SWIFT 5.2

Beginners guide

LEARN THE BASICS AND CORE VALUES OF PROGRAMMING IN SWIFT

Index

Hello there!

This book covers basic concepts of programming in Swift .

Hope you learn and achieve your goal.

Have a great day :)

1. Variables and Constants

Before we start to learn about swift variables and constants.

To print a statement:-

```
print(“Hello World”)
```

To write the comments

// -> for single line comments.

/* */ -> for multi line comments.

NOTE-> THE EXPLANATION IS DONE IN THE WAY, TO GET FAMILIAR WITH COMMENTS SYNTAX IN SWIFT

Variables

```
// Here "var" keyword is variable and is a mutable object.

var cardName = "Ace of Spades”

//here string value is declared in double quotes “ “ .

print(cardName)

//Output -> Ace of Spades
```

```
//Giving the variable another value
cardName = "3 of diamonds"

print(cardName)

//output -> 3 of diamonds

//Note -> variables can only be changed with data of same type , that is Integer, String, Boolean, etc.
```

Constants

```
// “let" keyword is used to declare constants and are immutable

let number  = "One"

// if we appended the constant (number  = “Two”)
//we will get an error , since its a let constant.

//Explicitly declaring variables with their types

var languageName: String = "Swift"

let monthNumber: Int = 7  //since its May :)

//if we assign a String to a type int , we will get an error for eg: month = "JavaScript"
```

```
//Adding string and other datatypes (which are not String) in printing a statement
//It's called "String Interpolation" in swift

let name = "yourName"


print("Hi, my name is \(name) and we are learning \(languageName) in month \(monthNumber)")

//Output -> Hi, my name is yourName and we are learning swift in month 7
```

2. Conditional Statements.

- *if - else*

```
var Celsius = 10

// Syntax of a simple if-else statement.
/* to compare the values or variables we use operators ,
Some of them are given below */

" < " less than operator
" > " greater than operator
" >= " greater than or equals to
" <= " less than or equals to
" == " equals to
" && " AND operator
" || " OR operator
" ! " NOT operator
" += " adds and equals to operator

 For example
var a = 3
a += 3  //which is basically  [a = a + 3]
print(a)  //Output will be 6 because [a = 3 + 3 ]

" -= " subtract and equals to
" *= "  multiply and equals to
" /= " divide and equals to


if Celsius < 0 {

  print("it's Snowing")

} else {
```

```
    print("it's cold out there")
}   // Output -> “its cold out there”
```

- ***else - if***

```
var age : Int = 8
if age < 5 {
    print("Go to preschool”)
} else if age == 5 {
        print("Go to kindergarten”)
} else if (age > 5) && (age <= 17 ) {
//use ( ) to be more specific about the condition
    let grade: Int = age - 4
    print("Go to grade \(grade)”)
} else {
    print("Go to college”)
} //Output - “Go to grade 4”
```

- *Nested- if*

```
let  num1 = 1 , num2 = 2, num3 = 3

// use comma  to declare constants or variables simultaneously..
if n1 >=  n2 {

        if n1 >= n3 {

                print("Largest number is ,\( n1)")

        } else {

                print("Largest number is ,\( n3)")

        }

} else {

        if n2 >= n3 {

                print("Largest number is ,\( n2)")

        } else {

                print("Largest number is ,\(n3)")

        }

}
```

//Output -> Largest number is 3.

3 . Switch Statements

//Every switch statement is made of “case” which executes the required condition and has a “default case” . .
//Switch statements can be of any Datatype. (which is Int, String, Boolean, etc.)
//Here is an example of switch statement

```
let ingredient = “pasta”  //example

switch ingredient {

        case "tomatoes", "pasta":

                print(“Spaghetti")

        case "beans":

                print(“Burrito")

        case "potatoes":

                print(“Mashed  potatoes”)

        default:

                print(“water")

}
```

//Output -> Spaghetti, because the (case "tomatoes", “pasta”) qualifies the condition and executes the code in it.

//Another switch statement using range (…)

```
let testScore: Int = 89

switch testScore{

        case 93...100:

                print("You got an A”)

        case 85...92:

                print("You got a B”)

        case 77...84:

                print("You got C”)

        case 69...76:

                 print("You got D”)

        default:

                  print("You got F”)

}
```

//Output -> You got a B4. Control flow statements

1. For loop

//Loops in general , iterate to the given range or any other collective datatype we provide to it.
//here a4 is an array . (Detail explanation about arrays in next topic)

```
var a4 = [1, 2, 3]


/* for loop syntax. Here “item” is the simple variable
Which iterates though the loop . Its called loop variable*/

for item in a4 {

   print(item)

} //output -> 1, 2, 3

//for loop in range
for i in 1...5 {

  print(i)

}

   //output -> 1, 2, 3, 4, 5
// Note : - here dots in 1…5 shows the range (1, 2, 3, 4, 5)
```

2. For - where loop

```
for i in 1...10 where i % 2 == 0 {

        //suitable for single conditions
          print("Even: \(i)")

}

// Output ->  2, 4, 6, 8, 10.
```

3. While loop

```
var number : Int = 1

while  number < 10 {

          if number % 2 == 0 {

                    number += 1

                    continue

            }

           if number  == 7 {

                    break

         }
```

```
    print(number)

        number += 1

}
```

//Output -> 1, 3, 5

//NOTE:-
//here “continue” will pause the execution of that particular value where its declared

//whereas the “break” will terminate the execution of that particular loop

5. Arrays, Dictionaries and Sets

1. Arrays

```
//creating an empty array

var numberOfContinents = [Int]()


//here number of continents is an empty array of type int
//OR
//we can also declare an array like this :-

var numberOfContinents2: [Int] = []

//here the array name "numberOfContinents2" is an empty
array of type Int

for continent in 1...7 {

   numberOfContinents.append(continent)

}

print(numberOfContinents)

   //Output -> [1, 2, 3, 4, 5, 6, 7]

// Note -> the three dots between 1 and 7 describes the
range (i.e. 1, 2, 3, 4, 5, 6, 7)
```

// Modifying an array

```
var nameofContinents = ["Asia", "NorthAmerica", "Europe", "Australia", "SouthAmerica" ,"Pluto", “Africa"]
```

//now array is declared .But Pluto isnt a continent 😬

//So the index (Position) of Pluto in array is 5

```
nameofContinents[5] = “Antartica"
```

//now we modified an array, lets print it out!

```
print(nameofContinents) //output ->  ["Asia", "Europe", "NorthAmerica", "Australia", "SouthAmerica" ,"Antartica", “Africa"]
```

//removing items in array

```
nameofContinents.remove(at: 5)
```

```
print(nameofContinents)
```

//Output -> antartica is removed :D

//adding custom continent in the array, lets say at position 2..

```
nameofContinents.insert("Pangea", at: 2)
```

```
print(nameofContinents)
```

//Output -> ["Asia", "Europe", "Pangea", "Australia", "SouthAmerica" ,"Antartica", “Africa"]

• ***Some additional methods to modify Array in swift.***

.remove() OR **.remove(at: indexOfAnArray)** to remove entities

.reversed() for reversing the array

.isEmpty() to check its empty or not , returns a boolean value

.first() to get the first value of an array

.append() to add an entity at the end of the array

.count() to check the amount of elements in an array

Use **(+= operator)** between two arrays to append from one array to another

2. Dictionaries

//each dictionary consists a value associated with its key

```
var courseBooks: [String: Int] = [:]

courseBooks["economicBooks"] = 2
```

//here economicBooks is the key of value 2.

//Modifying number of economic books to 4

```
courseBooks["economicBooks"] = 4
```

```
courseBooks = ["economic": 5, "computerSci": 9, "math":  4]
```

//here economic, compSci, math , each of them are key

//Now , printing the elements in the dictionary..

```
for (book, quantity) in courseBooks {

   print("I have \(quantity) \(book) books")

}
```

```
//Printing the dictionary will not print in the specific order which it's declared
```

//Thus they are not sorted and iterate random in case of dictionaries
//or accessing a single data from dictionary, we declare the key name

```
print(courseBooks["economic"]!)
```

//note-> the explanation mark is the optional unwrapping

3. Sets

```
   //to  declare a set , set the array variable to the type  “Set”

let emptyIntset : Set = [1, 2, 3, 4, 5, 6, 7]

print(emptyIntset)

//Output -> 6, 3, 2, 5, 7, 1, 4


//declaring set with duplicate values
var someset:Set = ["ab", "bc", "cd", "ef", “bc"]

print(someset)

//Output -> “bc”, “ab”, “ef”, “cd”

//Note -> the set does not print its value ordered/declared.

//access elements of the set

for val in someset {

   print(val)

}


//finding value in a set
if let value = someset.remove("cd") {
```

```
    print(value)
    print(someset)
} else {
    print("cannot find a value")
}
someset.insert("yz")
print(someset)

let a: Set = [1, 3, 5, 7 ,9]
let b: Set = [2, 4, 6, 8, 10, 7]
print(a.union(b))
print(a.intersection(b))
print(a.subtracting(b))
print(a.symmetricDifference(b))
    //7 will be removed
```

Similarly we can also use the below methods in the set in the previous Set

.isSubset(of:)

//checks if the set is subset of another set or not.

.isDisjoint(with:)

//checks if the set is subset of another set or not.

.isEmpty ()

//checks if the set is empty or not.

.first()

//returns first element of the set.

.insert(_:)

//inserts element in the set.

.reversed()

//reverse the set.

.count()

//returns the amount of elements in the set.

.removeFirst()

//removes first element in the swift.

6. Functions

// Use func keyword to initialise a function

- ***Function declaration with _ (underScore)***

```
let age = 18

func bio(_ age: Int) {

   print("Hi my age is \(age)")

}

bio(age)
```

//To explain functions, they are divided by their types.

Type 1 -> Takes nothing and returns nothing (Void)

```
func printSomething() {

   print("Hello")

}

printSomething()
```

Type2 -> Takes something and returns nothing

```
let time: Int = 7

func timeName(currentTime: Int)  -> Void {

   print("I am learning swift at \(currentTime)'o clock")

}

timeName(currentTime: time)
```

Type3 -> Takes nothing and returns a value

```
func developerType() -> String {

   let a = "I am"

   let b = “ learning Swift."

   let c = a + b

   return c

}

let printThis = developerType()

print(printThis) //Output -> “I am learning Swift”
```

Type4 -> Takes something and returns something

```
let numberofmonths = 6

func goal(month: Int) -> String {

   return "It takes \(month) months at least to be an iOS Developer"

}

print(goal(month: numberofmonths))

//Output -> “It takes 6 months at least to be an iOS developer
```

//Here the arrow " -> " represents what type of value a function is returning

Type5 -> takes multiple values and returns something

```
let num1 = 1

let num2 = 2

func addition(no1: Int , _ no2: Int) -> Int {

   let no3 = no1 + no2

   return no3
```

```
}

print(addition(no1: 4, 5)) //output 9

print(addition(no1: num1, num2)) //output 3
```

Type6 -> takes nothing and returns 3 values

```
func returnStuff() -> (String, Int, Bool) {

    let a :String = "Hi"

    let b :Int = 1

    let c : Bool = true

    return (a, b, c)

}

let printStuff = returnStuff()

print(printStuff)

//output "Hi, 1, true”
```

- **Nested functions**

//Nested functions are basically function inside function.

Nested function with no parameter and does not return value

```
func outer(){
    print("Content of outer function ")
    func inner(){
        print("Content of inner function")
    }
    inner()
}
outer()
```

//Output -> Content of outer function

//Output -> Content of inner function

Nested function with parameters and return values

```
func operate(with symbol: String) -> (Int, Int) -> Int {
    func add(num1: Int, num2: Int) -> Int {
        return num1 + num2
    }
    func sub(num1: Int, num2: Int) -> Int {
        return num1 - num2
    }
            let operation = (symbol == "+") ? add : sub
          //Here " ? " Is called the Ternary operator. If the above condition is true ,"add" will get executed ,else "sub" will be executed
                return operation
}
```

```swift
let operation = operate(with: "+")

let result = operation(2, 3)

print(result)
```

//Output -> 5

//there are many types we can make a function that returns "n" number of stuff and takes "n" number of stuff

7. Structs and Classes

//structs are generally used to make an object..
//Example -> Jake wants to buy a laptop from the store ..
//first of all
//Initialising the struct named Laptop and adding the properties it has (which is companyName and amountOfRa

```
struct Laptop {

    var companyName: String

    var amountOfRam: Int

}
```

//Class syntax
//Making the class name Store In order to show , the laptop is available in which store

```
class Store {

    var nameOfStore: String

    var laptopType: [Laptop]
```

//here we are taking an array of struct Laptop because there are more than one laptops available in store 😁

//Now, to inherit the contents the struct we use init() to initialise the contents of the struct..

```
    init(nameOfStore: String, laptopType: [Laptop]) {
        self.nameOfStore = nameOfStore
        self.laptopType = laptopType
            }
//here we used self because we want the class properties of its own.
        //we can also add function in the class.
        func storeArea() -> String {
        return "The store area is 144,000 square meters"
            }
}
let laptop1 = Laptop(companyName: "DellXPS", amountOfRam: 16)
let laptop2 = Laptop(companyName: "AppleMac", amountOfRam: 8)
let store =  Store(nameOfStore: "Walmart", laptopType: [laptop1, laptop2])
```

```
print("The \(store.nameOfStore) has \
(laptop1.companyName) with \(laptop1.amountOfRam)
Gigs of RAM and \(laptop2.companyName) with \
(laptop2.amountOfRam) Gigs of RAM laptops available
and \(store.storeArea())")
```

//Output -> The Walmart has DellXPS with 16 Gigs of RAM and AppleMac with 8 Gigs of RAM laptops available and The store area is 144,000 square meters

- ***Difference between classes and structs is struct can't inherit , while class can.***

//while in application development structs have faster performance than class when there's a lots of code. Apple heavily uses structs in SwiftUI..

//lets see the practical difference between classes and structs..

//making an object laptop3 which is a struct because we are assigning a struct (laptop2)

```
var laptop3 = laptop2

laptop3.companyName = "HP"

print(laptop2.companyName) //Output -> Apple

print(laptop3.companyName) //Output -> HP
```

```
//Same for the class but here store2 is a class because we are assigning it with a class (store)

var store2 = store

store2.nameOfStore = "Amazon"

print(store.nameOfStore) //Output -> Amazon

print(store2.nameOfStore) //Output -> Amazon
```

//here structs passes the value to the variable which is assigned that's why output was HP not APPLE
//while classes share the reference when assigned with a variable and thus both store and store2 name changes

- **Class with inheritance and overriding of the functions.**

```
class Introduction {

    func intro() {

        print("Hi my name is ABC")

    }

}
```

```
    //Making the class SecondIntro which inherits the class Introduction
    //Introduction class is called main-class while SecondIntro is a sub-class
class SecondIntro: Introduction {

    override func intro() {

        print("Hi, my name is XYZ")

    }

}

//Output -> Hi, my name is XYZ
//because , when the class SecondIntro is called,  the function intro() gets updated and prints the statement in the class SecondIntro
```

8. Optionals

To declare optionals we use (?, !) with variable or constant data type.

```
var optionalInt: Int? = 1

optionalInt = nil


//optional binding, a safer way to deal with optionals and to avoid crash in programs which use optionals.

if let optionalCheck = optionalInt {

    print("Value is \(optionalCheck)")

} //Output -> Value is nil.


/* optionals with classes
 and optional chaining */

class Person {

                let first: String

                let last : String

//initialising the class.

                        init(first: String, last: String){
```

```
                self.first = first

                self.last = last

    }

        func greetings() {

                print("hello my name is \(first) \(last)")

        }

}
```

//creating an optional .to check whether the person class exists or a nil value is to be returned.

```
var personoptional:Person?

personoptional?.greetings()

personoptional = Person(first: "eddy", last: "schmidt")

personoptional?.greetings() //Output -> “hello my name is eddy schmidt”
```

//if there is a statement which returns value then it will be an optional too
//unwrapping it

```
print((personoptional?.first)!)
```

```
//Output -> eddy
```

//! is used for explicitly unwrapping optional values . Only do it when you are sure

//if optional is nil and we don't know it , we use the nil coalesce operator
print((personoptional?.first ?? 0))

- Taking the example of struct we made in previous topic.

```
if let collegeBook = courseBooks["SocialScience"] {

    print("I have \(collegeBook) available")

}
```

//Output -> the print statement will not get executed because the courseBooks dictionary does not have a SocialScience book available

9. Enums

Enums are used to gather particular case for an instance

```
enum Place {

   case mountain , hill, street

}
```

//to access the elements/case of the enum and assigning them to the variable

```
let atMountain = Place.mountain

let atHill: Place = .hill

let atStreet = Place.street

print(atMountain, atHill, atStreet)
```

//Output –> Mountain, Hill, Street . Just printing the name of the case.

//to print the cases in the enum we have to use switch statements but to simply it more we use switch statements inside the function..

```
func printWhereWeAre(place: Place) {
    switch place {
        case .mountain:
            print("We are at mountain")
    case .hill:
            print("We are at the hill")
    default:
            print("Home sweet Home")
    }
}

printWhereWeAre(place: .hill)
    //Output -> We are at the hill
printWhereWeAre(place: .street)

//Output -> Home sweet Home
```

10. Protocols

```
//protocols give the specific functionality to the struct
//structs can inherit protocols but not a subclass
//while classes too can inherit protocol and a subclass too

//lets take the example of birds

protocol canFly {

    //calling a function in the protocol to provide the method in it.

  func fly()

}

extension canFly {

    //adding the default content in the feature, the below print statement gets executed unless some another stuff is modified in it.
  func fly() {

    print("The object takes off into the air")

  }

}
```

```
//initiating a class and making a method

class Bird {

   func layEgg() {

      print("A bird makes a new bird in shell")

   }

}


//lets take the bird “Eagle" for an instance which inherits the mainClass Bird (since its a Bird :) and a feature/protocol “canFly"

class Eagle: Bird, canFly {

            func soar() {

           print("Eagle flies in the air using air currents”)

            }

            func fly() {

              print("Eagle flies")
```

```
        }

}

// Similarly for Penguin, but it cannot fly so we are not
inheriting the protocol (canFly)

class myPenguin: Bird {

    func swim() {

        print("Penguins can swim")

    }

}

let myEagle = Eagle()

myEagle.fly()

    //Output -> Eagle flies


let mypenguin = myPenguin()

mypenguin.swim()

    //Output -> Penguins can swim
```

//Here we can see, to differentiate the bird Eagle with penguin , we used protocol….

/* Basically in swift protocols are used to run the require set of functions in the required place, to that particular objects and thus it helps in maintaining the transparency of the code…….. */

And the beginners course ends here. I hope u learnt a lot about Swift programming fundamentals !!

Thank you very much...

Book by..
*@the_shadow_dev_*
on Instagram .