**I. Function**

* Function Scope: Trong kotlin, function có thể được khai báo ở cấp độ file, tức là ta không cần phải có 1 class để chứa nó như trong java mà khai báo thẳng luôn function đó trong file
* Trong kotlin sử dụng từ khoá “fun" để khai báo cho function

fun getSum(x : Int, y : Int) : Int{

return x+y

}

* Parameter truyền vao theo dạng ( name : type), các parameter được phân chia bởi dấu phẩy., sau sau đó là return type
* default argument:

fun getSum(x : Int = 0, y : Int = 0) : Int{

….

}

* giá trị default values sẽ được sử dụng khi parameter tương ứng bị bỏ qua. Với cách này t sẽ giảm được số lượng lớn overload method so với java
* **fun** getSum(a : Int = 10, b : Int = 10) : Int {
* return a+b
* }
* getSum(12) // a = 12, b = 10 => 22
* khi call function theo cả 2 cách positional và name arguments thì parameter của positional luôn luôn phải được đặt trước cái name arguments đầu tiên
* khi override method thì 2 method đó sẽ cùng sử dụng 1 default argument, và ko khai báo lại ở method con nữa.

open class A{

open **fun** getSum(a : Int = 10, b : Int = 10) : Int = a+b

}

class B : A {

override fun getSum(a: Int, b: Int) : Int = a+b // không đựợc khai báo default argument nữa

}

* đối với single-expression function (function chỉ có 1 biểu thức duy nhất) thì ta có thể viết rút gọn như sau:

fun getSum(x : Int = 0, y : Int = 0) : Int = x + y

* Đối với Single Expression thì có thể khai báo return type hoặc không .
* Nếu 1 function ko trả về 1 giá trị nào cả thì return type của function đó là Unit.Unit type có thể khai báo hoặc không.
* Variable number of arguments (Varargs) cho phép ta truyền vào 1 tập các parameter có type giống nhau, thong thường ta sẽ truyền varargs parameter là parameter cuối cùng

**fun** getSum(a: Int = 10, b: Int = 10, **vararg** numbers: Int): Int {  
 **var** sum = 0  
 **for** (number **in** numbers) {  
 sum += number  
 }  
 *println*(numbers.toString())  
 **return** sum  
 }

getSum(1, 2, 3, 4, 5, 6) // 18

Trong 1 function chỉ có duy nhất 1 parameter được khai báo vararg, các parameter đằng sau vararg parameter sẽ đường truyền theo kiểu name arguments.

* Member Function : là những function được khai báo trong class
* Local function: Kotlin cho phép chúng ta khai báo 1 function ở trong 1 function khác

fun getSum(x : Int, y : Int) : Int{

func validate(){

if (x!= null && y!= null ) return x+y

}

validate()

}

**II. Lambdas**

* Kotlin function có thể được lưu trữ trong 1 biến và cũng có thể truyền đi như 1 paramater.

**2.1 Higher-Order  Functions**

* higher-order function là 1 function nhận paramater truyền vào là 1 function và return 1 function khác.

fun fetchApi(callback: () -> Unit){

// do fetchApi

onSuccess(){

callback()

}

}

=> fetchApi({() -> //do some update UI })

**2.2 Lambda expression**

* ví dụ :

max(strings, { a, b -> a.length < b.length })

* Lambda expression syntax:

val sum = { x: Int, y: Int -> x + y }

* 1 Lambda Expression luôn luôn được bao quanh bởi dấu ngoặc nhọn {} .
* Trong kotlin , nếu parameter cuối cùng của function là 1 function  thì sẽ được viết như sau:

fetchApi(){ () ->

//do some update UI

}

**III. Inline function**

* + Sử dụng higher-order functions sẽ gây tốn tài nguyên trong quá trình run-time. Mỗi 1 function là 1 object với 1 số method , object đó sẽ lưu trữ các biến và cả các function khác nữa (trong trường hơp parameter truyền vào là function.). Và trong function ta tiếp tục gọi 1 function nữa, 1 function object nfunctionữa lại tiếp tuc đc gọi ra. Để tránh việc tiêu tốn bộ nhớ, ta có inline function
  + 1 function được khai báo là inline thì thay vì tạọ 1 function object, toàn bộ code của function sẽ được thêm vào tại ngay nơi mà function được gọi.

ta khai báo:

**inline fun** hehe(callback: (a: Int, b: Int) -> Int) {  
 callback(2, 3)  
 }

và call:

*hehe*() **{** a, b **->** a + b  
 **}**

compile ra java:

**int** $i$f$hehe = **false**;  
 **int** b = 3;  
 **int** a = 2;  
 **int** var7 = **false**;  
 **int** var8 = a + b;

* + Inline không chỉ tác động đến function đươc khai báo mà còn tác dụng lên toàn bộ các lambda truyền vào.
  + Tránh sử dụng inline với các function lớn , đặc biệt là các vòng lặp.

**IV.Noinline function.**

* + Noinline được sử dụng khi t muốn truyền các Noinline lambdas vào 1 inline function

**inline fun** hehe(**noinline** callback: (a: Int, b: Int) -> Int) {  
 callback(2, 3)  
 }

compile ra java:

Function2 callback$iv = (Function2)**null**.INSTANCE;  
 **int** $i$f$hehe = **false**;  
 callback$iv.invoke(2, 3);

* + Nếu 1 inline function không có 1 inline parameter nào thì complier sẽ warning