Fresher Android

GST PTG Fresher Training

Git Flow

Basic Syntax

#1. Package definition and imports

A source file may start with a package declaration:

```
package my.demo
import kotlin.text.*
// ...
```

Classes in Kotlin are declared using the keyword **class:**

```
class Invoice { /*...*/ }
```

#2. Functions

Functions in Kotlin are declared using the **fun** keyword:

```
fun double(x: Int): Int {
    return 2 * x
}
fun message(str: String) {
    println(str)
}
```

#3. Variables

Read-only variables are defined using the keyword **val**

```
val a: Int = 1  // immediate assignment
val b = 2   // `Int` type is inferred
val c: Int  // Type required when no initializer is provided
c = 3       // deferred assignment
```

Variables that can be reassigned use the keyword **var**

```
var x = 5 // `Int` type is inferred
x += 1
```

#4. Conditional expressions

In Kotlin, **if** is an conditional expression:

```
// Traditional usage
var max = a
if (a < b) max = b

// With else
var max: Int
if (a > b) {
    max = a
} else {
    max = b
}

// As expression
val max = if (a > b) a else b
```

#5. Nullable values and null checks

A reference must be explicitly marked as nullable or null by “**?**”

Example:

```
fun parseInt(str: String): Int? {
        // ...
    }
```

The keyword **null** represent for the null value

Basic Syntax. Summary

- Package definition and imports
- Functions
- Variables
- Conditional expressions
- Nullable values and null checks
- Q&A

Basic Types

#1. Numbers – built-in types

Kotlin provides a set of built-in types that represent numbers with different sizes, hence, value ranges:

Type	Size (bits)	Min value	Max value
Byte	8	-128	127
Short	16	-32768	32768
Int	32	-2,147,483,648 (-2^{31})	2,147,483,647 ($2^{31} - 1$)
Long	64	-9,223,372,036,854,775,808 (-2^{63})	9,223,372,036,854,775,807 ($2^{63} - 1$)

All variables initialized values not exceeding the maximum value of the expect type.

```
val one = 1 // Int
val threeBillion = 3000000000 // Long
val oneLong = 1L // Long
val oneByte: Byte = 1
```

#2.Numbers – floating-point numbers

Kotlin provides a set of built-in types that represent numbers with different sizes, hence, value ranges:

Type	Size (bits)	Significant bits	Exponent bits	Decimal digits
Float	32	24	8	6-7
Double	64	53	11	15-16

For variables initialized with fractional numbers, the compiler infers the **Double** type

```
val pi = 3.14 // Double
val e = 2.7182818284 // Double
val eFloat = 2.7182818284f // Float, actual value is 2.7182817
```

#3. Numbers – literal constants

There are the following kinds of literal constants for integral values:

```
Decimals: 123
Longs are tagged by a capital L: 123L
Hexadecimal: 0x0F
Binaries: 0b00001011
```

You can use underscores to make number constants more readable:

```
val oneMillion = 1_000_000
val creditCardNumber = 1234_5678_9012_3456L
val socialSecurityNumber = 999_99_9999L
val hexBytes = 0xFF_EC_DE_5E
val bytes = 0b11010010_01101001_10010100_10010010
```

#4. Characters

Characters are represented by the type **Char**

```
fun check(c: Char) {
    if (c == 1) { // ERROR: incompatible types
        // ...
    }
}
```

Character literals go in single quotes: '1‘

The following escape sequences are supported: \t, \b, \n, \r, \', \", \\ and \$.

#5. Booleans

The type **Boolean** represents booleans, and has two values: **true** and **false**.

Built-in operations on booleans include

- || – lazy disjunction
- && – lazy conjunction
- ! - negation

#6. Arrays

Arrays in Kotlin are represented by the **Array** class.

To create an array, we can use a library function arrayOf() and pass the item values to it:

```
val arr = arrayOf(1, 2, 3)
```

or use the Array constructor that takes the array size and the function that can return the initial value:

```
// Creates an Array<String> with values ["0", "1", "4", "9", "16"]
val asc = Array(5) { i ->
    (i * i).toString()
}
asc.forEach { println(it) }
```

To call the members of array, use get/set function or the **[]** operation.

#7. Strings

Strings are represented by the type **String**.

```
val str : String = "This is a string"
```

You can concatenate strings using the + operator.

A raw string is delimited by a triple quote (""")

```
val text = """
    |Tell me and I forget.
    |Teach me and I remember.
    |Involve me and I learn.
    |(Benjamin Franklin)
    """.trimMargin()
```

#8. Operations

Kotlin supports the standard set of arithmetical operations over numbers (+ - * / %)

Division of integers always returns an integer

```
val x = 5 / 2
//println(x == 2.5) // ERROR: Operator '==' cannot be applied to 'Int' and 'Double'
println(x == 2)

val y = 5L / 2
println(y == 2L)

val z = 5 / 2.toDouble()
println(z == 2.5)
```

#9. Comparison

Equality checks:

```
a == b
a != b
```

Comparison operators:

```
a < b
a > b
a <= b
a >= b
```

Range instantiation and range checks:

```
a..b
x in a..b
x !in a..b
```

Basic Types. Summary

- Numbers – built-in types
- Numbers – floating-point numbers
- Numbers – literal constants
- Characters
- Booleans
- Arrays
- Strings
- Operations
- Comparison
- Q&A

Control Flow

#1. If Expression

The **if** statement specifies one or more statements to execute if an expression evaluates to true

```
// Traditional usage
var max = a
if (a < b) max = b
```

The **if** statement can have **else** branch:

```
// With else
var max: Int
if (a > b) {
    max = a
} else {
    max = b
}
```

The **if** can work as a expression:

```
val max = if (a > b) a else b
```

#2. When Expression

when expression evaluates a section of code among many alternatives.

```
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> { // Note the block
        print("x is neither 1 nor 2")
    }
}
```

when matches its argument against all branches sequentially until some branch condition is satisfied.

The **else** branch is evaluated if none of the other branch conditions are satisfied.

If many cases should be handled in the same way, the branch conditions may be combined with a comma:

```
when (x) {
    0, 1 -> print("x == 0 or x == 1")
    else -> print("otherwise")
}
```

#3. For Loops

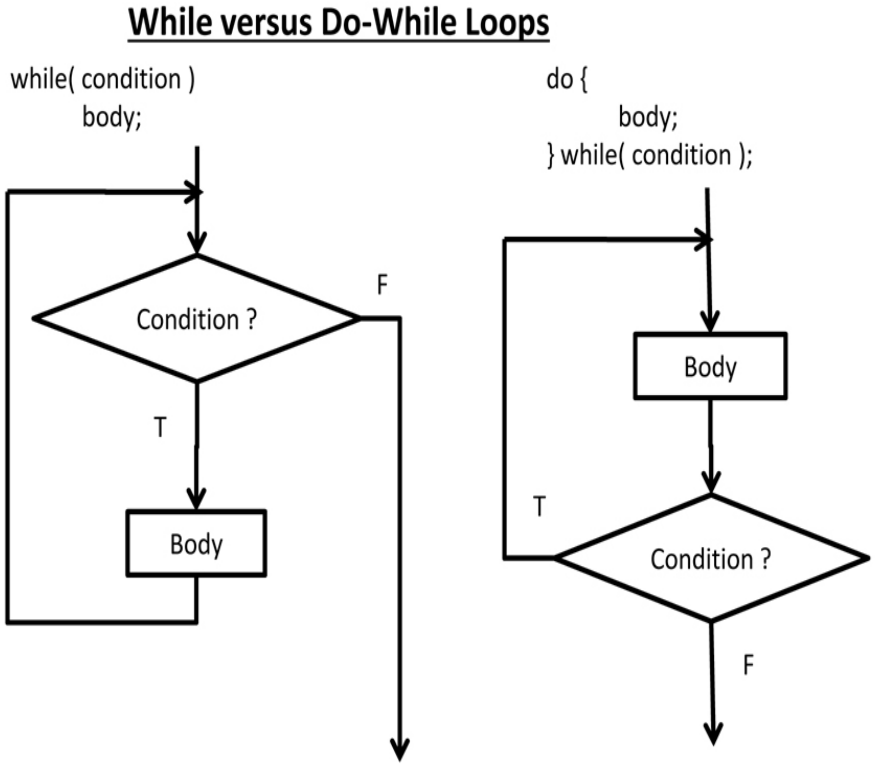
for loop iterates through anything that provides an iterator.

```
for (item in collection) print(item)

for (item: Int in ints) {
    // ...
}

for (i in 1..3) {
    println(i)
}

for (i in 6 downTo 0 step 2) {
    println(i)
}
```

#4. While Loops

while and **do..while** work as usual

```
while (x > 0) {
    x--
}

do {
    val y = retrieveData()
} while (y != null) // y is
visible here!
```



Control Flow. Summary

- If Expression
- When Expression
- For Loops
- While Loops
- Q&A

Returns & Jumps

#1. Returns and jump

Kotlin has three structural jump expressions:

- **return** By default returns from the nearest enclosing function or anonymous function.
- **break** Terminates the nearest enclosing loop.
- **continue** Proceeds to the next step of the nearest enclosing loop.

All of these expressions can be used as part of larger expressions

```
val s = person.name ?: return
```

#2. Break and Continue labels

Any expression in Kotlin may be marked with a label. Labels have the form of an identifier followed by the @ sign.

Then we can qualify a **break** or a **continue** with a label

```
loopA@ for(i in 1..100) {
    println(i)
    if (i ==10) {
        break@loopA
    }
}
```

Returns & Jumps. Summary

- Returns and jump
- Break and Continue labels
- Q&A

Assignment

Assignment 1: Write a program to find all numbers divisible by 7 but not multiples of 5, between 10 and 200 (counting 10 and 200). The resulting numbers will be printed as strings on a line, separated by commas.

Assignment 2: Write a program that input a two-digit integer number. Convert and printout the value of inputted number in binary and hexadecimal.

Assignment 3: Enter an array of integer numbers a_0, a_1, a_2, ..., a_{n-1}. Do not use any other array, print the above array screen in ascending order.

Assignment 4: Enter an string. Count the number of words in the string. Capitalize the first letter of the word if it begins for a sentence.

Assignment 5: Write a program input month and year, print out the number of days that month.

Fpt Software CAMPUS

Thank you