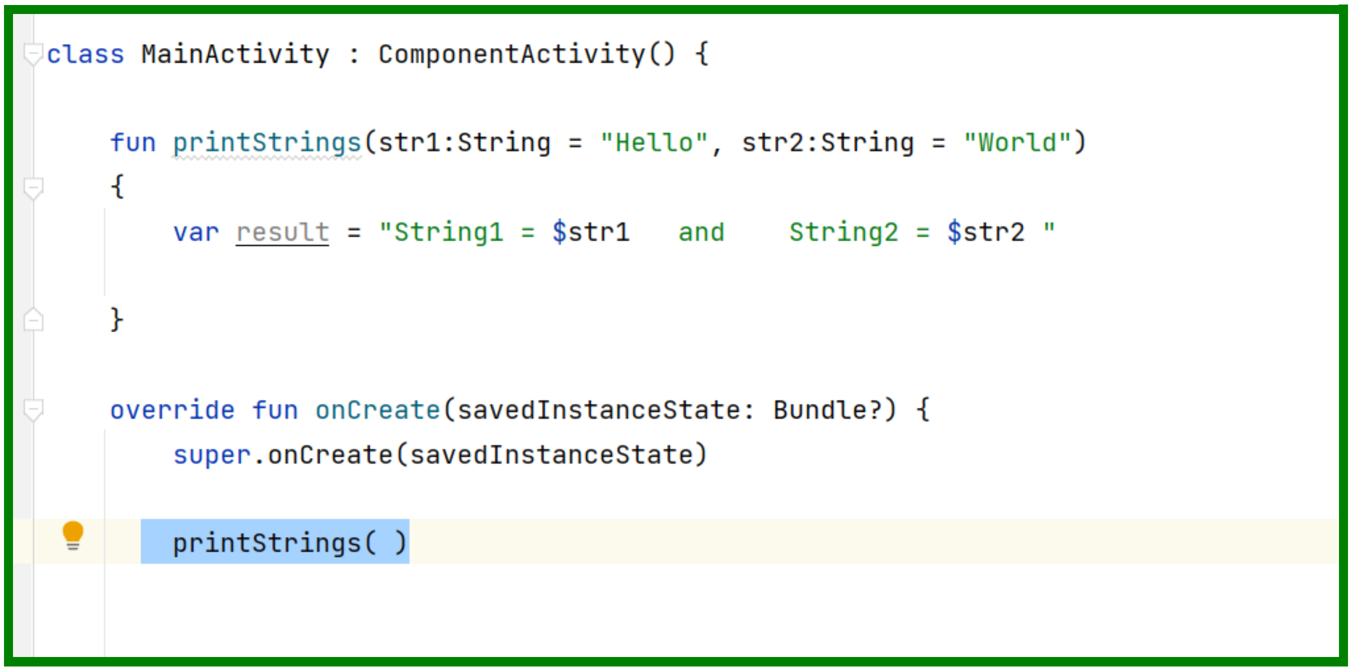
Default parameters on functions

In function parameters, you can specify default parameters, meaning that if you don't supply parameters to the function, the function will pick the default parameter instead. Have a look at this function:

```
fun printStrings(str1:String = "Hello", str2:String = "World")
{
  var result = "String1 = $str1   and    String2 = $str2 "
}
```

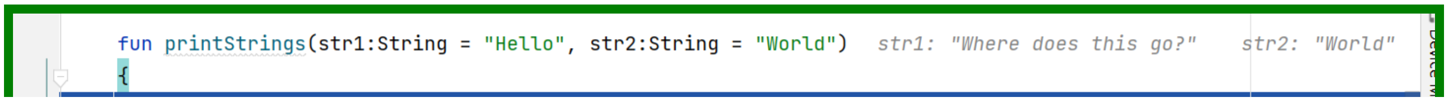
Try calling this function in the onCreate() function. Set a breakpoint and step into the function. Notice that str1 and str2 are "Hello" and "World". Since you didn't provide values for the parameters, then they take the default values.



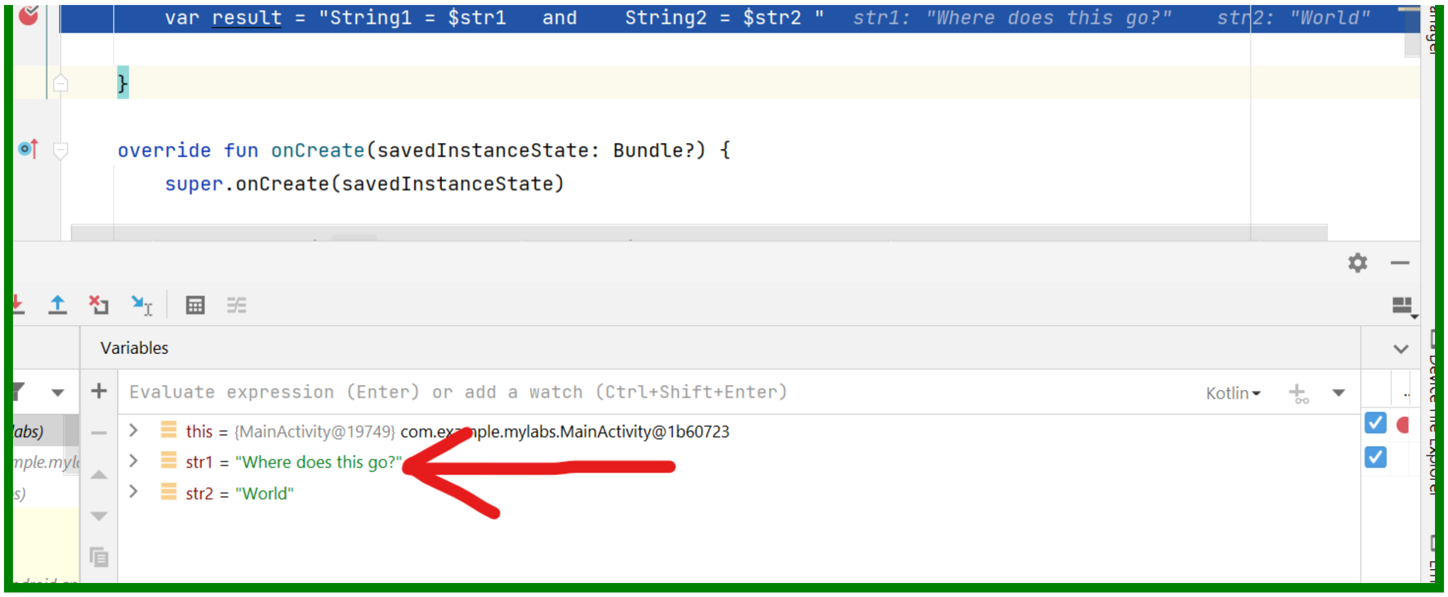
```
class MainActivity : ComponentActivity() {

    fun printStrings(str1:String = "Hello", str2:String = "World")
    {
        var result = "String1 = $str1   and    String2 = $str2 "

    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        printStrings( )
```

Now try calling the function providing only parameter:

```
  printStrings("Where does this go?")
```



```
    fun printStrings(str1:String = "Hello", str2:String = "World")  str1: "Where does this go?"   str2: "World"
    {
```



```
    var result = "String1 = $str1   and    String2 = $str2 "  str1: "Where does this go?"    str2: "World"

}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
```

Variables
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
Kotlin

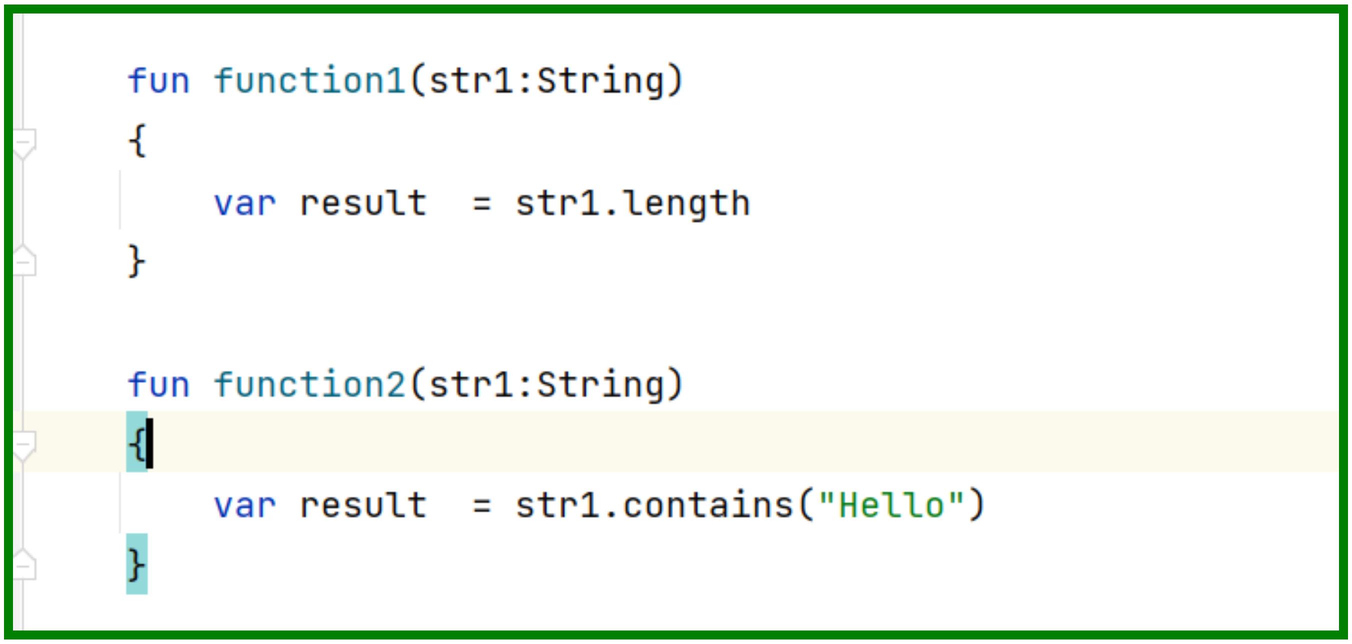
```
this = {MainActivity@19749} com.example.mylabs.MainActivity@1b60723
str1 = "Where does this go?"
str2 = "World"
```

Notice that the default parameters get assigned from left-to-right, just like C++ and other languages handle default parameters. Something that Kotlin provides is the ability to use named parameters, meaning that you can specify which value goes to which to function parameter. For instance, you can switch the order of the function parameters in the call:

```
printStrings( str2="Where does this go?", str1 = "First Parameter")
```

If you don't specify the parameter name "str1=" before the parameter, then Kotlin uses Left-To-Right assignment of parameter values.

You can also send a function as a parameter to another function. For instance, declare two functions that take a String as a parameter:



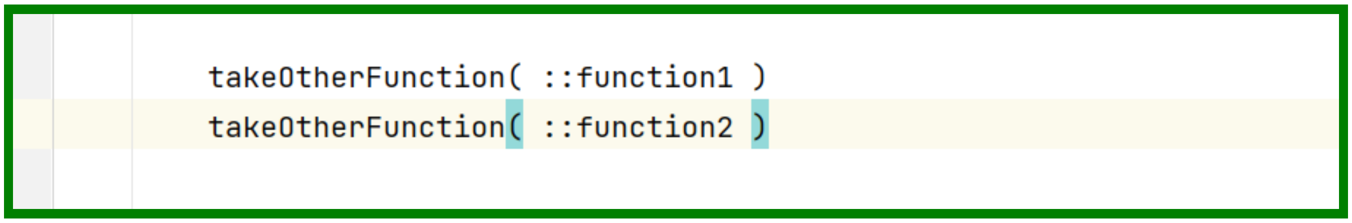
```
fun function1(str1:String)
{
    var result  = str1.length
}

fun function2(str1:String)
{
    var result  = str1.contains("Hello")
}
```

Now you can write a function that takes a function as a parameter. Just change the parameter type to be a function instead.

```
 fun takeOtherFunction( function : (String) -> Unit )
{


}
```

The syntax for a function type is (Parameter types) -> Return type. If your function takes an Int, and a String, then the parameter types would be **(Int, String)**. The arrow (->) separates the parameter types from the return type. Kotlin uses "Unit" to mean void. If your function doesn't return anything then the return type is **Unit**.
So how would you declare a parameter type that is a function that takes and **Int** and **Long**, and returns **Double**?
answer: **param: (Int, Long) -> Double**

So now try calling your function passing in function1 and function2 as parameters:



```
takeOtherFunction( ::function1 )
takeOtherFunction( ::function2 )
```

Lambda functions

Instead of declaring functions somewhere else, and passing them as parameters, you can declare the function right where they are needed as parameters. Since our takeOtherFunction() takes a function which needs a string, you can declare the function right in the parentheses:

```
takeOtherFunction(  { a:String -> var len = a.length}  )
```

You declare the function parameters inside the { } braces, and then the -> arrow separates the parameters from the function body. How would you declare a lambda function that takes an Int, and a Long, and returns a Double?
answer: **{ i:Int, l:Long -> i + l as Double}**
To convert or cast an object from one type to another, use the **as** keyword followed by the new type:

```
5 as Long
34.5 as Double
```

When you are passing a lambda function, and it is the last variable in the parameter list, then you can declare the lambda function after the parentheses. These two lines mean the same thing:

```
takeOtherFunction( { str1:String -> var len = str1.length } )
takeOtherFunction() {str1:String -> var len = str1.length }
```

You can declare the lambda function after the closing () when the last parameter is a lambda function. When the function only takes 1 parameter which is a lambda function, you can also leave out the () entirely, like this:

```
takeOtherFunction {str1:String -> var len = str1.length }
```