**import** UIKit

**var** greeting = "Hello, playground"

/\*func generic<T> (a: T) -> T.Type{

//print(type(of : a))

return type(of: a)

}

let b = "asd"

print(generic(a: b))\*/

/\*var name:String?

name = "sheraz"

print("name is \(name ?? "ali")")\*/

// Array //

/\*var arr : [Int] = [1,2]

arr.append(4)

arr.append(5)

arr += [6]

print(arr)

print(arr[3])

arr[0] = 8

arr.insert(10, at: 1)

print(arr)

arr.remove(at: 5)

arr.removeLast()

print(arr)

for item in arr{

print(item)

}

var arr1 = ["a","b","c","d","e"]

for (index, value) in arr1.enumerated(){

print("\(index): \(value)")

}\*/

// Set //

/\*var names : Set = ["ali","ahmad","hasan","afaq"]

names.insert("sheraz")

if names.contains("ali"){

print("have a name")

}else{

print("not")

}

print(names)

for items in names.sorted(){

print(items)

}\*/

// Dictionary //

/\*var dic = [1 : "a" ,2 : "b",3 : "c"]

print(dic.count)

//print(dic.updateValue("a", forKey: 1))

if let dictionary = dic.updateValue("b", forKey: 4){

print("value is \(dictionary)")

}

if let dictionary = dic.removeValue(forKey: 1){

print("value is \(dictionary)")

}

for (dicCode, dicValue) in dic {

print("\(dicCode) : \(dicValue)")

}\*/

// for //

/\*var number = [1,2,3,4,5,6,7,8]

for item in 1...5{

print(item)

}

for item in 1..<4{

print(item)

}\*/

// defer: call after the function execute //

// inout :

// closures: //

//let add:(Int,Int) -> Int = {(a,b) in return a+b }

//let result = add(4,6)

//print(result)

//let mul:(Int, Int) -> Int ={(a,b) in return a/b}

// property //

//Stored Properties: wo properties hoti hain jo class/structure ke andar value ko directly store karti hain.

//Computed Properties: wo properties hain jo kisi aur property ke basis par value ko calculate karti hain. create func

//Lazy Properties: wo properties hain jo pehli baar access karne par initialize hoti hain.

//Property Observers: aapko allow karte hain ke jab property ki value change ho to aap kuch actions perform kar sakein.

// willset: before change, didset: after change

// lazy: The someString property is initialized only when it is accessed for the first time. it is not acccessed seconddd time

/\*class Example {

lazy var someString: String = {

print("Initializing property")

return "Hello, Swift!"

}()

}

let obj = Example()

print("Before accessing lazy property")

print(obj.someString) // print only first time

print(obj.someString)

print(obj.someString)

print("After accessing lazy property")\*/

/\*struct TimesTable {

var multiplier: Int

subscript(index: Int) -> Int{

return multiplier \* index

}

}

let tableOfFive = TimesTable(multiplier: 5)

print(tableOfFive[3]) // subscript define in [

print(tableOfFive[10])\*/

// Optional binding using if let

/\*func checkValue(name : String?, age : Int?) -> String{

if let unwrappedName = name, let unwrappedAge = age{

return ("\(unwrappedName) and \(unwrappedAge)")

}else{

return ""

}

}

print(checkValue(name: "sheraz", age: 20))\*/

// guard let //

/\*func checkValue(name: String?, age: Int?) -> String{

guard let unwrappedName = name, let unwrappedAge = age else{

return ""

}

print("\(unwrappedName) and \(unwrappedAge)")

return ""

}

checkValue(name: "sheraz", age: 25)\*/

// 3. optional binding

//var name : String?

//print("name is \(name ?? "sheraz")")

// Error Handling //

/\*enum NetworkError:Error{

case badURL

case requestFailed

case unknown

}

func fetchData(url : String)throws -> String{

guard url == "validURL" else{

throw NetworkError.badURL

}

return "Network getch successfully"

}

do{

let result = try fetchData(url: "validURL") // try is used to call the function

print(result)

}

catch NetworkError.badURL{

print("bad request")

}

catch NetworkError.requestFailed{

print("requestFailed")

}

catch {

print("unknown request")

}\*/

// concurrency: execute multiple tasks parallel //

/\*import Foundation

func getData(name : String)async -> String{

try? await Task.sleep(for: .seconds(5))

return "Data fetched successfully \(name)"

}

Task{

let result = await getData(name: "sheraz")

print(result)

}\*/

// Actor : access actor's data one by one through actor methods , reference type //

/\*actor BankAccount{

private var balance = 1000

func deposit(amount : Int) -> Int{

balance += amount

//balance = balance+amount

return balance

}

func withdraw(amount : Int) -> Int{

if balance > amount{

balance = balance - amount

return balance

}

return balance

}

func getBalance() -> Int{

return balance

}

}

Task{

let account = BankAccount()

async let depositTask = account.deposit(amount: 200)

async let withdraw = account.withdraw(amount: 150)

let depositBalance = await depositTask

let withdrawbalance = await withdraw

let currentBalance = await account.getBalance()

print("deposit balance is \(depositBalance) , remaning is \(withdrawbalance) and balance is \(currentBalance)")

}\*/

// Macros : use that code in multiple places without writing it again and again //

/\*@macro calculateDiscount(price : Double, discount : Double) -> Double{

return price - (price \* discount)

}

let calculatePrice1 = calculateDiscount(price : 100.0 , discount : 2.2 )\*/

// Type Casting : convert one type to another, as(UpCasting) ,as?(DOWN) ,as! (ForceCasting) //

/\*class Teacher{

func teaches(){

print("teacher teaches")

}

}

class Student:Teacher{

func study(){

print("student study")

}

}

let ali = Student()

ali.study()

let ahmad : Teacher = ali // as: Upcasting

ahmad.teaches()

if let ali = ahmad as? Student{ // as? : downcasting

ali.study()

}else{

print("Downcasting Failed")

}\*/

// mutate is used to modify the value //

// Protocol : blueprint, define methods and properties that adopt same type //

/\*protocol Vehicle{

var name: String {get}

func start()

}

class Car: Vehicle{

var name: String

init(name: String){

self.name = name

}

func start(){

print("\(name) is starting")

}

}

let car = Car(name : "Corolla")

print(car.name)

car.start()\*/