Практическая работа № 11

**Опциональные типы**

Цель работы:

1. Изучить опциональные типы, научиться применять их на практике.
2. Получить навыки использования опциональных типов.
3. Создать консольное приложение, работа которого будет построена на использовании опциональные типы

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](https://en.wikipedia.org/wiki/Nullable_type) 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:

|  |  |
| --- | --- |
| 1  2 | 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:

|  |  |
| --- | --- |
| 1  2 | 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:

|  |  |
| --- | --- |
| 1 | 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:

|  |  |
| --- | --- |
| 1  2  3 | 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:

|  |  |
| --- | --- |
| 1  2 | 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:

|  |  |
| --- | --- |
| 1  2  3 | 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:

|  |  |
| --- | --- |
| 1  2  3 | vara: Int? = 12  varb: Int= 10  varc = a! + b      // с = 22 |

Другой пример:

|  |  |
| --- | --- |
| 1  2  3  4  5 | 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):

|  |  |
| --- | --- |
| 1  2  3  4  5 | 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:

|  |  |
| --- | --- |
| 1  2  3  4  5 | 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 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:

|  |  |
| --- | --- |
| 1  2 | 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:

|  |  |
| --- | --- |
| 1  2  3 | else{      print(b)  } |

Но также в данном случае мы могли и по другому проверить на значение nil:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 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:

|  |  |
| --- | --- |
| 1  2  3  4 | leta: Int? = 10  ifa > 5{      print("a is greater than 5")  } |

И в подобных операциях к объекту Optional необходимо применить оператор !:

|  |  |
| --- | --- |
| 1  2  3  4 | 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 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