KOTLIN - BASICS

Package definition and imports

- Package specification should be at the top of the source file.
  - package my.demo
  - import kotlin.text.* // ...
- It is not required to match directories and packages: source files can be placed arbitrarily in the file system.

Program entry point

- An entry point of a Kotlin application is the main function.
  - **fun** main() {
    - println("**Hello world!**")

    }
- Another form of main accepts a variable number of String arguments.
  - **fun** main(args: Array<String>) {
                println(args.contentToString())

    }

Print to the standard output

- print *prints* its argument to the standard output.
- print("**Hello** ")print("**world!**")

- println prints its arguments and adds a line break, so that the next thing you print appears on the next line.
- println("**Hello world!**")
- println(42)

Functions

- A function with two Int parameters and Int return type.
  - **fun** sum(a: Int, b: Int): Int {
    - **return** a + b}
- A function body can be an expression. Its return type is inferred.
  - **fun** sum(a: Int, b: Int) = a + b
- A function that returns no meaningful value.
  - **fun** printSum(a: Int, b: Int): Unit {
    - println("**sum of $a and $b is ${a + b}**")

    }
- Unit return type can be omitted.
  - **fun** printSum(a: Int, b: Int) {
    - println("**sum of $a and $b is ${a + b}**")
    - }

Variables

- Read-only local variables are defined using the keyword val. They can be assigned a value only once.
- **val** a: Int = 1
- // immediate assignment
- **val** b = 2
- // `Int` type is inferred
- **val** c: Int
- // Type required when no initializer is provided
- // deferred assignment
- c = 3
- Variables that can be reassigned use the var keyword.
- **var** x = 5
- // `Int` type is inferred
- x += 1

- You can declare variables at the top level.

```
val PI = 3.14
var x = 0

fun incrementX() {
    x += 1
}
```

Creating classes and instances

- To define a class, use the class keyword.
- class Shape
- Properties of a class can be listed in its declaration or body.
- class Rectangle(var height: Double, var length: Double) {
  - var perimeter = (height + length) * 2

  }
- The default constructor with parameters listed in the class declaration is available automatically.
- **val** rectangle = Rectangle(5.0, 2.0)

- Inheritance between classes is declared by a colon (:). Classes are final by default; to make a class inheritable, mark it as open.
- open class Shape
- class Rectangle(var height: Double, var length: Double): Shape() { var perimeter = (height + length) * 2 }

Comments

- Just like most modern languages, Kotlin supports single-line (or *end-of-line*) and multi-line (*block*) comments.
- // This is an end-of-line comment
- /* This is a block comment on multiple lines. */
- Block comments in Kotlin can be nested.
- /* The comment starts here
  - /* contains a nested comment *   /
  - and ends here. */

String templates

- var a = 1
- // simple name in template:
- val s1 = "a is $a"

- a = 2
- // arbitrary expression in template:
- val s2 = "${s1.replace("is", "was")}, but now is $a"

Conditional expressions

```
fun maxOf(a: Int, b: Int): Int {
    if (a > b) {
        return a
    } else {
        return b
    }
}
fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

for loop

```
val items = listOf("apple", "banana", "kiwifruit")
for (item in items) {
    println(item)
}
Or
val items = listOf("apple", "banana", "kiwifruit")
for (index in items.indices) {
    println("item at $index is ${items[index]}")
}
```

while loop

```
val items = listOf("apple", "banana", "kiwifruit")
var index = 0
while (index < items.size) {
    println("item at $index is ${items[index]}")
    index++
}
```

when expression

```
fun describe(obj: Any): String =
    when (obj) {
        1          -> "One"
        "Hello"    -> "Greeting"
        is Long    -> "Long"
        !is String -> "Not a string"
        else       -> "Unknown"
    }
```

Ranges

- Check if a number is within a range using in operator.

```
val x = 10
val y = 9
if (x in 1..y+1) {
   println("fits in range")
}
```

- Check if a number is out of range.

```
     val list = listOf("a", "b", "c")
     if (-1 !in 0..list.lastIndex) {
      println("-1 is out of range")
     }if (list.size !in list.indices) {
      println("list size is out of valid list indices range, too")
}
```

- Iterate over a range.

```
for (x in 1..5) {
    print(x)
}
```

Or over a progression.

```
for (x in 1..10 step 2) {
    print(x)
}println()
for (x in 9 downTo 0 step 3) {
     print(x)}
```

Collections

- Iterate over a collection.
- **for** (item **in** items) {
  - println(item)}
- Check if a collection contains an object using in operator.
- **when** {
  - "**orange**" **in** items -> println("**juicy**")
  - "**apple**" **in** items -> println("**apple is fine too**")}

Using lambda expressions to filter and map collections

```
val fruits = listOf("banana", "avocado", "apple", "kiwifruit")
fruits
.filter { it.startsWith("a") }
.sortedBy { it }
.map { it.uppercase() }
.forEach { println(it) }
```

```
APPLE
AVACADO
```