کاتلین یک زبان شی گرا توصیف شده است ولی درواقع یک زبان تابع محور یا به اصطلاح Functional نیز هست به این صورت که ما با توابع به مانند کلاس ها رفتار میکنیم.البته خیلی بحث در مورد functional بودن یا نبودن زبان کاتلین هست ولی از نظر خیلیا همین که توابع رو بتونیم پاس کاری کنیم و به صورت اشیا با انها برخورد کنیم این زبون رو functional میکنه.و حالا میایم و یکم در مورد توابع صحبت میکنیم، چگونه ساخته میشن، چگونه استفاده میشن و غیره و غیره .

من یک فایل به نام Functions توی پکیج functions ساختم.میخوایم توابع مختلف رو بررسی کنیم و ببینیم چجوری ساخته میشن. اساسا به خاطر این که ما Top-Level-Functions هارو توی کاتلین داریم نیازی به این قظیه نیست که شما برای ساختن یک تابع نیاز به یک کلاس داشته باشین که درواقع اون تابعمون رو نگه داره و در واقع ما میتونیم توابع رو داخل فایل بنویسیم دقیقا مثل تابع Main که قبلا درست میکردیم.

خب حالا بیاین و کد زیر و نگاه کنین

fun hello**()**{  
 println**(**"Hello"**)**}  
  
fun main**(**args: Array<String>**)** {  
 hello**()**}

یه تابع ساختم به اسم hello و تنها کاری میکنه این یک Hello پرینت میکنه. تابع main رو هم ساختیم و تابع hello رو توش صدا کردیم.

توجه کردین که تابع hello هیچ ورودی رو نگرفت و تنها کلید واژه fun رو داشتیم که مشخص میکنه تابع میخوایم بسازیم و همچنین هیچگونه تایپی که بخوایم برگردونیمش هم مشخص نکردیم. و وقتی میخواستیم صداش کنیم تنها کاری که کردیم این بود که نام hello به علاوه پرانتز هاش صدا زدیم.همین. همینطور که گفتم این تابع return نداره، درواقع توی زبون های برنامه نویسی دیگه این یک تابع void است یا اگه بخوایم کاتلینی صحبت کنیم این تابع درواقع داره Unit رو برمیگردونه، پس اگه بخوایم میتونیم همین تابع رو به این صورت بنویسیم

fun hello**()**: Unit{  
 println**(**"Hello"**)**}

درواقع تنها کاری که کردیم اینه که return type رو مشخص کردیم.

Unit تقریبا میشه گفت همون void خودمونه ولی خب اگر به سمت تعریف Unit هدایت بشین متوجه میشین که این درواقع یک object هست که میتونیم با اشیا دیگه مقایسه کردش و درواقع من میتونم بگم "اگه این تابع داره Unit برمیگردونه یک کاری رو انجام بده" و خب این یک ویژگی خوب میتونه باشه که بعدا بدردمون بخوره.

اینجا میخوام یک نکته یکم سطح بالاتر بگم. در کاتلین ما میتونیم مقدار خروجی رو یک تایپ دیگه معرفی کنیم به نام Nothing . درواقع اگه دوباره به تعریف Nothing رجوع کنین متوجه میشین که صرفا یک کلاس خالیه.خب این جایی به دردمون میخوره که مثلا میخوایم بگین این تابع هیچ مقداری برنمیگردونه ولی میاد و یک Exception رو پرت میکنه. مثلا تابع زیر رو نگاه کنین

fun throwingException**()**: Nothing{  
 throw Exception**(**"This function throws an exception"**)**}

اینجا ما یک تابع نوشتیم که هیچ مقداری رو برنمیگردونه (Nothing) و در واقع داره یک Exception رو برمیگردونه.

خب پس اینجا متوجه شدین که اگر حتا return type رو ننویسین در واقع داریم Unit برمیگردونیم ولی اگه بخوایم تابعی بنویسیم که هیچی برنگردونه باید از Nothing استفاده کنیم.

خب حالا اگه من بخوام یک تابع برگردونم که یک مقداری رو برمیگردونه، مثلا کد زیر رو نگاه کنین

fun returnAFour**()**: Int{  
 return 4  
}  
  
fun main**(**args: Array<String>**)** {  
// hello()  
 val value = returnAFour**()**}

اومدیم یک تابع ساختیم که تایپ بازگشتیش Intه ، و داره مقدار 4 رو برمیگردونه و بعدشم اومدیم توی تابع main یک متغیر ساختیم که مقدار رو به اون پاس بده.

حالا میخوایم به تابع یک مقدار پاس بدیم

fun takingString**(**name: String**)**{  
 println**(**name**)**}  
  
fun main**(**args: Array<String>**)** {  
// hello()  
// val value = returnAFour()  
 takingString**(**"Some Value"**)**}

حالا اگه فرض کنیم تابعی داشته باشیم که مقادیری رو از ورودی بگیره و به خروجی پاس بده

fun sum**(**x:Int, y:Int**)**:Int{  
 return x+y  
}  
  
fun main**(**args: Array<String>**)** {  
// hello()  
// val value = returnAFour()  
// takingString("Some Value")  
 sum**(**2,3**)**}

البته میتونیم توابع ساده بالا رو به صورت دیگه هم نوشت، مثلا تابع sum رو به این صورت بنویسیم

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

توجه کنین که تابع بالا دقیقا عملکردی به مانند قبل خودش داره، و هیچ گونه تایپ بازگشتی مشخص نشده، کامپایلر اینجا با توجه به این که مقادیر ورودی x و y هستند تایپ مقدار خروجی رو خودش لحاظ میکنه و کارمون رو راحت تر میکنه.