Kotlin

آموزش صفر تا صد زبان برنامه نویسی کاتلین +pdf
(قسمت دوم)

آموزش صفر تا صد زبان برنامه نویسی کاتلین +pdf
(قسمت دوم)

کاتلین یک زبان ایستا که توسط برنامه نویسان مستقر در روسیه توسعه داده شده است. کاتلین این قابلیت را دارد تا بر روی ماشین مجازی جاوا اجرا شود. و همینطور می توان به زبان جاوا اسکریپت نیز آن را کامپایل نمود.
کاتلین در سال ۲۰۱۱ توسط کمپانی Jetbrains معرفی شد و اولین ورژن پایدار(Stable) آن در سال ۲۰۱۶ به نام ورژن 1.0 ارائه شد.
سرپرست تیم توسعه دهندگان کاتلین Andrey Breslav بیان داشته که کاتلین یک زبان قدرتمند و شیءگرا است که قصد دارد از جاوا بهتر عمل کند و به برنامه نویسان جاوا اجازه مهاجرت تدریجی به این زبان را بدهد. این زبان برنامه نویسی برای توسعه ی برنامه های اندروید و طراحی اپلیکیشن مورد استفاده قرار میگیرد.
در سال ۲۰۱۷ گوگل رسما این زبان را به عنوان زبان دوم برنامه نویسی برای اندروید معرفی کرد.

در این مقاله به بررسی ویژگی های زبان کاتلین می پردازیم.

واسط

در این بخش در مورد واسط در کاتلین بحث می کنیم. در کاتلین، واسط دقیقا مثل جاوا8 عمل می کند، که یعنی می توانند انجام متد و همینطور اعلام متد های انتزاعی را شامل شوند. واسط می تواند توسط یک طبقه برای استفاده از کاربرد های تعریف شده اش استفاده شود. ما در بخش

(طبقه ی درونی بدون نام) مثالی برای واسط ارائه کرده ایم. در این بخش بیشتر در موردش یاد خواهیم گرفت. از کلیدواژه ی "interface" برای تعریف یک واسط در کاتلین استفاده می شود، مانند کد های زیر.

```
interface ExampleInterface { var myVar: String // abstract property fun absMethod() // abstract method fun sayHello() = "Hello there" // method with { default implementation
```

در مثال بالا واسطی به نام "ExampleInterface" ساخته ایم و داخل آن دو خاصیت و متد انتزاعی داریم . به کارکرد با نام "sayHello()" نگاه کنید. که یک روش اجرا شده است.

در مثال زیر، ما واسط بالا را در یک طبقه اجرا می کنیم.

```
interface ExampleInterface { var myVar: Int // abstract property fun absMethod():String // abstract method fun hello() { println("Hello there, Welcome to TutorialsPoint.Com!") } } class InterfaceImp : ExampleInterface { override var myVar: Int = 25 override fun absMethod() = "Happy Learning " } fun main(args: Array<String>) { val obj = InterfaceImp() println("My Variable Value is = ${obj.myVar}") print("Calling hello(): ") obj.hello() print("Message from the Website-- ") println(obj.absMethod()) }
```

کد های بالا خروجی زیر را در مرورگر ارائه می دهند.

```
Calling hello(): Hello there, Welcome to TutorialsPoint.Com!
Message from the Website-- Happy Learning
```

همانطور که قبلا گفته شد، کاتلین توارث بیش از یکی را پشتیبانی نمی کند، اما نتیجه ای مشابه قابل دریافت است اگر بیشتر از دو واسط در یک زمان اجرا کنیم.

در مثال زیر دو واسط می سازیم و در ادامه آن ها را در یک طبقه اجرا می کنیم.

```
A { fun printMe() { println(" method of interface A") } } interface B { fun printMeToo() { println("I am another Method from interface B") } } // implements two interfaces A and B class multipleInterfaceExample: A, B fun main(args: Array<String>) { val obj = multipleInterfaceExample() obj.printMe() obj.printMeToo()
```

در مثال بالا دو واسط نمونه ساخته ایم a , b و داخل طبقه ای به نام "multipleInterfaceExample" دو واسط که قبلا گفته شده بود را اجرا کرده ایم. کد های بالا خروجی زیر را در مرورگر به نمایش خواهند گذاشت.

```
method of interface A
I am another Method from interface B
```

کنترل قابلیت رویت

در این بخش در مورد تغییر دهنده های مختلف موجود در زبان کاتلین حرف می زنیم. تغییر دهنده ی دسترسی برای توسعه دهنده برنامه موبایل در ساخت اپلیکیشن (Access modifier) برای محدود کردن استفاده از متغیر ها، متد ها و طبقه ها مورد استفاده در نرم افزار می شود. مانند دیگر زبان های برنامه نویسی مقصود گرا ، این تغییر دهنده در جاهای مختلفی مثل داخل سرپیام طبقه یا اعلام متد قابل استفاده اند. چهار تغییر دهنده در کاتلین موجود اند.

1.شخصی

طبقه ها، متد ها، و بسته ها می توانند با یک تغییر دهنده ی شخصی اعلام شوند. زمانی که هر چیزی به عنوان شخصی تلقی شود، در محدوده ی لحظه ای خود قابل دسترسی خواهد بود. برا ی مثال، یک بسته ی شخصی در محدوده ی همان فایل قابل دسترسی خواهد بود. یک طبقه یا واسط شخصی یا می تواند تنها با اعضای داده و غیره اش در دسترسی باشد.

```
private class privateExample { private val i = 1 private val doSomething() { } }
```

در مثال بالا طبقه ی "privateExample" و متغیر i هر دو در یک فایل کاتلین قابل دسترسی خواهند بود، در حالی قبلا گفته شده بود که همه ی آن ها به عنوان شخصی در بلوک اعلام، اعلام شده اند.

2.محافظت شده

محافظت شده یک تغییر دهنده ی دسترسی برای آموزش کاتلین است، که در حال حاظر برای اعلام های سطح بالا موجود نیست مثلا هر بسته ای نمی تواند محافظت شود. یک طبقه یا واسط محافظت شده تنها برای طبقه های زیرین خود قابل مشاهده است.

```
class A() { protected val i = 1 } class B : A() { fun getValue() : Int { return i } }
```

در مثال بالا، متغیر i محافظت شده اعلام شده است، در نتیجه، فقط برای طبقه های زیرینش قابل مشاهده است.

3.درونی

درونی یک تغییر دهنده جدید است که در کاتلین معرفی شده است. اگر چیزی به عنوان درونی علامت گذاری شود، آن زمینه ی مشخص داخل زمینه درونی خواهد بود. یک بسته ی درونی فقط درون ماژول قابل مشاهده است که درون آن اجرا می شود. واسط طبقه ی درونی تنها توسط سایر طبقه های داخل همان بسته یا ماژول قابل مشاهده است. در مثال زیر، چگونگی اجرای یک متد درونی را خواهیم دید.

```
class internalExample { internal val i = 1 internal fun doSomething() { } }
```

در مثال بالا، متد با نام "doSomething" مشخص شده است و متغیر به عنوان درونی معرفی شده است، در نتیجه این دو زمینه فقط داخل بسته که به آن ها مربوط می شوند قابل دسترسی اند.

4.عمومی

تغییر دهنده ی عمومی از هر جا در فضای کار پروژه قابل دسترسی است. اگر هیچ تغییر دهنده ی دسترسی ای مشخص نشده باشد، به طور پیش فرض در محدوده ی عمومی خواهد بود. در تمام مثال های قبلی، هیچ گونه تغییر دهنده را اعلام نکرده ایم، در نتیجه همه ی آن ها در محدوده ی عمومی هستند. در زیر مثالی برای فهم بیشتر این که چطور یک متد یا متغیر عمومی اعلام کنیم آورده شده است.

```
class publicExample { val i = 1 fun doSomething() { } }
```

در مثال بالا، هیچ گونه تغییر دهنده ای نام برده نشده است، در نتیجه همه ی این متد ها و متغیر ها به طور پیش فرض عمومی هستند.

پسوند

در این بخش، در مورد یک ویژگی جدید آموزش کاتلین به نام پسوند صحبت می کنیم. با استفاده از پسوند، قادر خواهیم بود کارکرد های متد را اضافه یا کم کنیم حتی بدون توارث یا تغییر دهنده. پسوند ها به شکل آماری حل می شوند. واقعا طبقه ی موجود را تغییر نمی دهند بلکه یک کارکرد قابل فراخواندن که با استفاده از دستور دات (.) قابل فراخواند است ، می سازد.

پسوند کارکرد

در پسوند کارکرد، کاتلین اجازه می دهد یک متد خارج از طبقه ی اصلی تعریف کنید. در مثال زیر، می بینیم که چطور پسوند در سطح کارکردی اجرا می شود.

```
class Alien
{ var skills : String = "null" fun printMySkills() { print(skills) } } fun main(args:
Array<String>) { var a1 = Alien() a1.skills = "JAVA" //a1.printMySkills() var a2
= Alien() a2.skills = "SQL" //a2.printMySkills() var a3 = Alien() a3.skills =
a1.addMySkills(a2) a3.printMySkills() } fun
Alien.addMySkills(a:Alien):String{ var a4 = Alien() a4.skills = this.skills + " "
+a.skills return a4.skills }
```

در مثال بالا، هیچ متدی داخل طبقه ی "Alien" به نام "addMySkills()" نداریم اما هنوز همین متد را جای دیگری خارج از این طبقه اجرا می کنیم. این جادوی پسوند است.
کد های بالا خروجی زیر را در مرورگر به نمایش می گذارند.

```
JAVA SQL
```

پسوند مقصودی

کاتلین مکانیزم دیگری را ایجاد کرده است تا کارکرد ثابت جاوا را اجرا کند. این عمل با استفاده از کلیدواژه ی "companion object" قابل انجام است. با استفاده از این مکانیزم، میتوانیم مقصودی از یک طبقه داخل یک factory method ایجاد کنیم و سپس می توانیم آن متد را با

استفاده از ارجاع نام طبقه فرا بخوانیم. در مثال زیر یک "companion object" می سازیم.

```
fun main(args: Array<String>) { println("Heyyy!!!"+A.show()) } class A { companion object { fun show():String { return("You are learning Kotlin from TutorialsPoint.com") } } }
```

کد های بالا خروجی زیر را در مرورگر نشان می دهند.

```
Heyyy!!! You are learning Kotlin from TutorialsPoint.com
```

مثال بالا مثل ثابت در جاوا است اما در زمان واقعی داریم یک مقصود به عنوان متغیر عضو همان طبقه می سازیم. این دلیل آن است که داخل خاصیت پسوند هم هست و می تواند به نوبت به عنوان یک پسوند مقصود نیز فراخوانده شود. به طور کل شما دارید مقصود همان طبقه را گسترش می دهید تا از بخشی از کاکرد های اعضا استفاده کنید.

طبقه های داده

در این بخش در مورد طبقه های داده ی زبان برنامه نویسی کاتلین حرف می زنیم. یک طبقه می تواند به عنوان یک طبقه ی داده علامت گذاری شود که که به عنوان "data" نام گذاری شده باشد. این نوع طبقه می تواند داده های اصلی را از هم جدا نگهدارد. به غیر از این هیچ کاربرد دیگری ندارد.

همه ی طبقه های داده باید یک سازنده ی اولیه داشته باشند و کل سازنده ی اولیه باید حداقل یک پارامتر داشته باشد. هر وقت که یک طبقه به عنوان داده نام گذاری شد، می توانیم از کاکرد درونی آن طبقه ی داده مثل "hashCode","()toString" و غیره استفاده کنیم. هر طبقه ی داده ای نمی تواند تغییر دهنده ای مثل انتزاعی یا باز یا درونی داشته باشد. طبقه ی داده می تونند به طبقه های دیگر گسترش یابد. در مثال زیر یک طبقه ی داده می سازیم.

```
fun main(args: Array<String>)
{ val book: Book = Book("Kotlin", "TutorialPoint.com", 5) println("Name of the Book is--"+book.name) // "Kotlin" println("Puclisher Name--"+book.publisher) // "TutorialPoint.com" println("Review of the book is--"+book.reviewScore) // 5 book.reviewScore = 7 println("Printing all the info all together--"+book.toString()) //using inbuilt function of the data class println("Example of the hashCode function--"+book.hashCode()) }
data class Book(val name: String, val publisher: String, var reviewScore: Int)
```

کد های بالا خروجی زیر را در مرورگر به نمایش می گذارند.، جایی که یک طبقه ی داده ساخته ایم تا تعدادی از داده ها را نگهداری کند و از کارکرد اصلی به همه اعضای داده اش دسترسی پیدا کرده ایم.

Name of the Book is--"Kotlin"
"Puclisher Name--"TutorialPoint.com
Review of the book is--5
Printing all the info all together--(name-Kotlin, publisher-TutorialPoint.com, (reviewScore-7
Example of the hashCode function---1753517245

طبقه بسته شده

در این بخش، در مورد نوع طبقه ی دیگری به نام طبقه بسته شده می آموزیم. این نوع از طبقه برای نشان دادن سلسله مراتب طبقه ی محدود شده استفاده می شود. بسته شده به توسعه دهنده این امکان را می دهد که نوع داده ی یک نوع از پیش مشخص را نگهداری کنند. برای ساخت یک طبقه ی بسته شده، باید از کلیدواژه ی "sealed" به عنوان اجرا کننده ی آن طبقه استفاده کنیم. یک کلاس بسته شده می تواند زیر طبقه ها ی خود را داشته باشد اما تمام آن زیر طبقه ها باید داخل همان فایل کاتلین با طبقه ی بسته شده اعلام شوند. در مثال زیر چگونگی استفاده از طبقه ی بسته شده را میبینیم.

```
sealed class MyExample
{ class OP1 : MyExample() // MyExmaple class can be of two types only class
OP2 : MyExample() } fun main(args: Array<String>) { val obj: MyExample =
MyExample.OP2() val output = when (obj) { // defining the object of the class
depending on the inuputs is MyExample.OP1 -> "Option One has been
chosen" is MyExample.OP2 -> "option Two has been chosen" } println(output)
}
```

در مثال بالا، یک طبقه ی بسته شده به نام "MyExample" داریم، که می تواند فقط دو نوع باشد: یکی "OP1" و دیگری "OP2". در طبقه ی اصلی مقصودی را در طبقه ی مان ایجاد می کنیم و نوعش را روی رانتایم قرار می دهیم. حالا، "MyExample" که بسته شده است، می توانیم از عبارت "when " در رانتایم استفاده کنیم تا خروجی بدست آید.
در طبقه ی بسته شده، احتاجی به استفاده از هیچ دستور "else" غیر ضروری نیستتا کد را پیچیده تر کند. کد های بالا خروجی زیر را در مرورگر نشان می دهند.

option Two has been chosen

کلیات

مثل جاوا ، آموزش کاتلین تایپ متغیر سطح بالاتری را به نام کلیات(Generics.) داراست. در این بخش، در مورد این که چطور کاتلین کلیات را اجرا می کند و این که چطور به عنوان یک توسعه دهنده می توانیم از کاربرد های داخل کتابخانه ی کلیات استفاده کنیم.

از لحاظ اجرا، کلیات بسیار شبیه به جاواست اما توسعه دهنده ی کاتلین دو کلیدواژه ی جدید "out" و "in" را معرفی کرده است تا کد های کاتلین را قابل خواندن تر و برای توسعه دهنده راحت تر کند.

در کاتلین طبقه و نوع مفهوم های کاملا متفاوتی اند. مثلا، لیست یک طبقه در کاتلین است در حالی که لیست <String> یک نوع است. مثال زیر نحوه ی اجرا ی کلیات در کاتلین را نشان می دهد.

```
fun main(args: Array<String>)
{ val integer: Int = 1 val number: Number = integer print(number) }
```

در کد های بالا ما یک "integer" (عدد صحیح) وارد کرده ایم و سپس آن متغیر را به یک متغیر عددی منتصب کرده ایم. این امکان پذیر است زیرا "Int" زیر طبقه ی طبقه ی عددی است در نتیجه تبدیل نوع به طور خودکار در رانتایم اتفاق می افتد و خروجی "1" را تولید می کند.

بیشتر در مورد کلیات در کاتلین بدانیم. بهتر است هر گاه مطمئن نیستیم از نوع داده ای که می خواهیم در نرم افزار استفاده کنیم بهتر است به سراغ داده ی کلیات برویم. به طور عمومی، در کاتلین، کلیات با <T> تعریف شده است که T مخفف template (قالب) است. که به طور دستی توسط برنامه ی مترجم کاتلین قابل اجراست. در مثال زیر می بینیم که چطور از انواع داده ی کلی در زبان برنامه نیوسی کاتلین استفاده کنیم.

```
fun main(args: Array<String>)
{ var objet = genericsExample<String>("JAVA") var objet1 = genericsExample<Int>(10) } class genericsExample<T>(input:T) { init { println("I am getting called with the value "+input) } }
```

در کد های بالا، یک طبقه با نوع بازگشت کلی ساخته ایم، که به وسیله ی <T> نشان داده می شود. به متد اصلی نگاه کنید. مقدارش را به طور دستی در اجرا با اثبات نوع مقدار تعریف کرده ایم، در حالی که مقصود این طبقه را می ساختیم. این گونه کلیات در آموزش کاتلین ساخته می شود. خروجی زیر را به محض وارد کردن کد در محل کد نویسی در مرورگر خواهیم دید.

```
I am getting called with the value JAVA
I am getting called with the value 10
```

وقتی می خواهیم نوع کلی را به یکی از انواع فوقش منتصب کنیم، باید از کلیدواژه ی "out" استفاده کنیم و وقتی می خواهیم نوع کلی را به یکی زا زیر نوع هایش منتصب کنیم از کلیدواژه ی "in" استفاده می کنیم. در مثال زیر ما از کلیدواژه ی "out" استفاده کرده ایم. به همین شکل شما می توانید از کلیدواژه ی "in" استفاده کنید.

Live Demo

```
fun main(args: Array<String>)
```

{ var objet1 = genericsExample<Int>(10) var object2 = genericsExample<Double>(10.00) println(objet1) println(object2) } class genericsExample<out T>(input:T) { init { println("I am getting called with the value "+input) } }

کد های بالا خروجی زیر را در مرورگر نشان می دهند.

I am getting called with the value 10
I am getting called with the value 10.0
genericsExample@28d93b30
genericsExample@1b6d3586

نمایندگی (Delegation)

کاتلین از طراحی نمایندگی با معرفی کلیدواژه ی جدید by پشتیبانی می کند.با استفاده از این کلیدواژه یا متد نمایندگی، کاتلین به طبقه ی بدست آمده اجازه می دهد به تمام متد های عمومی اجرا شده ی یک واسط توسط یک مقصود مشترک دسترس یابد. مثال زیر نشان می دهد این امر در کاتلین چگونه اتفاق می افتد.

Live Demo

interface Base
{ fun printMe() //abstract method } class BaseImpl(val x: Int) : Base { override fun printMe() { println(x) } //implementation of the method } class Derived(b: Base) : Base by b // delegating the public method on the object b fun main(args: Array<String>) { val b = BaseImpl(10) Derived(b).printMe() // prints 10 :: accessing the printMe() method }

در مثال، واسط "Base" با متد انتزاعی اش به نام "printme()" را داریم. در طبقه ی BaseImpl ما این "printme()" را اجرا می کنیم و سپس از طبقه یدیگری ما از این اجرا با کلیدواژه ی "by" استفاده می کنیم.

کد های بالا خروجی زیر را در مرورگر نشان میدهند.

10

نمایندگی خاصیت

در بخش قبل در مورد طراحی نمایندگی با کلیدواژه ی "by" آموختیم. در این بخش، در مورد نمایندگی خاصیت ها با استفاده از متد های استاندارد موجود در کتابخانه ی کاتلین می آموزیم.

نمایندگی یعنی انتقال مسئولیت به طبقه یا متدی دیگر. وقتی یک خاصیت در بعضی مکان ها اعلام شده است، باید از همان کد برای اجرایشان استفاده کنیم.در مثال های زیر از یک سری

متد های نمایندگی موجود در کاتلین و کارکرد های استاندارد کتابخانه در حین اجرای نمایندگی در مثال هایمان استفاده می کنیم.

استفاده از Lazy()

است که یک خاصیت را به عنوان ورودی می گیردو در نتیجه یک lambda یک کارکرد Lazy نوع خواص استفاده شده است را می دهد. برا یفهم بیشتر به مثال <T> که Lazy<T> نمونه از زیر توجه کنید.

Live Demo

```
val myVar: String by lazy { "Hello" } fun main(args: Array<String>) { println(myVar +" My dear friend") }
```

در کد های بالا، ما متغیر "myVar" را به کارکرد Lazy انتقال می دهیم، که در نتیجه مقدار را به مقصودش متصب می کند و همان را به کارکرد اصلی بازمی گرداند. در زیر خروجی در مرورگر را می بینید.

```
Hello My dear friend
```

Delegetion.Observable()

Observable() دو استدلال را برای اجرا کردن مقصود و بازگشت به کارکرد فراخوانده شده انجام می دهد. در مثال زیر می بینیم که چطور از متد Observable() برای نمایندگی اجرا استفاده کنیم.

Live Demo

```
import kotlin.properties.Delegates
class User
{ var name: String by Delegates.observable("Welcome to Tutorialspoint.com") { prop, old, new -> println("$old -> $new") } } fun main(args: Array<String>) { val user = User() user.name = "first" user.name = "second" }
```

کد های بالا خروجی زیر را در مرورگر به نمایش می گذارند.

```
first -> second
```

به طور کل، چینش آن به شکل عبارت بعد از اجرای کلیدواژه ی by است. متد های get() و set() از متغیر p به متد های getValue()و setValue() که در کلاس نمایندگی تعریف شده اند منتقل می شوند.

```
class Example
{ var p: String by Delegate() }
```

برای کد های بالا، در ادامه طبقه ی نمایندگی ای را می بینیم که برای منتصب کردن مقدار به متغیر p به اجرای آن نیاز داریم.

```
class Delegate
{
operator fun getValue(thisRef: Any?, property: KProperty<*>): String {
return "$thisRef, thank you for delegating '${property.name}' to me!"
}
operator fun setValue(thisRef: Any?, property: KProperty<*>, value: String) {
println("$value has been assigned to '${property.name} in $thisRef.'")
}
}
```

در حین خواندن ، متد getValue() فراخوانده می شود و در حالی تنظیم متغیر، متد setValue() فراخوانده می شود.

کارکرد ها

کاتلین یک زبان برنامه نویسی ثابت است، در نتیجه کارکرد ها نقش اساسی ای در آن ایفا می کنند. با کارکرد ها آشنا هستیم، همان طور که در مثال ها مدادم از آن ها استفاده کرده ایم. کارکرد با کلیدواژه ی "fun" اعلام می شود. مانند زبان های برنامه نویسی مقصودگرا، به یک نوع بازگشتی و یک لیست استدلال گزینه ای نیز نیاز است.

در مثال زیر کارکردی به نام MyFunction را تعریف می کنیم و از کارکرد اصلی این کارکرد را فرا می خوانیم و چند استدلال را بررسی می کنیم.

```
Live Demo
fun main(args: Array<String>) {
println(MyFunction("tutorialsPoint.com"))
}
fun MyFunction(x: String): String {
var c:String = "Hey!! Welcome To ---"
return (c+x)
}
```

کد های بالا خروجی زیر را در مرورگر نشان می دهند.

```
Hey!! Welcome To ---tutorialsPoint.com
```

کارکرد باید به صورت زیر اعلام شود.

```
fun <nameOfFunction>(<argument>:<argumentType>):<ReturnType>
```

کارکرد Lambda

Lambda کارکرد سطح بالایی است که به شکل موثر کد های سطحی را کم می کند در حالی که کارکردی را اعلام می کند و همان را تعریف می کند. کاتلین به شما اجازه می دهد Lambda ی خود را تعریف کنید. در کاتلین، می تواند Lambda تان را اعلام کنید و آن Lambda را به کارکردی اختصاص دهید.

به مثال زیر توجه کنید.

.

در کد های بالا، ما Lambda ی خود را به نام "mylambda" ساخته ایم و یک متغیر به این Lambda اختصاص داده ایم، که از نوع رشته ای است و شامل مقدار"TutorialsPoint.com" می شود.

Live Demo

```
fun main(args: Array<String>) { val mylambda :(String)->Unit = {s:String->print(s)} val v:String = "TutorialsPoint.com" mylambda(v) }
```

کد های بالا خروجی زیر را در مرورگر نشان می دهند.

```
TutorialsPoint.com
```

کارکرد درون خطی

مثال بالا اصول اولیه ی عبارت lambda را نشان می دهد که می توانیم در نرم افزار کاتلین استفاده کنیم. حالا، می توانیم یک lambda را به کارکرد دیگری اختصاص دهیم که تا خروجی مان را بگیریم که کارکرد فراخواندن را یک کارکرد درون خطی م ی کند.

به مثال زیر توجه کنید.

Live Demo

```
fun main(args: Array<String>) { val mylambda:(String)->Unit = {s:String->print(s)} val v:String = "TutorialsPoint.com" myFun(v,mylambda) //passing lambda as a parameter of another function } fun myFun(a :String, action: (String)->Unit) { //passing lambda print("Heyyy!!!") action(a)// call to lambda function }
```

کد های بالا حروجی زیر را در مرورگر به ارمغان می آورند. با استفاده ی کارکرد درون خطی lambda را به عنوان یک پارامتر انتقال داده ایم. هر کارکرد دیگری با استفاده از کلیدواژه ی "inline" می تواند به کارکرد درون خطی تبدیل شود.

```
Heyyy!!!TutorialsPoint.com
```

اعلام از بین برنده

کاتلین ویژگی های زیادی از سایر زبان های برنامه نویسی دارد. این موضوع به شما این اجازه را می دهد که چند متغیر را در یک زمان اعلام کنید. به این تکنیک اعلام از بین برنده می گویند. در ادامه چینش اولیه ی اعلام از بین برنده را می بینید.

```
val (name, age) = person
```

در چینش بالا، یک مقصود ساخته ایم و تمام آن ها را در یک دستور تعریف کرده ایم. در ادامه می توانیم به شکل زیر از آن ها استفاده کینم.

```
println(name)
println(age)
```

حالا، چگونه می توانیم از همین در نرم افزار خود استفاده کنیم. به مثال زیر توجه کنید که یک طبقه ی دانش آموز ساخته ایم با چند ویژگی و در ادامه از آن ها استفاده می کنیم تا مقدار های مقصود را بدست بیاوریم.

```
Live Demo
fun main(args: Array<String>) {
val s = Student("TutorialsPoint.com","Kotlin")
val (name,subject) = s
println("You are learning "+subject+" from "+name)
}
data class Student( val a :String,val b: String ){
var name:String = a
var subject:String = b
}
```

کد های بالا خروجی زیر را در مرورگر به نمایش می گذارند.

```
You are learning Kotlin from TutorialsPoint.com
```

رسیدگی به استثناعات

رسیدگی به استثناعات بخش مهمی از یک زبان برنامه نویسی است. این تکنیک جلوی نرم افزار ما را از تولید خروجی اشتباه در رانتایم می گیرد. در این بخش، یاد می گیریم که چطور به یک استثناء رانتایم در کاتلین رسیدگی کنیم. استثناعات در کاتلین بسیار شبیه به استثناعات در جاوا است. تمام استثناعات از طبقه ی "Throwable" می آیند. مثال زیر چگونگی استفاده از تکنیک رسیدگی به استثناعات در کاتلین را نشان می دهد.

```
fun main(args: Array<String>)
```

```
{
try {
val myVar:Int = 12;
val v:String = "Tutorialspoint.com";
v.toInt();
} catch(e:Exception) {
e.printStackTrace();
} finally {
println("Exception Handeling in Kotlin");
}
}
```

در کد های بالا، یک ریشه (String) اعلام کرده ایم و سپس آن ریشه را به یک integer چسبانده ایم که در حقیقت یک استثناء رانتایم است. در نتیجه خروجی زیر را در مرورگر می بینیم.

```
val myVar:Int = 12;
Exception Handeling in Kotlin
```

نکته : مثل جاوا، کاتلین هم بلوک نهایی را بعد از اجرای بلوک catch اجرا می کند.