تا الان دیدیم وقتی تابع operation ای که جلسه پیش ساختیم رو میخواستیم استفاده کنیم تابع sum رو با صدا زدن نامش استفاده می کردیم ولی آیا واقعا لازمه که تابع رو حتما بسازیم و بعد نامش رو صدا بزنیم؟ چرا نتونیم تابع رو داخل بدنه تابع دیگه بسازیم؟ اینجاست که ما از عبارت های Lambda استفاده میکنیم.

کد زیر رو نگاه کنین:

fun operation**(**x: Int, y: Int,op: **(**Int,Int**)->**Int**)**: Int {  
 return op**(**x,y**)**}

fun sum**(**x: Int, y: Int**)** = x + y

fun main**(**args: Array<String>**)** {  
   
 println**(**operation**(**2, 3, ::sum**))** println**(**operation**(**2, 3, **{** x, y **->** x + y **}))**}

دو عبارت بالا یک عمل رو انجام میدن. اگه دقت کنین ما تایپ x و y رو توی بدنه مشخص نکردیم، درواقع کاتلین خودش متوجه میشه که چون داریم دو عبارت 2 و 3 رو به این تابع میدید پس باید x و y از جنس Int باشن! کاتلین خودش متوجه میشه که چه تایپی رو باید به x و y بده! مثلا اگه همین عبارت رو میخواستیم بنویسیم باید اینجوری مینوشیتم:

fun operation**(**x: Int, y: Int,op: **(**Int,Int**)->**Int**)**: Int {  
 return op**(**x,y**)**}

fun sum**(**x: Int, y: Int**)** = x + y

fun main**(**args: Array<String>**)** {  
   
 val LambdaSum: **(**Int, Int**) ->** Int = **{** x, y **->** x + y **}** println**(**operation**(**2, 3, LambdaSum**))**

}

و دقت کنین که در اینصورت باید تایپ x و y رو مشخص میکردیم!

خب حالا یک تابع High-Order دیگه بسازیم

fun unaryOperation**(**x:Int,op: **(**Int**) ->** Int**)**{  
  
}

fun main**(**args: Array<String>**)** {  
 unaryOperation**(**2, **{** x **->** x \* x **})**}

توی کاتلین به مانند زبون Groovy ، اگر عبارت Lambda تنها یک پارامتر داشته باشه، لازم نیست که ما دقیق بیان کنیم چی رو داریم صدا میزنیم، کافیه از کلیدواژه “it” استفاده کنیم:

fun unaryOperation**(**x:Int,op: **(**Int**) ->** Int**)**{  
  
}

fun main**(**args: Array<String>**)** {  
 unaryOperation**(**2, **{ it** \* **it })**}

در واقع ما توی کاتلین میتونیم تابع High-Order رو ساده تر و هم به صورت منظم تری بنویسیم. توی کاتلین اگر تابع به عنوان اخرین پارامتر به یک تابع High-Order پاس داده بشه میشه اون رو به صورت دیگه ای هم نوشت. مثلا کد بالا رو میتونیم اینجوری هم بنویسیم:

fun unaryOperation**(**x:Int,op: **(**Int**) ->** Int**)**{  
  
}

fun main**(**args: Array<String>**)** {  
 unaryOperation**(**2, **{ it** \* **it })** unaryOperation**(**2**) {** it \* **it  
 }**}

ممکنه شما بگین خب این کجا ممکنه به درد بخوره؟ خب بذارین توی یک مثال بهتون تفاوت و قشنگیه سینتکس کاتلین رو نشون بدم.

بیاین یک کلاس بسازیم به نام Database و اون کلاس هم یک تابع داشته باشه به نام commit ، لازم هم نیست بدنه تابع commit رو هم بنویسیم.

class Database {  
 fun commit**()**{  
 }  
}

حالا خارج کلاس، تابع دیگه ای بنویسیم به نام transaction که یک Database و یک سری کد رو به عنوان ورودی میگیره:

class Database {  
 fun commit**()**{  
 }  
}

fun transaction**(**db: Database, code: **() ->** Unit **)**{  
 try {  
 code**()** } finally {  
 db.commit**()** }  
}

و تنها کاری که میکنه اینه که code رو روی database اجرا میکنه و در انتها اون دیتابیس رو هم commit میکنه.

حالا کافیه که اینجوری ازش استفاده کنیم:

class Database {  
 fun commit**()**{  
 }  
}

fun transaction**(**db: Database, code: **() ->** Unit **)**{  
 try {  
 code**()** } finally {  
 db.commit**()** }  
}

fun main**(**args: Array<String>**)** {  
 val db = Database**()** transaction**(**db**){** //interact with database  
 **}**}

خیلی قشنگ اومدیم یک متغیر دیتابیس درست کردیم و به تابع transaction دادیم که این تابع میاد اون کد هایی که داخل براکت هامون نوشتیم رو اجرا میکنه و در انتها هم دیتابیس رو commit میکنه.

یکی دیگه از روش های ساختن یک تابع High-Order در کاتلین، استفاده از توابع بی نام است.

fun unaryOperation**(**x:Int,op: **(**Int**) ->** Int**)**{  
  
}

fun main**(**args: Array<String>**)** {  
 unaryOperation**(**2, fun**(**x: Int**)**: Int { return x \* x }**)**}

در واقع همونطور که از نامش مشخصه تابعی رو ساختیم که نام نداره. خب ممکنه بپرسین که کجا ممکنه این بدردمون بخوره ؟ برای شروع باید بگم جایی که لازم نباشه تایپ مقدار بازگشتی رو مشخص کنین و یا جایی که دلتون بخواد چندین نقطه داشته باشین که بر حسب شرایط یک مقدار متفاوت رو برگردونن.