# Swift

## Swift

* UIKit is a framework that encompasses all the basic stuff such as string and so on.

## Operators

* || = or
* && = and
* % = modulus

## Types of datatypes

* Int: 1,2,3
* Float, Double: 3.14, 1.168
* Decimal: 1.11
* Bool: true and false
* String: “Apple”
* Struct: Class: we can create our own data type
* In all of swift, the following shorthand is used:
  + **Array<String> = [String] // Array of String**
  + **Array<Int> = [Int] // Array of Integer**
  + **[Array<String>] = [[String]] // 2D array of String**
  + You can use any of the above but the right one is used mostly because its shorter.

## Declare variable in Swift

* **var software = "XCode"**
* **var version = 1.2**
* **var name:String = "Vishisht"**
* **var age:Int = 22**
* If the initial value of a variable is not provided then we provide the type
  + **var explicitDouble: Double**
* Conversion of types:
  + let label = "The width is "
  + let width = 94
  + let widthLabel = label + String(width)

## Declare constant in Swift

* **let language = "Swift"**

## Strings

* Declare a String: **car courseName = "swift"**
* String Concatenation: let sentence = **"Hello" + "World"**
* String Interpolation:
  + **var courseName = "iOS Development"**
  + **var language = 3**
  + **let fact = "\(courseName) in \(language)" // This will output iOS Development in 3**
* Check is a string is empty: **stringName.isEmpty //false or true**
* Change to uppercase or lowercase: **stringName.lowercaseString, stringName.uppercaseString**
* Compare two strings:
  + **let mySchool1 = "Harvard"**
  + **let mySchool2 = "harvard"**
  + **let yourSchool = "Stanford"**
  + **mySchool1 == yourSchool // false**
  + **mySchool1 == mySchool2 // false**
* Multiple line String is done using three double quotation
  + **let quotation = """**
  + **I said "I have \(apples) apples."**
  + **And then I said "I have \(apples + oranges) pieces of fruit."**
  + **"""**

## Arrays

* You cant have array that has 1 integer, 1 float, 1 object. All of them have to be int.
* Declare Array:
  + **var apps = ["Reminders", "Mail", "XCode"] // mutable**
  + **var apps:[String] = ["Reminders", "Mail", "XCode"] // mutable**
  + **let numbers = [3,4,5,100,1000] // immutable**
  + **let numbers:[Int] = [3,4,5,100,1000] // immutable**
  + **var groups = [["Duc", "Tran"], ["Vishisht", "Tiwari"]]**
* Declare Empty Array:
  + **var apps = Array<String>() // This one easier to understand in the code**
  + **var apps = [String]() // However, this is the preferable one**
* Access elements in array:
  + **var apps = ["Reminders", "Mail", "XCode"]**
  + **let reminder = apps[0]**
  + **let lastElement = apps[3]**
* Number of elements: arrayName.count
* Check if an array is empty: arrayName.isEmpty
* Mutable vs Immutable Array:
  + **Mutable: var apps = ["Reminders", "Mail", "XCode"] // mutable**
  + **Immutable: let apps = ["Reminders", "Mail", "XCode"] // immutable**
* Mutable Array:
  + Append: **arrayName.append("Safari")**
  + Remove an element at Index: **arrayName.removeAtIndex(1)**
  + Remove the last element: **arrayName.removeLast()**
  + Remove all elements: **arrayName.removeAll()**
  + Add element to an array: **arrayName.insert(1000, atIndex:3)**

## 2D Arrays

* Declare Array:
  + **var image:[[Int]] = [[3, 7, 10],[6,4,2], [8,5,2]]**
* Iterating through 2D array:
  + **for row in image {**
  + **for col in row {**
  + **print(col)**
  + **}**
  + **}**
* Access elements in array:
  + **var pixel = image[row][col]**

## Dictionary

* Dictionary is an unordered list and don’t have the same order as you created. Same Hashtable.
* Declare Dictionary:
  + **var apps = ["Productivity":"KeyNote", "Social":"Facebook", "Music":"Itunes"]**
  + **var apps:[String:String] = ["Productivity":"KeyNote", "Social":"Facebook", "Music":"Itunes"]**
* Declare Empty Dictionary:
  + **var emptyDictionary = [String:Int]()**
  + **var emptyDictionary = Dictionary<String, Int>()**
* Access elements in dictionary:
  + **let optinalApp = apps["Productivity"] // This will return KeyNote**
  + **let optinalApp = apps["Developer"] // *This will return nil. Hence original App will be optional string***
* Add key:value:
  + **apps["Developer"] = "XCode"**
* Change value:
  + **apps["Social"] = "Twitter"**
* Delete key:value:
  + **apps["Social"] = nil**

## Optionals

* **var str:String = nil // This is not allowed in swift.**
* So when there is a string, you can always be sure that it contains a string.
* If you want to create a string that can contain nil then you will have to create an optional.
* **var str:String? = nil // This is allowed**
* The '?' means that it can be nil also.
* **let optinalApp = apps["Developer"] // If dictionary app does not contain something called Developer then this will return nil. Hence optionalApp is an optional because it can have nil.**
* Sometimes when you access the variable toy have to unwrap it. This is done using !.
* ! forces the optional to be a string.
* **String!.characters.count**
* How to use if in optionals
  + **let nickName: String? = nil**
  + **let fullName: String = "John Appleseed"**
  + **let informalGreeting = "Hi \(nickName ?? fullName)" // If nickName is nil then fullName**

## Tuples

* Declare Tuples:
  + **let person = ("Billy", "Johnson")**
  + **let person = ("Billy", "Bob", "Johnson")**
  + **let person = (firstName: "Billy", secondName: "Bob", thirdName: "Johnson")**
* Access elements in tuple:
  + **person.0, person.1**
  + **person.firstName, person.secondName**
* They are used in functions for returning
  + **func divide(x: Int, y: Int) -> (Int, Int) {**
  + **let quotient = x/y**
  + **let remainder = x%y**
  + **return (quotient, remainder)**
  + **}**
  + **let result = divide(x:7, y:2)**
  + **result.0**

## Control Flow

* If Loop
  + - **if (number > 10) {**
    - **print("Its more than 10")**
    - **}**
    - **else if (number > 5) {**
    - **print("Its more than 5")**
    - **}**
    - **else {**
    - **print("I dont even know")**
    - **}**
  + How to use optionals in if loop
    - **if (let name = optionalName) {**
    - **greeting = "Hello, \name"**
    - **}**
    - If optionalName is optional then it will skip this if loop
  + Another way
    - **let nickName: String? = nil**
    - **let fullName: String = "John Appleseed"**
    - **let informalGreeting = "Hi \(nickName ?? fullName)" // If nickName is nil then fullName**
* Switch Case
  + Its necessary that a default is there
  + **switch number {**
  + **case 6...10:**
  + **print("Its between 6 and 10 including 6 and 10")**
  + **case 5..<7:**
  + **print("Its between 5 and 7 not including 7")**
  + **case 4>..7:**
  + **print("Its between 4 and 7 not including 4")**
  + **case 3>.<7:**
  + **print("Its between 3 and 7 not including 3 or 7")**
  + **default:**
  + **print("I dont even know")**
  + **}**
* While Loop
  + **while (number > 10) {**
  + **number -= 1**
  + **}**
* Repeat While Loop
  + **repeat {**
  + **number -= 1**
  + **} while (number > 10)**
* For Loop
  + Simple For Loop
    - **for (var number = 0; number < 10; number++) {**
    - **print(number)**
    - **}**
  + Fast Enumeration
    - **for number in 0..<10 {**
    - **print(number)**
    - **}**
  + Fast Enumeration with array
    - **for number in [2,5,1,9,6] {**
    - **print(number)**
    - **}**
    - **var animals = ["Cow", "Rabbit", "Dog"]**
    - **for animal in animals {**
    - **print(animal)**
    - **}**
  + 2D array
    - **var image:[[Int]] = [[3, 7, 10],[6,4,2], [8,5,2]]**
    - **for row in image {**
    - **for col in row {**
    - **print(col)**
    - **}**
    - **}**
    - **The following accesses the index and not the value**
    - **for row in 0..<image.count {**
    - **for col in 0..<image[row].count {**
    - **print(image[row][col])**
    - **}**
    - **}**

## Struct

* Structs unlike Classes are always copied
* **struct Card {**
* **var rank: Rank**
* **var Suit: Suit**
* **func SimpleDescription() -> String {**
* **return "The \(rank.simpleDescription()) of \(suit.simpleDescription())"**
* **}**
* **}**
* **let threeOfSpades = Card(rank: .three, suit: .spades)**
* **let threeOfSpadesDescription = threeOfSpades.simpleDescription()**
* Mutating keyword is used with a method modifies the struct
  + **struct Card {**
  + **var rank: Rank**
  + **var Suit: Suit**
  + **func SimpleDescription() -> String {**
  + **Suit = .spades**
  + **}**
  + **}**

## Enum

* Like classes, enums can have methods associated with them
* By default, Swift assigns the raw values starting at zero and incrementing by one each time but you can change this behaviors by explicitly specifying values as shown below.
  + **enum Rank: Int {**
  + **case ace = 1**
  + **case two, three, four, five, six, seven, eight, nine, ten**
  + **case jack, queen, king**
  + **func simpleDescription() -> String {**
  + **switch self {**
  + **case .ace:**
  + **return "ace"**
  + **case .jack**
  + **return "jack"**
  + **case .queen**
  + **return "queen"**
  + **case .king**
  + **return "king"**
  + **default:**
  + **return String(self.rawValue)**
  + **{**
  + **}**
  + **}**
  + **let ace = Rank.ace**
* If there is no meaningful raw value then I dont have to provide one:
  + **enum Suit {**
  + **case spades, hearts, diamonds, clubs**
  + **func simpleDescription() -> String {**
  + **switch self {**
  + **case .spades:**
  + **return "spades"**
  + **case .hearts:**
  + **return "hearts"**
  + **case .diamonds:**
  + **return "diamonds"**
  + **case .clubs:**
  + **return "clubs"**
  + **}**
  + **}**
  + **}**
  + **let hearts = Suit.hearts**
  + **let heartsDescription = hearts.simpleDescription()**

## Value Types

* **var a = 3**
* **var b = a**
* **b = 5**
* If we change b then there is no way that a will change
* **class number P**
* **var n: Int**
* **init(n: Int) {**
* **self.n = n**
* **}**
* **}**
* **var aNumber = number(n: 3)**
* **var bNumber = aNumber**
* **bNumber.n = 5**
* **bNumber.n // This will be 5**
* **aNumber.n // This will also be 5. So this also changed**
* So what we understand is that integer is value type and number(class) is reference type
* Now struct is very similar to class except that it is a value type
* **struct valueNumber {**
* **var n: Int**
* **init(n: Int) {**
* **self.n = n**
* **}**
* **}**
* **var aNumber = number(n: 3)**
* **var bNumber = aNumber // This just copies**
* **bNumber.n = 5**
* **bNumber.n // This will be 5**
* **aNumber.n // This will 3 still**
* So number is now a value type.
* Structs are copied
* So when you are using variables and struct you can always be safe that it is always copied.
* Classes are copied by reference
* Structs are copies by value

## Generics

* arrays are strongly types. But without generics we will have to create an array of char, an array of int etc.
* So arrays use generics that use angled brackets and are declared like the following
* Array<Element>

## Operator functions

* Swift supports operator overloading
* This means you can change the meaning of any operator
* Like you can change the meaning of +