//函数

func add(a:Int, b:Int) -> Int{

return a + b

}

var sum = add(a: 3, b: 4)

//无参数无返回值

func welcome(){

print("欢迎学习")

print("iOS应用开发技术")

}

welcome()

//多返回值（使用元组）Int默认是Int64

func maxMin() -> (Int,Int){

return(Int.min,Int.max)

}

maxMin()

//可以给某个参数以默认值

func add2(x:Int, increment:Int = 2) -> Int{

return x + increment

}

add2(x: 3)

add2(x: 3, increment: 5)

//参数类型可以是函数

func calculate(x:Int, y:Int, **method:(Int,Int) -> Int)** -> Int{

return method(x, y)

}

func add3(x:Int, y:Int) -> Int{

return x + y

}

//传递的函数类型要与参数中的函数类型一致

calculate(x: 5, y: 8, method: add3)

//参数标签

//func retangleArea(w width: Double, h height: Double) -> Double{

// let area = width \* height

// return area

//}

//retangleArea(w: 20, h: 30)

//如果不写参数标签，用\_代替

func retangleArea(\_ width: Double, h height: Double) ->Double{

let area = width \* height

return area

}

retangleArea(20, h: 30)

//闭包，是一种函数的简写形式，省去函数名，把参数和返回值放入花括号内

var city = ["xuzhou", "nanjing", "suzhou", "lianyungang"]

var cityRank1 = city.sorted()

func daoxu(a:String, b:String) -> Bool{

return a > b

}

var cityRank2 = city.sorted(by: daoxu)

//用闭包来改写，闭包里面不用写参数类型，可以自动推断，sorted方法敲两次回车

var cityRank3 = city.sorted { (a, b) -> Bool in

return a > b

}

//参数和返回类型可自动推断，单表达式可以忽略return关键词

var cityRank4 = city.sorted { (a, b) in

a > b

}

//可使用快捷参数，前缀$，从0开始递增

var cityRank5 = city.sorted { $0 > $1

}

cityRank5

//枚举

//给定一个名称，然后把每一种情况列举出来

enum week {

case Monday

case Tuesday

case Wednesday

case Thursday

case Friday

case Saturday

case Sunday

}

//使用：点语法

week.Monday

//与switch配合使用

var today = week.Friday

switch today {

case .Monday:

print("今天是周一")

case .Tuesday:

print("今天是周二")

case .Wednesday:

print("今天是周三")

case .Thursday:

print("今天是周四")

case .Friday:

print("今天是周五")

default:

print("今天是周末")

}

//类与对象

class 角色{

var id = ""

var money = 0

}

let Mike = 角色()

Mike.id = "迈克"

Mike.money = 1000

struct 坐标{

var x = 0, y = 0

}

enum 移动方式{

case 走

case 跑

case 骑

case 传送

}

//类型属性，属于类型固有的，实例不能调用，前面加static

class 生命体{

//static var 遗传方式:String{

class var 遗传方式:String{

return "RNA"

}

}

生命体.遗传方式

class Human: 生命体 {

override class var 遗传方式:String{

return "DNA"

}

}

Human.遗传方式

//计算属性：当前坐标

class 巫师:角色{

var 人物坐标 = 坐标()

var 人物移动方式 = 移动方式.走

var 当前坐标: 坐标{

get{

switch 人物移动方式 {

case .走:

人物坐标.x += 1

人物坐标.y += 1

case .跑:

人物坐标.x += 5

人物坐标.y += 5

case .骑:

人物坐标.x += 10

人物坐标.y += 10

case .传送:

人物坐标.x += 1000

人物坐标.y += 1000

}

return 人物坐标

}

set{

人物坐标 = newValue

}

}

}

var 巫师1 = 巫师()

巫师1.当前坐标

巫师1.人物移动方式 = .跑

巫师1.当前坐标

巫师1.人物移动方式 = .传送

巫师1.当前坐标

//set 设置器，给人物坐标设置新值

巫师1.当前坐标 = 坐标(x: 3000, y: 2000)

巫师1.人物坐标

//属性监视器

//willSet：事前响应，新值newValue

//didSet：事后响应，旧值oldValue

class 经验{

var 总经验 = 0{

willSet {

print("当前总经验是:\(newValue)!")

}

didSet {

print("增加了\(总经验-oldValue)点经验")

}

}

}

var 经验1 = 经验()

经验1.总经验 = 1000

经验1.总经验 = 3000

//实例方法

enum Monster : Int{

case sheep = 10,

cat = 50,

snake = 100

}

class Player{

var exp = 0

func idle(){

exp += 500

print("挂机成功1次！总经验为\(exp)")

}

func kill(monster: Monster,expTime: Int){

exp += (monster.rawValue \* expTime)

print("打了一只\(monster),现在的经验值是\(exp)") }

}

var Jack = Player()

Jack.idle()

Jack.kill(monster: .snake, expTime: 3)

//类型方法，属于类型本身，无需实例化就可以使用，方法前加关键字static

class Player1{

static var nick = "John"

//static func server(){

class func server(){

print("\(nick)，您在江苏电信1区")

}

}

Player1.server()

//对于继承，父类类型方法加关键字class，可被子类重写(override)

class WuhanPlayer : Player1{

override static func server(){

print(nick,"您在湖北移动2区")

}

}

WuhanPlayer.server()

//初始化

class RoomTemp{

var season: String

var temp: Int

init(season: String, temp: Int) {

self.season = season

self.temp = temp

}

//便利构造器，实现便利的初始化

convenience init(){

self.init(season: "Winter", temp: 4)

}

}

let temp1 = RoomTemp(season: "Spring", temp: 22)

temp1.season

temp1.temp

let temp2 = RoomTemp()

temp2.season

temp2.temp

//可失败构造器：针对有可能的初始化失败，返回nil

struct Animal{

let name: String

init?(name: String){

if name.isEmpty {

print("没有给动物命名")

return nil

}

self.name = name

}

}

let cat = Animal(name:"")