Kiến thức cơ bản của kotlin

Code online tại : <https://play.kotlinlang.org/>

1. Biến và kiểu dữ liệu

a, Biến :

* Khai báo :

Cách 1 - var tên biến : kiểu dữ liệu = giá trị

Cách 2 – var tên biến = giá trị

* Có 2 kiểu biến là :

+ var : là dạng cơ bản chứa các kiểu dữ liệu như bình thường

+ val : những biến được tạo ra bởi val sau khi gán giá trị lần đầu nó sẽ không thể gán lại giá trị mới. Tương tự final trong java

b, Kiểu dữ liệu : Nhớ viết hoa chữ cái đầu chẳng hạn Int chứ không có int như java

+ Kiểu số : Double, Float, Long, Int, Short, Byte

+ Kiểu ký tự

+ Kiểu Strings

+ Kiểu Boolean

+ Kiểu Arrrays : DoubleArray, … ,BooleanArray, CharArray

c, Ép kiểu dữ liệu :

tên biến . toInt() : Ép biến sang kiểu dữ liệu int

d, Kiểu tra 1 biến có thuộc kiểu dữ liệu này hay không (trong java là inStandOf())

VD var a : Int = 10

Print (a is Int) -> true

e, In dữ liệu :

Sử dụng print hay println () để in ra, ngoài ra thay vị dùng ‘ + ‘ để nối chuỗi, ta có thể dùng

“ ${biến} “ để lấy dữ liệu . VD print (“Hello ${sv.name}”) hoặc print (“Hello $name”)

1. Null Safety : Kiểu dữ liệu null

Sử dụng 2 toán tử là ? và !!. Chú ý cách đặt vị trí : ‘ ? ‘ được đặt sau kiểu dữ liệu trả về còn ‘!!’ đặt sau giá trị gán

* Trong kotlin khi một biến đã được gán kiểu dữ liệu thì không thể gán giá trị cho nó thành null

VD : var a : String = “Tú”

a = null -> Sai

* Để cho phép gán giá trị null, ta sử dụng toán tử ‘ ? ‘

var a : String? = “Tú”

a = null

Khi đã cho phép nó gán giá trị null thông qua toán tử ? , trong quá trình tính toán dữ liệu nó dù bị null cũng sẽ ko gây crash app (chưa thử nghiệm – xem video thử nghiệm thì mới thử trong biến là đối tượng print (student?.name) )

var a : Int? = null

print (a?.plus(5)) -> kết quả là null

Tuy nhiên khi sử dụng này tương đương sử dụng object nên các thao tác phải xử lý qua toán tử dot (.) của object tương ứng. Ví dụ a?.plus(5 ) thay vì dùng a + 5

* Sử dụng !! sau giá trị khi không cho phép biến đó giữ giá trị null

VD var a : String = “Tú”!!

a = null -> lỗi do bắt buộc ko được gán null

* Kiểm tra 1 biến có null hay không. Nếu không null thì in ra :

var a : String? = "Hello"

a?.let{ a ->

print(a)

}

1. Toán tử

Cộng : a.plus(b) hoặc a + b

Trừ : a.minus(b) hoặc a – b

Nhân : a.times(b) hoặc a \* b

Chia : a.div(b) hoặc a / b

Chia lấy dư (%) : a.rem(b) hoăc a % b

So sánh : equals , compareTo

Và : a.and(b) hoặc a && b

Hoặc : a.or(b) hoặc a || b

Tăng 1 : a.inc() hoặc a++

Giam 1 : a.dec() hoặc a—

Sử dụng dấu . trong a.plus(b) sử dụng khi a có thể bị null -> a?.plus(b)

1. Các câu lệnh lặp, rẽ nhánh

a, IF-ELSE: Như java

b, WHEN : giống Swich trong java

var a :Int = 15

when(a){

15 -> print (“Hello”)

10 -> print (“No”)

}

Tuy nhiên when trong kotlin có thể bắt một loạt điều kiện chẳng hạn

15,14,13 -> print (“Hello”). Nghĩa là nếu a = 15 hoặc 14 hoặc 13 sẽ in ra Hello.

Ngoài ra có thể xét điều kiện trong 1 khoảng, chẳng hạn trong khoảng từ 13 đến 15

in 13..15-> print(“Hello”)

Nếu tất cả các điều kiện ko đúng, ta dùng thêm else tương tự default của java

when(a){

15 -> print (“Hello”)

10 -> print (“No”)

else -> {

print (“Nothing”)

}

}

c, While : như java

d, FOR

|  |  |
| --- | --- |
| Java | Kotlin |
| for(int i = 1; i <= 10; i++){  //…  } | for(i in 1..10){  //…  } |
| For(int I = 1; I <= 10; I += 2){  //…  } | For(I int 1..10 step 2){  //…  } |
| For (int I = 10; I >= 0; i--){  //…  } | For(I in 10 downTo 0){  //…  } |
| for(int i = 1; i < 10; i++){  //…  } | for(i in 1 until 10 ){  //…  } |
| for (String s : arrayList) {  //...  } | for (item: Int in ints){  //…  } |

e, **break, continue và return**

* Break : thoát khỏi vòng lặp gần nhất
* Continue : tiếp tục bước tiếp theo của vòng lặp gần nhất
* Return : Thoát khỏi function gần nhất

Để chỉ định break và continue đến một expression nào đó, sử dụng label: [@label](https://viblo.asia/u/label)

loop@ for (i in 1..100) {

for (j in 1..100) {

if (j > 2) break@loop

}

}

Mặc định, nếu không có label [*@loop*](https://viblo.asia/u/loop) ,break sẽ thoát ra khỏi vòng lặp gần nhất, vòng lặp j. Với label [*@loop*](https://viblo.asia/u/loop), break@loop sẽ thoát ra cả 2 vòng lặp.

Label với function

fun foo(){

var ints:Array<Int> = arrayOf(1,3,5,6)

ints.forEach lit@ {

if (it == 0) return @lit

print(it)

}

}

Câu lệnh return sẽ chỉ thoát ra khỏi function forEach, và sẽ làm tiếp các công việc phía sau. Nếu không có label, return sẽ thoát ra khỏi function foo().

1. Mảng – Arrray

* Mảng số : var mangso : IntArray = intArrayOf(1,2,3,4,5)

Lấy phần tử ở vị trí i : mangso.get(i)

Thay thế IntArray thành các kiểu khác ta sẽ được các mảng tương ứng

Tuy nhiên trong kotlin không hỗ trợ mảng String

* Mảng String : truyền qua list :

var mangString : List<String> = listOf()….

* ArrayList

var mangString : ArrayList<String> = ArrayList()

1. Collections (List, MutableList)

* Dạng List : Không cho phép add thêm phần tử, thông thường các hàm của list sẽ trả về 1 list mới sau khi thực hiện thay đổi

var ds : List<Int> = listOf(1,5,7,3,10)

ds = ds.sortedDescending()

for (i in 0 until ds.size ){

print("${ds[i]} \t")

}

println()

println("Lọc các số chẵn : ")

ds = ds.filter{

x -> x % 2 == 0

}

for (i in 0 until ds.size ){

print("${ds[i]} \t")

}

println()

* Dạng MutableList : Cho phép add thêm phần tử, hầu hết các hàm của nó thực hiện ngay trên list mà không trả về một list mới – chẳng hạng ds2.sort()

var ds2 : MutableList<Int> = mutableListOf(1,5,7,3)

ds2.add(15)

ds2.add(10)

ds2.sort()

for (i in 0 until ds2.size ){

print("${ds2[i]} \t")

}

1. Hàm – Function

a, Hàm thông thường

fun tên\_hàm (tham số){

// … do nothing

}

b, Suspend function

* Là một function có khả năng start, resume , pause (resume và pause có thể thực thi nhiều lần) sau đó là kết thúc
* Chưa hiểu rõ lắm nhưng để sử dụng thì thêm suspend vào trước fun

1. OOP trong Kotlin

* Tạo class :

+ Cách 1 : Như java

+ Cách 2 : class SinhVien(tham số truyền vào){

init {

//Đặt các tham số như contructor

// Hoten = hoten…

}

}

* Thuộc tính của đối tượng lúc tạo class cần gán một giá trị mặc định
* Phạm vi biến : như java
* Contructor :

+ contructor(tham số truyền vào)

+ Khác java, contructor trong java trùng tên với tên class còn kotlin thì tên nó là contructor()

* Data Class

+ Data class là final không thể kế thừa hay edit

+ Thêm từ khóa ‘data’ trước từ khóa ‘class’

+ Có tham số truyền vào , có thể ngay tên class . Tuy nhiên tham số truyền vào phải kèm theo từ khóa var hoặc val , khác với class bình thường chỉ cần

‘name : Kiểu dữ liệu’ != ‘var name : String’

+ Ví dụ : data class Phuongtien(var name : String, var mau : Int)

Tạo như vậy là xong, ko cần {} hay get/set gì cả

Data class được sử dụng khi lưu trữ 2 class có giá trị giống hệt nhau, khi đó data class sẽ mặc định 2 class đó = nhau khi so sanh chúng bằng ‘= =’

* Enum Class : Là class chứa danh sách các giá trị

+ Dạng 1 : Enum class Vehical {

Honda, Cup, Tree

}

Var vehical = Vehical.Honda

+ Dạng 2 : Với mỗi loại xe ta thêm giá xe cho nó thì làm như sau

Enum class Vehical (var price : Int){

Honda(1600000),

Cup (1500000)

}

Lấy giá trị : var vehical = Vehical.Honda.price

Android Kotlin

Một số lưu ý

* Kiểu Unit trong kotlin tương đương với void trong java, nó là kiểu trả về của khá nhiều hàm trong kotlin và ko cần khai báo kiểu trả về là unit nếu có

1. **Button**

a, Bắt sự kiện cho button

Cách 1 : button.setOnClickListener{

// do nothing…

}

Hàm này chứa một view trỏ về chính nó nếu ta đăt { it -> …} view này có tên là it. Tìm hiểu thêm về SCOPE FUNCTION (đã note trong sticky)

Hoặc ko cần đặt it-> cũng được vì nó mặc định là it :))

Cách 2 : button.setOnClickListener(object : View.OnClickListener{

Override fun onClick(v : View?){

// do nothing….

}

})

Hàm này chứa v : View? Chính là cái button đang tác động

1. Intent

* Khai báo :   
  var intent : Intent = Intent(this, SecondActivity::class.java)

startActivity(intent)

* Truyền dữ liệu :

+ Truyền string : intent.putExtra(key, value)

+ Truyền array như truyền String

+ Truyền đối tượng : Cần ép kiểu về Serializable và truyền như String

* Nhận dữ liệu :

+ Nhận String : var name : String = intent.getString Extra(key)

+ Nhận số : var number : Int = intent.getIntExtra(key, default)

Khi nhận số phải kèm theo giá trị mặc định default để nếu sai key thì giá trị nhận sẽ là default, còn đúng key thì nhận là values

+ Nhận array nhưng nhận String

+ Nhận đối tượng object : intent.getSerializableExtra(key) as object

1. AsynTask

* onPreExcute : khởi tạo
* dolnBackground : Xử lý logic, không update UI
* onPostExcute : Đợi dolnBackground thực hiện xong thì lấy dữ liệu update lên UI
* onProgressUpdate : Hiển thị liên tục dữ liệu xử lý của dolnBackground lên UI
* publishProgress : Cập nhật dữ liệu liên tục từ dolnBackground sang onProgressUpdate

3 tham số đầu vào lần lượt là :

+ Đối số truyền vào khi khởi tạo

+ Kiểu trả về trong quá trình doln thực hiện

+ Kiểu trả về cho onPostExcute khi dolnBackground hoàn thành xong