Основы Swift

Control Flow (Управление потоком)

Выражения для управления

Управление осуществляется с помощью следующих выражений

Операторы ветвления

- if
- guard
- switch

Циклы

- forin
- while

Выражениями для передачи контроля

- continue
- break
- fallthrough
- return
- throw

Циклы For-In (коллекции)

for in для доступа к элементам коллекций

Массивы и Множества — получаем отдельный элемент коллекции

```
let digits = [1, 2, 3]

for element in digits {
    print(element)
}

//1
//2
//3
```

Словари — получаем кортеж вида (Ключ, Значение)

```
let keyValues = ["One" : 1, "Two" : 2,
"Three" : 3]

for element in keyValues {
    print(element)
}

// (key: "One", value: 1)
// (key: "Three", value: 3)
// (key: "Two", value: 2)
```

Циклы For-In (использование диапазонов)

Диапазоны для создания циклов с определенным количеством итераций.

```
let array = [1, 2, 3, 4, 5]
for index in 1...3 {
    print(array[index])
}

// 2
// 3
// 4
```

значение передается в index, если index не нужен заменяем на _

```
for _ in 1...3 {
    print("Hello")
}
```

Циклы While и Repeat-While

Цикл выполняется пока условие не примет значение false

```
var value = 0

while (value < 10) {
    value += 1
    print(value)
}
```

```
var value = 0

repeat {
    value += 1
    print(value)
} while (value < 10)
```

Операторы ветвления

Предназначены для выполнения частей кода
в зависимости от условий.

if
switch
guard

Операторы ветвления (if - else if - else)

Гибкая конструкция, можно использовать только if или if else.

if — проверка простого условия
if else — проверка условия и выполнение альтернативного кода, при невыполнении условия
if - else if — для проверки нескольких условий. после выполнения одного из условий, выход из блока.

После if или else if мы добавляем условие, возвращающее true или false. Условия можно объединять с помощью логических операторов.
&& для проверки на выполнение двух условий
|| для проверки выполнения одного из двух условий.
После else условие не добавляется, это выражение будет выполнено только в том случае, если все остальные условия до этого были провалены.

Операторы ветвления (if - else if - else)

```
let x = 50

if x == 1 {
    print("One")
} else if x == 2 {
    print("Two")
} else if x < 10 && x >= 0 {
    print("X где-то от 0 до 10")
} else {
    print("X больше 10")
} // X больше 10
```

Операторы ветвления (Switch)

```
let char = "C"

switch char {
case "A":
    print("похоже на A")
case "B":
    print("похоже на B")
default:
    print("Ни A, ни B")
}
```

Операторы ветвления (Switch интервалы)

Интервалы можно подставлять в case.

```
let value = 6

switch value {
    case 0...5:
        print("Значение от 0 до 5")
    case 6...10:
        print("Значение от 6 до 10")
    default:
        print("Не входит в проверяемые
        диапазоны")
}
```

Операторы ветвления (Switch tuples)

Использование кортежей в switch case

```
let coordinate = (0, 1)

switch coordinate {
case (0, 0):
    print("Начало координат")
case (_, 0):
    print("Лежит на оси Y")
case (0, -10...10):
    print("На оси X в диапазоне от -10 до 10")
default:
    print("Где-то рядом")
}
```

Операторы ветвления (Switch составные условия)

Для выполнения case для нескольких условий используйте запятую

```
let value = 8
switch value {
    case 0:
        print(0)
    case 1, 3, 5, 7, 9:
        print("Нечетное в диапазонеот 0 до 10")
    case 2, 4, 6, 8, 10:
        print("Четное в диапазонеот 0 до 10")
    default:
        print("Не входит в диапазон от 0 до 10")
}
```

Guard

Выражение guard предназначено для раннего выхода из скоупа

```
var value: Int?

guard let x = value else {
    print("Значение x равняется nil")
    return
}

print ("Значение x = \(x)")
```

Выражениями для передачи контроля

Операторы изменяющие порядок выполнения кода

- continue
- break
- fallthrough
- return
- throw

Передача управления (continue)

Останавливает выполнение итерации
и запускает выполнение следующей

```
let numbers = [1, 3, 4, 5, 6, 8, 9]

for number in numbers {
    if (number == 4) {
        print("Четыре на 2 умножать не будем")
        continue
    }
    print(number * 2)
}
```

Передача управления (break)

Break предназначен для ранней остановки
выполнения кода внутри switch
или операторов цикла

```
let numbers = [1, 3, 4, 5, 6, 8, 9]

for number in numbers {
    if (number == 4) {
        print("Снова 4, пора прекращать
        с умножением")
        break
    }
    print(number * 2)
}
```

Передача управления (break)

Break предназначен для ранней остановки выполнения кода внутри switch или операторов цикла

```
let color: UIColor = .white

switch color {
case .red:
    print("Красный")
case .blue:
    print("Синий")
case .green:
    print("Зеленый")
default:
    break
}
```

Передача управления (fallthrough)

В Swift, при выполнении условия в выражении switch, происходит прекращение выполнения кода в выражении.

```
var value = 4

switch value {
case 1...5:
    value = value * 2
    // выход после выполнения
    // этого условия
case 6...10:
     value = value + 1
default:
     value = value * 0
}
print(value) // value = 8
```

```
var value = 4

switch value {
case 1...5:
    value = value * 2
    fallthrough
    // падаем в следующий  case
case 6...10:
    value = value + 1
    // теперь значение 9
    fallthrough
    // падаем в следующий case
default:
     value =  value * 0
}
print(value) // value  = 0
```

Передача управления (fallthrough)

Нельзя использовать fallthrough
с ассоциированными значениями.
Подробная информация о них приведена в
лекции Перечисления

```
enum Result {
    case success(message: String)
    case failure(errorMessage: String)
}

let result: Result = .failure(errorMessage: "Error")

switch result {
case .success(let message):
    fallthrough // Ошибка!
case .failure(let errorMessage):
    print(errorMessage)
}
```

Передача управления (return)

Возвращает результат выполнения функции/метода

```
func sum(_ lhs: Int, _ rhs: Int ) -> Int {
    return lhs + rhs
}
```