package hs  
*/\*\*  
 \* Kotlin 函数编写规则  
 \* fun 函数名 （参数名 ： 参数类型） ： 返回值类型{  
 \* 函数体  
 \* }  
 \*/*fun main() {  
 *println*(*plus*(3,5))  
 *println*(*plus2*(3,3))  
 *diaoyong*()  
}  
fun plus(a: Int , b :Int ) : Int {  
 return a + b  
}  
fun plus2(a: Int , b :Int ) : Int = a + b  
*/\*\*  
 作业  
 \*/*fun sayHello (name : String) : String = name  
fun checkAge (age : Int) : Boolean{  
 */\*\*  
 \* 相当于  
 \* if (age == 0){  
 return true  
 }else {  
 return false  
 }  
 \*/* return age >= 0  
}  
fun saveLog (logLevel : Int) {  
 *println*("saveLog = $logLevel")  
}  
fun diaoyong (){  
 *println*(*sayHello*("name"))  
 *println*(*checkAge*(18))  
 *println*(*saveLog*(3))  
}  
*/\*\*  
 输出  
 8  
 6  
 name  
 true  
 saveLog = 3  
 kotlin.Unit  
  
 \*/*

空

？可以为空

！！不可以为空

When表达式

package hs  
  
fun main() {  
 *Whens*(60)  
  
}  
fun Whens(score : Int){  
 */\*\*  
 \* kotlin when 表达式 比 Java switch 更方便  
 \*/* when(score) {  
 100 -> *println*("优秀")  
 60 -> *println*("及格")  
 else -> *println*("请继续加油")  
 }  
}

函数表达式

package hs  
  
*/\*\*  
 \* 函数表达式  
 \* 下列表达不一样  
 \* 意思一样  
 \*  
 \* fun add(x : Int , y : Int) : Int {  
return x + y  
}  
  
  
fun adds(x : Int , y : Int) : Int = x + y  
 \*  
 \*  
 \*var addjs:(Int , Int ) -> Int = { x,y -> x+y} 第二种写法  
 \* var addj = { x : Int , y : Int -> x + y} 简洁的写法  
 \*  
 \*  
println(i(1,2))  
 \*/*fun main() {  
 var addj = **{** x : Int , y : Int **->** x + y**}** var addjs:(Int , Int ) -> Int = **{** x,y **->** x+y**}** *println*(addj(1,2))  
 *println*(addjs(9,9))  
 *println*(*add*(3,3))  
 *println*(*adds*(3,4))  
}  
fun add(x : Int , y : Int) : Int {  
 return x + y  
}  
fun adds(x : Int , y : Int) : Int = x + y

List or map

package hs  
  
*/\*\*  
 \* list or map  
 \*/*import java.util.\*  
  
fun main() {  
 *orlist*()  
 *orliste*()  
 *orMap*()  
}  
  
*/\*\*  
 \* 创建简单 list 数组  
 \* 输出  
 \*/*fun orlist(){  
 var list = *listOf*<String>("1","2","3")  
 for (i in list){  
 *println*("数据:$i")  
 }  
}  
  
*/\*\*  
 \* 迭代 输出 下标 和 值  
 \*/*fun orliste(){  
 var list = *listOf*<String>("1","2","3")  
 for ((i,e) in list.*withIndex*()){  
 *println*("key:$i,value:$e")  
 }  
}  
fun orMap(){  
 var map = TreeMap<String,String>()  
 map["好"] = "good"  
 map["学习"] = "study"  
 map["上"] = "up"  
 *println*(map["好"])//获取数据  
}

Loop or Range

package hs  
  
*/\*\*  
 \* Loop 循环  
 \* Range 区间 1..100  
 \* until 开区间 1 until 100  
 \* step 步长  
 \*/*fun main() {  
 *forDome*()  
 *forstep*()  
}  
fun forDome(){  
 var result = 0  
 for (num in 1..100){  
 *print*("$num,")  
  
 result += num  
 }  
 *println*()  
 *println*("结果：$result")  
}  
  
*/\*\*  
 \* step 步长  
 \*/*fun forstep(){  
 var nums = 1..100  
 var result = 0  
 for (num in nums *step* 2){  
 *print*("$num,")  
  
 result += num  
 }  
 var c = nums.*reversed*()  
 for (i in c){  
 *print*("$i,")  
 }  
 var b = nums.*count*()  
 *println*()  
 *println*("结果：$result,,$b")  
}

人机交互

package hs  
  
*/\*\*  
 \* 人机交互  
 \* kotlin readLine() 和 Java scanse 类似  
 \*/*fun main() {  
*orRj*()  
}  
  
fun orRj(){  
 *println*("请输入第一个数字")  
 var a = *readLine*()  
 *println*("请输入第二个数字")  
 var b = *readLine*()  
  
 var num1 = a!!.*toInt*()  
 var num2 = b!!.*toInt*()  
  
 var add = **{**x:Int ,y :Int **->** x + y**}** *println*("结果：" + add(num1,num2))  
}

Kotlin 异常处理

try {  
 var num1 = a!!.*toInt*()  
 var num2 = b!!.*toInt*()  
 var add = **{**x:Int ,y :Int **->** x + y**}** *println*("结果：" + add(num1,num2))  
}catch (e : Exception){  
 *println*("请输入正确的类型")  
 *orRj*()  
}

递归=======================

package dg  
  
*/\*\*  
 \* 演示递归  
 \* 阶乘 5的阶层 5\*4\*3\*2\*1  
 \*/*fun main() {  
*println*(*fach*(5))  
}  
fun fach(num : Int): Int{  
 if (num == 1) {  
 return 1  
 }else{  
 return num \* *fach*(num - 1)  
 }  
  
}

第二种递归写法

fun fach(num : Int): Int{  
 return if (num == 1) {  
 1  
 }else{  
 num \* *fach*(num - 1)  
 }  
  
}

*/\*\*  
 \* 尾递归优化  
 \*/*tailrec fun ollAdd(num : Int ,result : Int): Int{  
 *println*("num:$num,result:$result")  
 return if (num == 0) {  
 1  
 }else{  
 *ollAdd*(num - 1 ,result + num)  
 }  
  
}

接口 实现 ，

继承 ：

父类 open

子类 override 重写

抽象类 abstract

接口 interface

委托

代理 by class a :b by c

Sealed class or menu 的区别

印章类 Sealed class 更在意类型

枚举 menu 更在意数据

OOP 面向对象编程

FP 函数式编程

package img  
  
import java.awt.image.BufferedImage  
import java.io.File  
import javax.imageio.ImageIO  
  
*/\*\*  
 \* 生成图片  
 \*   
 \*/*fun main() {  
 var image = BufferedImage(100, 100, BufferedImage.*TYPE\_INT\_RGB*)  
 var w = 10 ..90  
 var h = 10 ..90  
 image.*apply* **{** for (i in w ){  
 for (j in h){  
 setRGB(i,j,0xff00ff)  
 }  
 }  
 **}** ImageIO.write(image,"bmp",File("a.bmp"))  
}

行为参数化

把筛选的行为（函数） 作为参数传递到过滤器里面

简洁的代码，提高效率

高阶函数

.maxBay{ it. } 最大值

.filter{ booeal 类型 and} 特点条件查找

.map{ } 映射新集合

.count { }

.find {} 查找第一个符合条件的

.groupBy{“} 分组