Kotlin

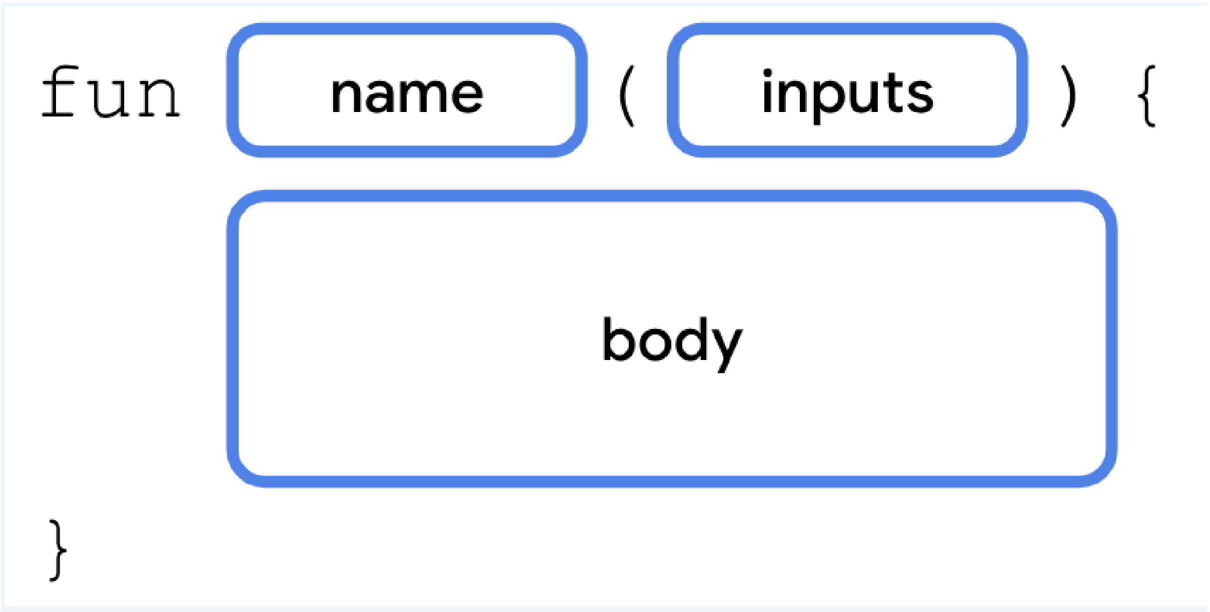
```
fun main() {
    println("Hello, world!")
}
```

- `fun main() { ... }`: This line defines the entry point of the Kotlin application. The `main` function is where the execution of the program begins. The parentheses `()` indicate that this function takes no parameters.
- `println("Hello, world!")`: Inside the `main` function, this line is a function call to `println`, which is a built-in function that prints the specified string to the standard output, followed by a newline.

Define vs call a function

In your code, you *define* a function first. That means you specify all the instructions needed to perform that task.

Once the function is defined, then you can *call* that function, so the instructions within that function can be performed or executed.



- The function definition starts with the word `fun`.
- Then the name of the function is `main`.
- There are no inputs to the function, so the parentheses are empty.

- There is one line of code in the function body, `println("Hello, world!")`, which is located between the opening and closing curly braces of the function

Unlike C, C++, ending ';' is not needed Here, as long as we are writing each statements in separate lines.

This Ex works :

```
 fun main() {
    println("Hello, Android ")
    println("Hello, Android ")
}
```

But the following Ex gives error :

```
fun main() {
    println("Cloudy") println("Partly Cloudy")
}
```

To fix this Add **;** after 1st statement

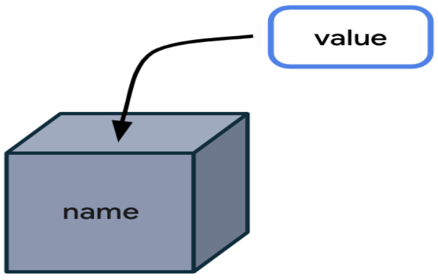
```
fun main() {
    println("Cloudy"); println("Partly Cloudy")
}
```

Here are some of the relevant style guide recommendations for what you've learned in Kotlin so far:

- Function names should be in camel case and should be verbs or verb phrases.
- Each statement should be on its own line.
- The opening curly brace should appear at the end of the line where the function begins.
- There should be a space before the opening curly brace
- The closing brace should line up with the `fun` keyword at the beginning of the function.

Variables

You don't want to write the code (or instructions) in your news app to only work for a user named Alex, or for a news article that always has the same title and publication date. Instead, you want a more flexible app, so you should write your code by referencing variable names like `name`, `article1Name`, `article1Date`, and so on



Basic Types

- Numbers
  - Byte: 8-bit signed integer
  - Short: 16-bit signed integer
  - Int: 32-bit signed integer
  - Long: 64-bit signed integer
  - Float: 32-bit floating-point number
  - Double: 64-bit floating-point number
- Characters
  - Char: represents a character. It is not treated as a number in Kotlin.
- Booleans
  - Boolean: represents a boolean value (true or false)

Advanced Types

- Arrays
  - Array: represents a fixed-size collection of elements of the same type. Arrays in Kotlin are represented by the `Array` class.
- Collections
  - List: an ordered collection with access to elements by indices. Can be mutable (`MutableList`) or immutable.
  - Set: a collection of unique elements. Can be mutable (`MutableSet`) or immutable.
  - Map: a collection of key-value pairs. Keys are unique and each key maps to exactly one value. Can be mutable (`MutableMap`) or immutable.

Special Types

- String

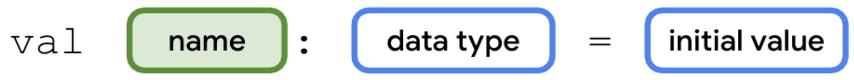
    - Represents a sequence of characters.
- Ranges
    - Provides a range of values which is often used in for-loops.
- Nullable Types
    - Any of the above types can be made nullable by adding a `?` at the end of the type name. This allows the variable to hold a null value.
- Unit
    - Corresponds to the `void` type in Java. It is used when a function does not return a value.
- Nothing
    - Represents a value that never exists. It's used for functions that never return (e.g., that always throw an exception).
- Dynamic
    - A type available in Kotlin/JS that allows you to bypass Kotlin's type checks

An *expression* is a small unit of code that evaluates to a value. An expression can be made up of variables, function calls, and more. In the following case, the *expression* is made up of one variable: the `count` variable. This expression evaluates to `2`.

expression	value
`count`	`2`

```
fun main() {
    val count: Int = 2
    println(count)
}
```

This program creates a variable called `count` with an initial value of `2` and uses it by printing out the value of the `count` variable to the output.



After the variable name, you add a colon, a space, and then the data type of the variable. As mentioned earlier, `String`, `Int`, `Double`, `Float,` and `Boolean` are some of the basic Kotlin data types.

Ex:

```
fun main() {
    val name: String = "Sachin"
    println(name)
}
```

NOTE : characters literals use **‘ ‘** and the String literals use **“ “**

String template

Use **$** to get the value of variable in the string itself

```
fun main() {
    val msgCnt: Int = 12
    println("You have $msgCnt number of messages")
}
```

Type inference

Type inference is when the Kotlin compiler can infer (or determine) what data type a variable should be, without the type being explicitly written in the code.

That means you can omit the data type in a variable declaration, if you provide an initial value for the variable.

```
val balance = 50.68

println("The balance is $balance")
```

Above code works fine

Math operations with integers

Inside the String Templates, we can use { } curly braces to evaluate complex expressions like addition,subtraction etc any other maths operation on variables.

```
val balance = 50.68
val bills = 10.3
println("The left money is ${balance-bills}")
```

Or We can assign the result of maths operation to new variable and print it.

Ex :
```
val balance = 50.68
val bills = 10.3
val left = balance - bills
 print(left)
```

Update the Variables

If you need to update the value of a variable, declare the variable with the Kotlin keyword `var`, instead of `val`.

- `val` keyword - Use when you expect the variable value will not change.
- `var` keyword - Use when you expect the variable value can change.

Ex : **val cartTotal = 0**

cartTotal = 20
println("Total: $cartTotal")
O/P : Val cannot be reassigned

Solved :
var cartTotal = 0
cartTotal = 20
println("Total: $cartTotal")
O/P : **Total: 20**

We also can't update by giving the value of another data type rather than one assigned to the variable.

var balance = 50.68
balance = 90
println(balance)

O/P : `The integer literal does not conform to the expected type Double`

Increment and decrement operators :

count = count + 1
Count ++
Count - -

Double

val trip1: Double = 3.20
val trip2 = 1.3
Var total = trip1 + trip2
println(total)
O/p : 4.5

String

```
val name = "sachin"
val lname = "Doddamani"
val fullname = name + lname
println(fullname)
```

o/p : **`sachinDoddamani`**

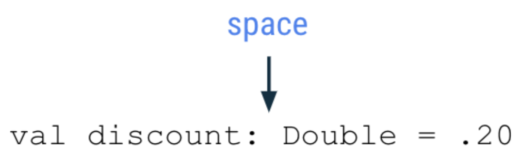
***escape sequences*. **

```
println(" Say \"hello\" ")
```

o/p : Say "hello"

Coding Conventions :

- Variable names should be in camel case and start with a lowercase letter.
- In a variable declaration, there should be a space after a colon when you specify the data type.



```
val discount: Double = .20
```

- There should be a space before and after an operator like the assignment (`=`), addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`) operators and more.

Comments :

```
// This is a comment.

/*
 * This is a very long comment that can
 * take up multiple lines.
 */
```

Define and call a function

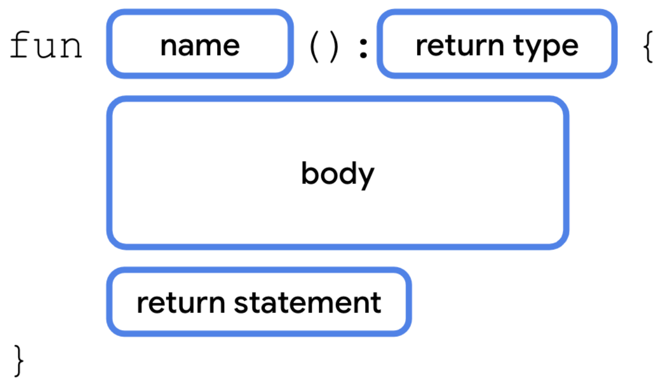
```
fun main() {
    birthdayGreeting()
}
```

```
fun birthdayGreeting() {
   println("Happy Birthday, Rover!")
   println("You are now 5 years old!")
}
```

o/p :

```
Happy Birthday, Rover!
You are now 5 years old!
```

<u>Return a value from a function</u>



The `Unit` type

By default, if you don't specify a return type, the default return type is `Unit`. **Similar to Void**

```
fun main() {

   eatCookies()

}

fun eatCookies(): Unit{

   println("hi")

}
```

Return a `String`

Ex :

```
fun birthdayGreeting(): String {

     println("Happy Birthday, Rover!")
     println("You are now 5 years old!")
   }
```

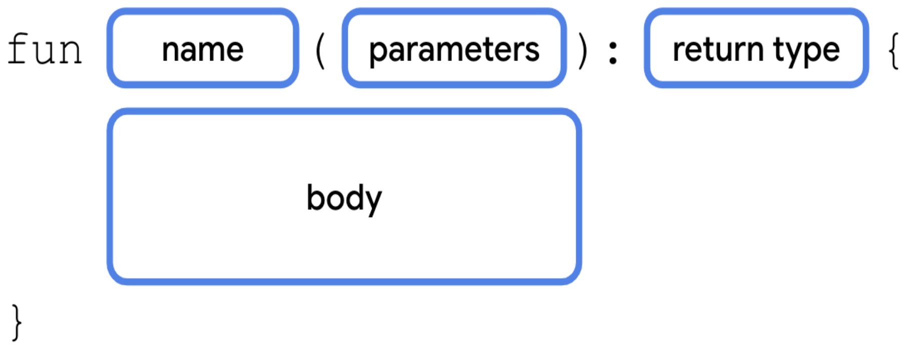
gives you Error , because you havent provided the return string

Correct :

```
fun main() {
    print(eatCookies())
}
fun eatCookies(): String{
    var name ="sachin"
    var roll = "222"
    return name+" "+roll
}
```

o/p : `sachin 222`

Function Parameters



Each parameter consists of a variable name and data type, separated by a colon and a space. Multiple parameters are separated by a comma.

```
fun main() {
    val greetings = birthdayGreet("Rohan",10)
    println(greetings)
    println(birthdayGreet("janu",12))
}
```

```
fun birthdayGreet(name : String, age : Int): String{

    return "happy birthday $name, You are $age yrs old"

}
```

O/P :

```
happy birthday Rohan, You are 10 yrs old

happy birthday janu, You are 12 yrs old
```

Named arguments

To address the possibility of passing values in any order we can pass name args.

```
println(birthdayGreet(age=11,name="sade"))
```

O/P : `happy birthday sade, You are 11 yrs old`

Default arguments

When args are not passed during calling the function, default values are assigned to those parameters if assigned.

```
fun birthdayGreet(name : String = "Jayashri", age : Int = 10): String{
    return "happy birthday $name, You are $age yrs old"
}
```

In main :

```
println(birthdayGreet(age=14))
println(birthdayGreet(name="Raju"))
println(birthdayGreet())
```

O/P :

```
happy birthday Jayashri, You are 14 yrs old
happy birthday Raju, You are 10 yrs old
happy birthday Jayashri, You are 10 yrs old
```