCoda» в режиме предварительного просмотра и выбираем Inspect. Изменим шрифт на «Subheadline» и отобразим предварительный просмотр и изменение кода в реальном времени.



Давайте также изменим цвет и установим его на «Серый». Данный код должен выглядеть следующим образом:

struct ContentView: View {

var body: some View {

HStack {

Image(systemName: "photo")

VStack(alignment: .leading) {

Text("Simon Ng")

Text("Founder of AppCoda")

.font(.subheadline)

.color(.gray)

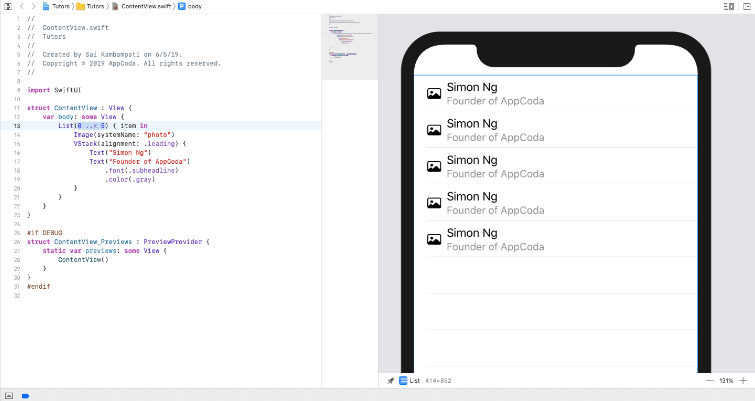
}

}

}

}

Теперь, после окончания проектирования ряда сэмплов, мы подошли к волшебной части. Посмотрите, как легко создать список. Выполним ⌘+Click на HStack и выполним клик на Embed in List. Вуаля! Посмотрите, как код будет автоматически меняться, и пустая область будет отображать 5 красивых новых строк, каждая из которых показывает Simon Ng в качестве члена команды.



Варианты заданий

1) Продуктовый супермаркет

2) Алкомаркет

3) Автозаправка

4) Шиномонтаж

5) Кофейня

6) Книжный магазин

7) Автопрокат

8) Прокат велосипедов

9) Спортлото

10) Вкуснолюбов

11) Курсы автовождения

12) Шаурма хауз

13) Кальяная

14) Ипподром

15) Плавательный бассейн

16) Бюро ритуальных услуг

17) Магазин волшебных палочек и магических товаров от ГаррПоттера

18) «Фрекен Бок» - агенство нянь и гувернанток

19) Прокат сегвеев

20) Турфирма

21) Бюро переводов

22) Юридическая консультация

23) Пицерия

24) Заказ и доставка букетов

25) Домашний репетитор

26) Прогулки на воздушном шаре

27) Аренда прогулочных катеров и яхт

28) Организация праздников

29) Украшение праздников, офисов и мероприятий воздушными шарами

30) Вызов мастера для ремонтных работ на дом «Муж на час»

**Контрольные вопросы итоговой аттестации**

1. Как описать константы?

2. Как описать переменные? Перечислите и поясните типы данных Swift

3. Перечислите и поясните операторы контроля управления

4. Что такое замыкание и для чего его применяют?

5. Что такое перечисления и в чем их отличия от структур?

6. Для чего нужны протоколы и расширения?

7. Что такое общие функции и типы?

8. Что такое Any и AnyObject?

9. Что такое "Опциональный тип"?

10. Что такое принудительное разыменование и опциональное связывание?

11. Каково назначение Unwind segue?

12. Из чего состоит архитектура IOS-приложения?

13. Для чего используется класс UINavigationController?

14. Для чего используется класс View?

15. Что такое Unwind segue в интерфейсе IOS-приложения?

16. Что такое outlet?

17. Какие кнопки используются в интерфейсе IOS-приложения?

18. Какие типы входят в коллекции Swift?

19. Как можно обратиться к элементу экрана из кода?

20. Что такое action в IOS-приложении?

21. Объектом какого класса сцена?

22. Как создается переход между сценами?

23. Как добавить контроллер навигации и указать точку входа?

24. Какие визуальные элементы используются в IOS-приложениях?

25. Перечислите основные компоненты IOS-приложения?

26. Что такое Acton?

27. Какие методы вызываются системой для управления жизненным циклом?

28. Каковы возможности и отличия классов и структур в Swift?

29. Как добавить View, и зачем оно нужно?

30. Что такое push-уведомления?

**Задачи для практической части**

1. Напишите программу на Swift, которая выводит на экран приветственное сообщение с именем пользователя.

2. Создайте приложение на Swift, которое отображает список фруктов и их цену, используя UITableView.

3. Разработайте приложение на Swift, которое использует GPS для определения текущего местоположения пользователя.

4. Напишите программу на Swift, которая позволяет пользователю вводить свой вес и рост, а затем рассчитывает индекс массы тела.

5. Создайте приложение на Swift, которое позволяет пользователю делать фотографии и сохранять их в галерею устройства.

6. Разработайте игру на Swift, используя SpriteKit или SceneKit.

7. Напишите приложение на Swift, которое подключается к API и отображает данные в виде списка или карточек.

8. Создайте приложение на Swift, которое позволяет пользователю совершать покупки через Apple Pay.

9. Разработайте приложение на Swift, которое отображает актуальный курс валюты и позволяет пользователю конвертировать суммы.

10. Напишите программу на Swift, которая генерирует случайные числа и отображает их на экране.

11. Создайте приложение на Swift, использующее CoreData для хранения данных локально на устройстве.

12. Разработайте приложение на Swift, которое позволяет пользователю записывать голосовые заметки.

13. Напишите программу на Swift, которая создает калькулятор для выполнения базовых математических операций.

14. Создайте приложение на Swift, которое позволяет пользователю создавать и управлять списками задач (to-do list).

15. Разработайте игру на Swift, используя ARKit для создания дополненной реальности.

16. Напишите приложение на Swift, которое интегрируется с социальными сетями для публикации контента.

17. Создайте приложение на Swift, которое использует CoreLocation для отображения ближайших точек интереса на карте.

18. Разработайте приложение на Swift, которое позволяет пользователям обмениваться сообщениями в реальном времени.

19. Напишите программу на Swift, которая позволяет пользователю создавать и настраивать свой профиль.

20. Создайте приложение на Swift, которое предоставляет пользователю возможность заказать такси через API сервиса такси.

**Пример контрольного задания**

Министерство общего и профессионального образования Ростовской области

ГБПОУ РО Ростовский-на-Дону колледж связи и информатики

|  |  |  |
| --- | --- | --- |
| **ОДОБРЕНО**  Цикловой комиссией  Программирования  Протокол №\_\_\_\_  от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С. Сулавко | **Экзаменационный билет**  **по *МДК.03.01 «Разработка мобильных приложений»***  **для специальности 09.02.11 «Разработка и управление программным обеспечением»**  **Vсеместр**  **Вариант №1** | **УТВЕРЖДАЮ**  Заместитель директора  по научно-методической работе  И.В. Подцатова  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2024г. |

Каково назначение Unwind segue?

Из чего состоит архитектура IOS-приложения?

**Практическая часть**

Напишите программу на Swift, которая создает калькулятор для выполнения базовых математических операций.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2024г. Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Критерии оценки экзаменационного билета:**

* оценка **«отлично»** выставляется студенту, если он смог выполнить работу безошибочно;
* оценка **«хорошо»** выставляется студенту, если он смог выполнить работу с незначительными ошибками;
* оценка **«удовлетворительно»** выставляется студенту, если он выполнил работу со значительными ошибками или не выполнена половина заданий;
* оценка **«неудовлетворительно»** выставляется студенту, если он не выполнил большую часть заданий

# **Контроль приобретения практического опыта. Оценка по учебной и (или) производственной практике (по профилю специальности)**

**4.1. Общие положения**

Целью оценки по учебной и (или) производственной практике (по профилю специальности) является оценка:

1) профессиональных и общих компетенций;

2) практического опыта и умений.

Оценка по учебной и (или) производственной практике выставляется на основании данных аттестационного листа (характеристики профессиональной деятельности обучающегося/студента на практике) с указанием видов работ, выполненных обучающимся во время практики, их объема, качества выполнения в соответствии с технологией и (или) требованиями организации, в которой проходила практика.

**4.2. Виды работ практики и проверяемые результаты обучения по частично вариативному профессиональному модулю ПМ.11«Технология разработки и защиты баз данных»**

Таблица 7

|  |  |  |
| --- | --- | --- |
| **Иметь практический опыт** | **Виды и объем работ на учебной практике, требования к их выполнению и/ или условия выполнения** | **Документ, подтверждающий качество выполнения работ** |
| **1** | **2** | **3** |
| * в работе с объектами базы данных в конкретной системе управления базами данных; * использовании стандартных методов защиты объектов базы данных; * работе с документами отраслевой направленности | * сбор, обработка и анализ информации для проектирования баз данных; * проектирование базы данных на основе анализа предметной области; * разработка объектов базы данных в соответствии с результатами анализа предметной области; * реализация базы данных в конкретной системе управления базами данных; * администрирование базы данных;   защита информации в базе данных с использованием технологии защиты информации. | * аттестационный лист о прохождении практики или справка с места работы. |

Таблица 8

|  |  |  |
| --- | --- | --- |
| **Иметь практический опыт** | **Виды и объем работ на производственной практике (по профилю специальности), требования к их выполнению и/ или условия выполнения** | **Документ, подтверждающий качество выполнения работ** |
| **1** | **2** | **3** |
| * В работе с объектами базы данных в конкретной системе управления базами данных; * Использовании стандартных методов защиты объектов базы данных; * В работе с документами отраслевой направленности | * Получение задания, сбор и описание требований к разработке ПП по заданию предприятия и согласованию с руководителем практики или собственная тема, если выбрана и утверждена с дипломным руководителем. * Проектирование базы данных, соответствующей требованиям, с использованием современных case-средств: * Разработка концептуальной, логической и физической модели базы данных. * Формирование схемы данных. * Реализация базы данных в конкретной системе управления базами данных (СУБД): * Описание СУБД, в которой реализована база данных, обоснование выбора СУБД. * Описание способов обеспечения достоверности информации в базе данных. * Разработка объектов базы данных. * Применение стандартных методов для защиты объектов базы данных. * Выполнение процедур резервного копирования и восстановления базы данных. * Обеспечение информационной безопасности на уровне базы данных. * Создание хранимых процедур и триггеров в базе данных. | * аттестационный лист о прохождении практики или справка с места работы. |

# **Контрольно-оценочные материалы для квалификационного экзамена**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| СВОДНАЯ ВЕДОМОСТЬ ПО ПРОФЕССИОНАЛЬНОМУ МОДУЛЮ  **ПМ.03 Разработка приложений для мобильных платформ**  Ф.И.О.«\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»  студент на 4 курсе по специальности СПО 09.02.11 «Разработка и управление программным обеспечением»  освоил программу профессионального ПМ.03 Разработка приложений для мобильных платформв объёме 342 час. с«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г. по «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_г.  Результаты промежуточной аттестации по элементам профессионального модуля | | | | |
| **Элементы модуля** | | **Формы промежуточ-**  **ной аттестации** | **Оценка** | |
| МДК.03.01 «Разработка мобильных приложений» | | Экзамен |  | |
| УП.03.01 Учебная практика | | Зачет |  | |
| ПП.03.01 Производственная практика | | Зачет |  | |
| ПМ.03 | | Квалификационный экзамен |  | |
| Итоги экзамена по профессиональному модулю | | | | |
| **Коды проверяемых компетенций** | **Показатели оценки результата** | | | **Оценка (освоен / не освоен)** |
| ПК 3.1 | Разрабатывать модули программного обеспечения для мобильных платформ. | | | освоен |
| ПК 3.2 | Проектировать и разрабатывать пользовательский интерфейс и пользовательский опыт. | | | освоен |
| ПК 3.3. | Проектировать и разрабатывать базы данных для мобильных платформ. | | | освоен |
| ПК 3.4. | Осуществлять внедрение мультимедиа в программное обеспечение для мобильных платформ. | | | освоен |
| ПК 3.5. | Выполнять тестирование и отладку программного обеспечения. | | | освоен |
| ПК 3.6. | Выполнять интеграция разработанного приложения с внешникми системами и платформами. | | | освоен |
| ПК 3.7. | Осуществлять защиту данных в мобильных приложениях. | | | освоен |
| ОК 1 | Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам | | | освоен |
| ОК 2 | Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности | | | освоен |
| ОК 3 | Планировать и реализовывать собственное профессиональное и личностное развитие, предпринимательскую деятельность в профессиональной сфере, использовать знания по финансовой грамотности в различных жизненных ситуациях | | | освоен |
| ОК 4 | Эффективно взаимодействовать и работать в коллективе и команде | | | освоен |
| ОК 5 | Осуществлять устную и письменную коммуникацию на государственном языке Российской Федерации с учетом особенностей социального и культурного контекста | | | освоен |
| ОК 9 | Пользоваться профессиональной документацией на государственном и иностранном языках. | | | освоен |
| Члены аттестационной комиссии  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | |

# 