Functions

Functions

A function is a named, reusable block of code that can be called and executed throughout a program.

A function is declared with the `fun` keyword, a function name, parentheses containing (optional) arguments, as well as an (optional) return type.
To call/invoke a function, write the name of the function followed by parentheses.

```kotlin
fun greet() {
  println("Hey there!")
}

fun main() {
  // Function call
  greet() // Prints: Hey, there!
}
```

Function Arguments

In Kotlin, an argument is a piece of data we can pass into a function when invoking it.
To pass data into a function, the function's header must include parameters that describe the name and data type of the incoming data. If a function is declared with parameters, then data must be passed when the function is invoked. We can include as many parameters as needed.

```kotlin
fun birthday(name: String, age: Int) {
   println("Happy birthday $name! You turn $age today!")
}

fun main() {
  birthday("Oscar", 26) // Prints: Happy birthday Oscar! You turn 25 today!
  birthday("Amarah", 30) // Prints: Happy birthday Amarah! You turn 30 today!
}
```

Default Arguements

We can give arguments a default value which provides an argument an automatic value if no value is passed into the function when it's invoked.

```
fun favoriteLanguage(name, language = "Kot
lin") {
  println("Hello, $name. Your favorite pro
gramming language is $language")
}


fun main() {
  favoriteLanguage("Manon") // Prints: Hel
lo, Manon. Your favorite programming langu
age is Kotlin

  favoriteLanguage("Lee", "Java") // Print
s: Hello, Lee. Your favorite programming l
anguage is Java
}
```

Named Arguments

We can name our arguments when invoking a function to provide additional readability.
To name an argument, write the argument name followed by the assignment operator (=) and the argument value.
The argument's name must have the same name as the parameter in the function being called.
By naming our arguments, we can place arguments in any order when the function is being invoked.

```
fun findMyAge(currentYear: Int, birthYear:
 Int) {
    var myAge = currentYear - birthYear
    println("I am $myAge years old.")
}

fun main() {
  findMyAge(currentYear = 2020, birthYear
= 1995)
  // Prints: I am 25 years old.
  findMyAge(birthYear = 1920, currentYear
= 2020)
  // Prints: I am 100 years old.
}
```

Return Statement

In Kotlin, in order to return a value from a function, we must add a return statement to our function using the `return` keyword. This value is then passed to where the function was invoked.
If we plan to return a value from a function, we must declare the return type in the function header.

```
// Return type is declared outside the par
entheses
fun getArea(length: Int, width: Int): Int
{
  var area = length * width

  // return statement
  return area
}

fun main() {
  var myArea = getArea(10, 8)
  println("The area is $myArea.") // Print
s: The area is 80.
}
```

Single Expression Functions

If a function contains only a single expression, we can use a shorthand syntax to create our function.
Instead of placing curly brackets after the function header to contain the function's code block, we can use an assignment operator `=` followed by the expression being returned.

```
fun fullName(firstName: String, lastName:
String) = "$firstName $lastName"

fun main() {
  println(fullName("Ariana", "Ortega")) //
 Prints: Ariana Ortega
  println(fullName("Kai", "Gittens")) // P
rints: Kai Gittens
}
```

Function Literals

Function literals are unnamed functions that can be treated as expressions: we can assign them to variables, call them, pass them as arguments, and return them from a function as we could with any other value.
Two types of function literals are anonymous functions and lambda expressions.

```
fun main() {
  // Anonymous Function:
  var getProduct = fun(num1: Int, num2: Int): Int {
     return num1 * num2
  }
  println(getProduct(8, 3)) // Prints: 24

  // Lambda Expression
  var getDifference = { num1: Int, num2: Int -> num1 - num2 }
  println(getDifference(10, 3)) // Prints: 7
}
```