class
Inheritance
Ram-Heap-FIFO ⇒ First in first out
Reference Type -
Reference to Heap
Mutable?
Objective-C

struct // does not work with objective-C
~~Inheritance~~ // biggest difference, recommend
RAM-Stack - FILO ⇒ First in last out
value type ⇒ copy and creating another object
Faster
Simpler
Immutable

// MacOS ⇒ commandLine tool ⇒ AdvancedSwiftProject
// MusicianClass.swift
```
import Foundation
class MusicianClass{ //reference type
    var name: String
    var age: Int
    var instrument: String

    init(nameInput: String, ageInput: Int, instrumentInput: String){ //used to create object out of this class
        self.name = nameInput
        self.age = ageInput
        self.instrument = instrumentInput
    }
    func happyBirthday(){
        self.age += 1
    }
}
```
// MusicianStruct.swift
```
import Foundation
struct MusicianStruct{ // value type
    var name: String
    var age: Int
    var instrument: String
    //no need to set up initializer ⇒ free initializer

    func happyBirthday(){
        self.age += 1 // error ⇒ left side of mutating operator isn't mutable: 'self' is immutable  ⇒ struct
    }
    mutating func happyBirthday(){ // must have the 'mutating' key word
        self.age += 1
    }
}
```
// main.swift
```
import Foundation

let classJames = MusicianClass(nameInput: "James", ageInput: "50", instrumentInput: "Guitar")
print(classJames.age) // 50
let structJames = MusicianClass(nameInput: "James", ageInput: "50", instrumentInput: "Guitar")
print(structJames.age) // 50

// immutable struct
classJames.age = 51
print(classJames.age) // 51
structJames.age = 51
print(structJames.age) // "error", have to declare ⇒ var structJames = .....
```

//Reference vs Value, copy an object and see the difference

```
let copyOfClassJames = classJames
var copyOfStructJames = structJames

print(copyOfClassJames.age) //51
print(copyOfStructJames.age) //51

copyOfClassJames.age = 52
copyOfStructJames.age = 52

print(copyOfClassJames.age) //52
print(copyOfStructJames.age) //52

print(classJames.age) //52
print(structJames.age) //51
```

//Reference Types -> Class
Copy -> same object new reference -> 1 object + 2 Reference

//value Types -> Struct
Copy -> new Objects -> 2 Objects

//Function vs. Mutating Function

```
print(classJames.age) //52
classJames.happyBirthDay()
print(classJames.age) //53

print(structJames.age) //51
structJames.happyBirthDay()
print(structJames.age) //52
```

//Tuple

```
let myTuple = (1,3)
print(myTuple.0) // 1

var myTuple2=(1,3,5)
myTuple2.2 = 10 //must be var tuple to be able to change the vale
print(myTuple2.2) //10

let myTuple3 = ("Atil", 100)
let myTuple4 = (10, [10,20,30])
print(myTuple4.1[1]) //20

let myString: String?
let predefinedTuple: (String, String)
predefinedTuple.0 = "Atil"
predefinedTuple.1 = "Sam"
print(predefinedTuple) //("Atil", "Sam")

let newTuple = (name: "James", metallica: true)
print(newTuple.name) //James
```

//Guard let VS If let

```
//Guard let -> Negative -> certain this will work
//if -> positive

let myNumber = "5"
func convertToIntegerGuard(stringInput:String) -> Int{
    guard let myInteger = Int(stringInput) else{ // what will happen if it does not work
        return 0
    }
    return myInteger
}

func convertToIntegerIf(stringInput:String) -> Int{
    if let myInteger = Int(stringInput){   // what will happen if it does work
        return myInteger
    }else{
        return 0
    }
}

//Switch Case
let myNum = 11
let myRemainder = myNum % 4 //reminder

switch myRemainder{
case 1:
    print("it's 1")        // or a..<b or a...b
↓similar ones
default:
    print("none of the above")
}

//Break points
var x = 5
print(x)
x += 1  //see the current values before this line excuted
print(x)
```