1. Kotlin

It is a high level strongly statically typed language that combines functional and technical part in a same place

2. Hello World

```
fun main() {
    println("Hello World)
}
```

3. Comments

3.1. Single-line Comments

```
// This is a comment
println("Hello World")
```

3.2. Multi-line Comments

```
/* The code below will print the words Hello World
to the screen, and it is amazing */
println("Hello World")
```

4. Variables

```
var mutableVariable = value // variable or mutable
val constantValue = value // constant or immutable
```

4.1. Variable Types

Variables in Kotlin do not need to be declared with a specified type. Kotlin is smart casing types

```
var name = "John"      // String (text)
val birthyear = 1975   // Int (number)
```

But you can specify the type

```
var name: String = "John" // String
val birthyear: Int = 1975 // Int
```

> You can also declare variable without a value, but you need to give it a type to make this possible

4.2. Notes on `val`

When you create a variable with the `val` keyword, the value **cannot** be changed/reassigned

5. Data Types

```kotlin
val myNum = 5             // Int
val myDoubleNum = 5.99    // Double
val myLetter = 'D'        // Char
val myBoolean = true      // Boolean
val myText = "Hello"      // String
```

5.1. Integer types

Type	Size (bits)	Min value	Max value
Byte	8	-128	127
Short	16	-32768	32767
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)

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

5.2. Floating-point types

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

```kotlin
val pi = 3.14 // Double
val one: Double = 1 // Error: type mismatch
val oneDouble = 1.0 // Double


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

Can use underscore to make integer values more readable

```kotlin
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
```

5.3. Booleans

```
val isKotlinFun: Boolean = true
val isFishTasty: Boolean = false
println(isKotlinFun)    // Outputs true
println(isFishTasty)    // Outputs false
```

5.4. Characters

```
val myGrade: Char = 'B'
println(myGrade)
```

> Can't use ASCII values to display a characters like in java

5.5. Strings

```
val myText: String = "Hello World"
println(myText)
```

5.6. Type Conversion

```
val x: Int = 5
val y: Long = x.toLong()
println(y)
```

6. Operations

- `+`
- `-`
- `*`
- `/`
- `%` - modulus - returns the division remainder
- `++` - increment the value - `++x`
- `--` - decrement the value

6.1. Addition assignment

```
var x = 10
x += 5
```

> Can do the same thing with other operators like +, -, *, /, %

6.2. Comparison Operators

- `==`
- `!=`
- `>`
- `<`
- `>=`
- `<=`

6.3. Logic Operator

- `&&`
- `||`
- `!`

7. String

```
var greeting = "Hello"
var greeting: String = "Hello"
```

7.1. Access a String

```
var txt = "Hello World"
println(txt[0]) // first element (H)
println(txt[2]) // third element (l)
```

7.2. Length

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
println("The length of the txt string is: " + txt.length)
```

7.3. String Functions

```
var txt = "Hello World"
println(txt.toUpperCase())   // Outputs "HELLO WORLD"
println(txt.toLowerCase())   // Outputs "hello world"
```

7.4. Comparing Strings

```
var txt1 = "Hello World"
var txt2 = "Hello World"
println(txt1.compareTo(txt2))  // Outputs 0 (they are equal)
```

7.5. Finding a String in a String

```
var txt = "Please locate where 'locate' occurs!"
println(txt.indexOf("locate"))  // Outputs 7
```

7.6. String Concatenation

```
var firstName = "John"
var lastName = "Doe"
println(firstName + " " + lastName)
```

OR

```
var firstName = "John "
var lastName = "Doe"
println(firstName.plus(lastName))
```

7.7. Quotes Inside a String

```
var txt1 = "It's alright"
var txt2 = "That's great"
```

7.8. 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"
// outputs "a was 1, but now is 2"
```

8. Boolean

```
val x = 10
val y = 9
println(x > y) // Returns true, because 10 is greater than 9
```

```
println(10 == 15); // Returns false, because 10 is not equal to 15
```

9. If ... Else

- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`
- Equal to `a == b`
- Not Equal to: `a != b`

> Unlike java, `if .. else` can be used as a **statement** or as an **expression** (to assign a value to a variable) in Kotlin

9.1. if

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

9.2. else

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

9.3. else if

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

9.4. if..else expression

```
val time = 20
val greeting = if (time < 18) {
  "Good day."
} else {
  "Good evening."
}
println(greeting)
```

> When using `if` as an expression, you **must** also include `else` (required)

10. When

```
val day = 4

val result = when (day) {
  1 -> "Monday"
  2 -> "Tuesday"
  3 -> "Wednesday"
  4 -> "Thursday"
  5 -> "Friday"
  6 -> "Saturday"
  7 -> "Sunday"
  else -> "Invalid day."
}
println(result)

// Outputs "Thursday" (day 4)
```

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

```
when {
    x.isOdd() -> print("x is odd")
    y.isEven() -> print("y is even")
    else -> print("x+y is odd")
}
```

11. While Loop

```
while (condition) {
  // code block to be executed
}
```

11.1. Do..While Loop

```
do {
  // code block to be executed
}
while (condition);
```

12. Break and Continue

Jump out of a loop

```
var i = 0
while (i < 10) {
  println(i)
  i++
  if (i == 4) {
    break
  }
}
```

Goes to the next iteration

```
var i = 0
while (i < 10) {
  if (i == 4) {
    i++
    continue
  }
  println(i)
  i++
}
```

Can also use tags to specify exact loop

```
loop@ for (i in 1..100) {
    for (j in 1..100) {
        if (...) break@loop
    }
}
```

Same with `return` when using lambda expressions

```
fun foo() {
    listOf(1, 2, 3, 4, 5).forEach lit@{
        if (it == 3) return@lit // local return to the caller of the lambda - the forEach loop
        print(it)
    }
    print(" done with explicit label")
}
```

13. Arrays

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
```

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

13.1. Primitive type arrays

`ByteArray`, `ShortArray`, `IntArray`

```kotlin
val x: IntArray = intArrayOf(1, 2, 3)
x[0] = x[1] + x[2]
```

```kotlin
// Array of int of size 5 with values [0, 0, 0, 0, 0]
val arr = IntArray(5)

// e.g. initialise the values in the array with a constant
// Array of int of size 5 with values [42, 42, 42, 42, 42]
val arr = IntArray(5) { 42 }

// e.g. initialise the values in the array using a lambda
// Array of int of size 5 with values [0, 1, 2, 3, 4] (values initialised to their index value)
var arr = IntArray(5) { it * 1 }
```

13.2. Change an Array Element

```kotlin
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
cars[0] = "Opel"
println(cars[0])
// Now outputs Opel instead of Volvo
```

13.3. Access the Elements of an Array

```kotlin
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
println(cars[0])
// Outputs Volvo
```

13.4. Array Length/Size

```kotlin
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
println(cars.size)
// Outputs 4
```

13.5. Check if an Element Exists

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
if ("Volvo" in cars) {
  println("It exists!")
} else {
  println("It does not exist.")
}
```

13.6. Loop Through an Array

```
val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")
for (x in cars) {
  println(x)
}
```

14. Collections

14.1. Iteration

```
for (item in items) {
    println(item)
}
```

```
fun saysHello(greeting: String, vararg itemsToGreat:String) {
    itemsToGreat.forEach{itemToGreat ->
        println("$greeting $itemToGreat")
    }
}


fun main() {
    var interestingThings = arrayOf("Kotlin", "Programming", "Comics")
    saysHello("Hello", *interestingThings)
```

14.2. Check if collection contains an object

```
when {
    "orange" in items -> println("juicy")
    "apple" in items -> println("apple is fine too")
}
```

14.3. Use 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) }
```

```
val list2 = listOf("Kotlin", "Java", "C++", "JavaScript", null, null)
    list2
        .filterNotNull()
        .filter(predicate)
//        .take(3)
//        .map{it.length}
        .associate{it to it.length}
        .forEach {
           println(it)
        }
```

15. Nullable Values and null checks

A reference must be explicitly marked as nullable when `null` value is possible. Nullable type names have `?` at the end.

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

16. Type checks and automatic casts

```
fun getStringLength(obj: Any): Int? {
    if (obj is String) {
        // `obj` is automatically cast to `String` in this branch
        return obj.length
    }

    // `obj` is still of type `Any` outside of the type-checked branch
    return null
}
```

or

```kotlin
fun getStringLength(obj: Any): Int? {
    if (obj !is String) return null

    // `obj` is automatically cast to `String` in this branch
    return obj.length
}
```

or

```kotlin
fun getStringLength(obj: Any): Int? {
    // `obj` is automatically cast to `String` on the right-hand side of `&&`
    if (obj is String && obj.length > 0) {
        return obj.length
    }

    return null
}
```

17. Null Safety

Checking for null in conditions

```kotlin
val l = if (b != null) b.length else -1
```

17.1. Safe call

```kotlin
val a = "Kotlin"
val b: String? = null
println(b?.length) // returns null
println(a?.length) // Unnecessary safe call
```

safe calls are usefull in chains

```kotlin
bob?.department?.head?.name
```

Use `let` to perform operations only for non-null values

```kotlin
val listWithNulls: List<String?> = listOf("Kotlin", null)
for (item in listWithNulls) {
    item?.let { println(it) } // prints Kotlin and ignores null
}
```

A safe call can also be placed on the left side of an assignment. Then, if one of the receivers in the safe calls chain is `null`, the assignment is skipped, and the expression on the right is not evaluated at all:

```
// If either `person` or `person.department` is null, the function is not called:
person?.department?.head = managersPool.getManager()
```

17.2. Elvis

```
val l = b ?: -1
// l = b if b is not null
// l = -1 if b is null
```

17.3. The !! operator

Not-null assertion Converts any value to a non-null type and throws an exception if the value is null

```
val l = b!!.length
```

18. For Loop

Unlike Java and other languages, there is no traditional `for` loop in kotlin

```
for (x in 1..10 step 2) {
    print(x)
}
// 13579
println()
for (x in 9 downTo 0 step 3) {
    print(x)
}
//9630
```

`For` iterates through anything that provides an iterator. That means that it:

- has a member or an extension function `iterator()` that returns `Iterator<>`:
  - has a member or an extension function `next()`
  - has a member or an extension function `hasNext()` that returns `Boolean`.

All of these three functions need to be marked as `operator`.

18.1. Ranges

```kotlin
for (chars in 'a'..'x') {
  println(chars)
}
```

```kotlin
for (nums in 5..15) {
  println(nums)
}
```

> The first and last value are included in the range

18.2. Check if a Value Exists

```kotlin
val nums = arrayOf(2, 4, 6, 8)
if (2 in nums) {
  println("It exists!")
} else {
  println("It does not exist.")
}
```

18.3. Break or Continue

Can also be used with for loop

19. Functions

```kotlin
fun main() {
  println("Hello World")
}

fun myFunction() {
  println("I just got executed!")
}
```

19.1. Parameters

```
 fun myFunction(fname: String, age: Int) {
  println(fname + " is " + age)
}

fun main() {
  myFunction("John", 35)
  myFunction("Jane", 32)
  myFunction("George", 15)
}
```

19.1.1. Variable Arguments

```
fun saysHello(greeting: String, vararg itemsToGreat:String) {
    itemsToGreat.forEach{itemToGreat ->
        println("$greeting $itemToGreat")
    }
}


fun main() {
    var interestingThings = arrayOf("Kotlin", "Programming", "Comics")
    saysHello("Hello", *interestingThings)
    // or
    //saysHello("Hello", "Kotlin", "Programming", "Comics")
```

19.2. Return Values

```
fun myFunction(x: Int): Int {
  return (x + 5)
}

fun main() {
  var result = myFunction(3)
  println(result)
}
```

19.2.1. Short hand for Return Values

```
fun myFunction(x: Int, y: Int) = x + y // Return type is inferred

fun main() {
  var result = myFunction(3, 5)
  println(result)
}
```

19.2.2. Void/Unit Return type

```
fun printSum(a: Int, b: Int): Unit {
    println("sum of $a and $b is ${a + b}")
}
```

But Unit can be omitted

20. OOP

20.1. Classes and Objects

20.1.1. Create a Class

```
class Car {
  var brand = ""
  var model = ""
  var year = 0
}
```

> Good practice to start class name with a Capital letter

20.1.2. Create an Object

```
// Create a c1 object of the Car class
val c1 = Car()

// Access the properties and add some values to it
c1.brand = "Ford"
c1.model = "Mustang"
c1.year = 1969

println(c1.brand)   // Outputs Ford
println(c1.model)   // Outputs Mustang
println(c1.year)    // Outputs 1969
```

20.2. Constructor

```
class Car(var brand: String, var model: String, var year: Int)

fun main() {
  val c1 = Car("Ford", "Mustang", 1969)
}
```

20.3. Class Functions

```kotlin
class Car(var brand: String, var model: String, var year: Int) {
  // Class function
  fun drive() {
    println("Wrooom!")
  }
}

fun main() {
  val c1 = Car("Ford", "Mustang", 1969)

  // Call the function
  c1.drive()
}
```

20.3.1. Class Function Parameters

```kotlin
class Car(var brand: String, var model: String, var year: Int) {
  // Class function
  fun drive() {
    println("Wrooom!")
  }

  // Class function with parameters
  fun speed(maxSpeed: Int) {
    println("Max speed is: " + maxSpeed)
  }
}

fun main() {
  val c1 = Car("Ford", "Mustang", 1969)

  // Call the functions
  c1.drive()
  c1.speed(200)
}
```

20.4. Inheritance (Subclass and Superclass)

- `subclass` (child) - the class that inherits from another class
- `superclass` (parent) - the class being inherited from

```
// Superclass
open class MyParentClass {
  val x = 5
}

// Subclass
class MyChildClass: MyParentClass() {
  fun myFunction() {
    println(x) // x is now inherited from the superclass
  }
}

// Create an object of MyChildClass and call myFunction
fun main() {
  val myObj = MyChildClass()
  myObj.myFunction()
}
```

> `open` keyword in front of the **superclass**/parent , to make this the class other classes should inherit properties and functions from. As classes are final by default

21. Package Definition and Imports

```
package my.demo

import kotlin.text.*

// ...
```

22. Program Entry Point

```
fun main() {
    println("Hello world!")
}
// or
fun main(args: Array<String>) {
    println(args.contentToString())
}
```

23. References

- W3schools
- Kotlin Types
- Kotlin Basics

- Kotlin Returns