day-1 { IOS - }

↳ Visual studio, & Anroid studio. ⩔ Xcode.

↳ IOS App Life Cycle.

↳ Ios App simulator (ios device).

↳ transfer data from one page.

↳ Firebase - Save data on cloud,

↳ push Notification (whats up Notification).

↳ Finger print Authentication.

* data types:

↳ var a = 10 (a store 10) ⇒ > output

var ↑ variable

NO compile
NO Running

→ many stuff is disabled.

print("my name is %d"+b)

Print("my name is \(b)") → ("my name is vipul")

↳ don't need to specify the data type.

↳

a = "Sample" → (Locked to Int) → NO reassigning is possible anymore.

a = a + 1 (→ 11)

Let PI_VALUE = 3.14 (Let cha cannot be changed)

var a:Int = 10
var b:String = "vipul"
var c: Float = 0.5
var d: Bool = true. → assigning value.

var temp.: Float (temp will be Float & i don't know the value now)

var age: Int8 (0---255)

Int8 → diff. int data type based on bit size
Int16
Int32

If you know value in advance int8, Int16

Int can be longer Int value.

var minValue = Int8.min (-128
var maxValue = Int8.max. (127

var answer = minValue + minValue

* type casting

Let numerator = 15
Let denominator = 2

print ("division is \(num / denomi)")

→ → (Float(numerator) / (Float) denom.)

Convert Int to Float before converting it]

→ // Character datatype. (single word, character)

//

Let mychar1 = "f"
let mychar2 = ";"
let mychar3 = "\U{123}" / Let

↑

[Unicode or emojis] Ctrl Commd t Space

* Concatinating String

var firstname = "Vipul"
var Lastname = "Shah"

var Fullname = firstName + lastname.

firstName. append (LastName)

if firstName == firstName {
print ("Equal!)
}

* Optional Operator.

default value

var temprature = 0.3 (valid temp.)
var temparature = -1 (invalid valid)

var age = -1 (wrong value)

Can't not show value

var tempature : Float ?

might have or might not have ☺)

print ("today \(temp)")

No value. →

Optional (a temprature) → Optional (27.21)

↳ Forced Unwrap (!) → crashed application


```
if let temp = temprature {
        Print ("today...")
}
else {
        Print(" No value")
}
```

} Conditional Unwrap.

```
var accessName : String ?
let printValue = accessName ?? "Anonymous"
print (printValue)
```

Nil-coalescing operator)

// Range operator

```
for i in 1...10 {
        print(" value is \(i)")
}
```

|→ ... (Range operator) → 10 time

```
for i in 1..<10 {
        print
}
```

→ 9-times

```
for friend in ["Vinay", "Vishal", "Vikas", "Ankit"] {
        print("\(friend)")
}
for char in "vipul".characters {
        print (char)
}
```

// while loop

```
var number = 0
repeat {
        print("value is \(num)")
        number = numb + 1
} while (num < 11)
```

do :- Reserve keyword in Swift (exception handling)
Repeat :- - with while loop.

// Switch case

```
switch flag1 {
  case 10:
       print ("Flag1 value \(flag1)")
  Case 20:
       print (" Flag2 value \(flag2)")
}
```

fallthrough → switch (case) → go to the next case
if fallthrough not used then it will stop there

* Switch allow Range operator.

```
switch flag1 {
  case 00... 10:
       print ("10")
  case 11... 20:
       print ("20");  → fallthrough (go to next case as well)
  case 30... 30:
       print ("30")
  default:
       Print
```

* Enum (datatype) -

```
enum type {
      case Friend     case other = 0
      case Family              = 1
}     case Company             = 2
var type = type.Family
print (type)  ——→ (Family)
print (type.hashvalue) → 1
```

```
switch type {
 case .Friend:
    print("Value is of type")
 default:
    Print ("default value")
 }
```

* tuples ≈ collections (multiple values).

Array ≈ one type of datatype (String, Int)
tuple ≈ multiple ~~datatype~~ type datatype

Let downloadFailed = (500, "Internal Server Error")
downloadFailed.0 → [500]
downloadFailed.1 → ["Internal Server Error"]

Let (responseCode, errorMessage) = downloadFailed
responseCode →
errorMessage → └ multiple Return

// Structure. ≈ Like Class & can have a function

```
Struct Contact {
        var firstName: String
        var tye: type
}
var contact = Contact (firstName: "Edu", type: .Comp)
Contact ("Edureca", .company) → which argu. for which variable.
```

→ contact.FirstName ("Edu")
contact.type (company)

```
Struct Town {
    var population = 5_422
    mutating func changePopulation (by: Int) {
        population += by
    }
}
var town = Town (population: 3000)
```

(keyword → mutating → change value of struct)

Storing value → & assigning by value
Assigning by Reference.

```
// Swift Func
func SyHello(){          →   sayHello() → Void
    print("Hello")
}

func getValue() → Int {
    return 30
}
func printMessage(message msg: String) {
    print("Message is \(msg)")
}
printMessage(message: "this is a simple Messag!")

func printFullMessage(name: String, surname:
                      String = "Smith"){
    Print("Full message is \(name) \(surname)")
}
printFullMessage(name: "Vipul", surname: "Shah")
Print FullMessage(name: "Vipul")
```

* tuples For multiple Return value.
  → Return by value.

```
func Calculate(number: Int) → (square: Int,
let ~~Int~~ Squre = number * number      cube: Int){
~~Int~~ Let Cube = number * number
    return (square, cube)
}
let let ans = Calculate(number: 10)
Print(ansvalue.square)
Print(ansvalue.cube)
```

* Function which take multiple argument.

```
func magic (numbers: Int...) {
        var temp = 0
        for num in numbers {
            temp += num
        }
        print ("Addition is \(temp)")
}
```

* Passing values by Reference

```
fun swapper (a: inout Int, b: inout Int) {

}

var magicvalue1 = 10
var magicvalue2 = 20
print (magicvalue1, magicvalue2)
swapper (a: &magicvalue1, b: &magicvalue)
```

→ & → Refereence of the variable.

→ & magic value.

// Collections.

```
// let friends = ["vipul", "vinay", "vishal"]
   var friends = [String]()
      friends.append ("vipul")
      friends.append ("vinay")
      friends.append ("vish")
     friends.insert ("Akshay", at: 1)
     friends.remove (at: 1)
     friends.removeAll() →
     friends.remove
```

friend += ["Anil"]

```
var marks = [String: Int]()
marks["vipul"] = 90
marks["Akshay"] = 70
print(marks)
if let mark = marks["Akshay"] {
}
for (name, mark) in marks {
    print ("\(name), \(mark).
}
```

tuple