# Android Development

⇒ Kotlin

↳ official language for Android.

↳ .kt file extension.

↳ use kotlin playground for easy access (Kotlinlang.org)

↳ functional programming language.

↳ a little similar to java.

- Functions

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

↑ name of function

- Printing

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

print ("Hi") → Same line

println ↑ new line print

- Variables

```
Var x = 5
Var Y = "Yash"
```

5 ← int

"Yash" ↑ string

- String Templates

```
dun main () {

    Var x = 5

    println (" World $x")
                      ↑ passes the value of x
}
```

Output → World 5

- Variables

```
Var x : Int = 5  ↗ Declaring variable with type

Val y : Int = 10
```

Var → Value can change

Val → Value cannot change ( immutable)

{ Explicit declaration is not necessary }

- Types of Data

| | |
|---|---|
| Int | { When there is no Datatype Kotlin returns Kotlin. unit } |
| String | |
| Char | |
| Double | { When there can be any datatype kotlin returns any } |
| Boolean | |

- Nullable Types

↳ All variables in Kotlin are by default not NULL

↳ you can create a null variable

```
Var myName: String? = null
```

↑ ? Sign is used to make a variable null

↳ put a ? after the datatype to make it NULL.

```
Var Num: Int ? = null
```

- If - Else and when

```
fun main () {

    Val age = 19
    if ( age > 18) {
        println (" You can vote ")
    }
    else {
        println (" Cannot ")
    }
}
```

← if - else syntax is very similar to java (exactly)

output → You can vote

When → Similar to switch-case in java

```
when (condition) {

    Case 1 → {

        — Code —

    }

    Case 2 → {

        — Code —

    }

    else → {

        — Code

    }

}
```

→ Code : When - 01
       : When - 02

- Arrays

```
Val name = arrayOf ("Yash", "Kotlin", "Java")
```

↑

Declaring an Array

{ if you try to print this directly, it will give Some Address and stuff, if you want to print whats inside the array use a for loop. }

- Loops

```
for (condition) {

        — Code —

}
```

→ printing array

```
for (names in name) {

        println (names)

}
```

```
output → Yash
         Java
         Kotlin
```

While loop

↳ 
```
while (condition) {
        — code —
}
```

Works Just like in Java

Do while

↳ 
```
do {
    — Code —
}
while (Condition)
```

• ranges

→ 
```
for (i in 0..3) {
    print (i)
}
```
↑ range { Creates a temporary list and then iterate over it }

output → 0123

→ 
```
for (i in 2..8) {
    print (i)
}
```

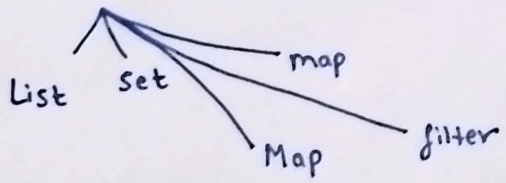
output → 2,345678

```
↳ for (i in 2..8 step 2) {
        print (i)        ↑ leave gaps of 2
  }

  output → 2,4,6,8
```

```
↳ for ( i in 3 downTo 0) {
        print (i)        ↑ print 3 to 0 in descending
                           order
  }

  output → 3201
```

→ Code : range

• Collections



→ Code : funAdd

: funGreet

: funDefault

↑ if you do not pass anything while calling fnc.

- Higher order functions

↳ a function that takes another function as parameter

```
fun operation (a: Int, b: Int, operate: (Int,Int) : Int {
    return (a,b)
}
```

↑ first function

↑ 2nd function

```
fun main () {
    val output = operation (4,5, { - code - })
    print (output)
```

↑ Parameter of 1st

↑ Parameters / code for 2nd.

→ Code : higherFnc

↳ List

- ↳ immutable
- ↳ ListOf () ← just like arrays
- ↳ for a mutable list → mutableListOf ()
- ↳ access using index
- ↳ update using index ( only for mutable)
- ↳ ListOf < String> ()

  ↑ define with datatype

  <> : generics

↳ map

- ↳ key : value pair ( Dictionary)
- ↳ ordered

↳ set

- ↳ unordered
- ↳ unique elements

• Functions

```
fun name (parameter) : Int {
       — code —
}
```

Keyword → fun

Paramater(s) with types → (parameter)

↑ name of function

↑ return type

• OOPS

Keyword

```
Class Person {
```

Class_name

```
    – Code –
}
```

Instance | object

Creating an object → Person ( )

Class_name

→ Code : class_1

: class_2

: class_3

• Special classes

1) Data classes

- ↳ makes it easy to create classes that is used to store values
- ↳ are provided with methods for copying, getting a string representation and using instances in collection
- ↳ copy ( )
- ↳ hashCode ( )
- ↳ equals ( )

Code : Data

: dataHashEqual

2) Enum classes

↳ used for certain values

↳ Enumeration is named list of constants

↳ each constant is an object

→ Code : Enum

3) Sealed classes

↳ provides more control over inheritance

↳ defines a set of subclasses inside it

4) inline class

↳ subset of value based classes

↳ don't have an identity, can only hold values

- filter and map

filter → To sort | filter

map → To transform

→ Code : filter
        : map