JANUARY					2008
Mon		05	12	19	26
Tue		06	13	20	27
Wed		07	14	21	28
Thu	01	08	15	22	29
Fri	02	09	16	23	30
Sat	03	10	17	24	31
Sun	04	11	18	25	

KOTLIN

=> val = 50
    → can't be changed

=> var = 50
    → can be changed

=> Datatypes = String, Int, Double, Boolean
    ↳ case sensitive

=> var name: String = "Yash"
or val
    ↳ can be replaced by "Any"

=> print("Yash")
    → print value

=> println("Yash")
    → print value followed by a new-line

=> val age: String = "" + 4.2 + 6 + "!"
    ↳ valid, use any values to concatenate them, but they must be in between two strings

=> Initializing long:

```
val i = 10L
val i3 : Long = 10
```

=> Float type is also present, but it has lower precision.

```
val pi = 3.14F
```

=> Type casting = i=10

```
var p = i.toDouble()
var q = i.toLong()
```

Notes

Appointment

=> += , -= , *= , /= , %= are present in kotlin

=> ++a, a++, --a, a-- is present in kotlin

=>
```
val c1 = 'A'
println(c1.code) //65
val a: Char = 'a'
```

=>
```
println("ABC".length) //3
```

=> String Template
```
val name = "Yash"
val age = 22
println("User $name, age $age") //User Yash
                                     age 22
```

=>
```
val a = 1
val b = 2
```

```
print(" a+b = ${a+b}") // a+b=3
```

=> multi-line strings should be surrounded by tripple double-quotes.

=>
```
val char = 'a'
val st = char.toString()
```

Notes

Appointment

JANUARY					2008
Mon		05	12	19	26
Tue		06	13	20	27
Wed		07	14	21	28
Thu	01	08	15	22	29
Fri	02	09	16	23	30
Sat	03	10	17	24	31
Sun	04	11	18	25	

=> Boolean functions with texts

```
      val sal = "Hello"
      val tv = sal.startsWith("Hel") // true
                  .endsWith("lo") // true
                  .first() //'H'
                  .last () // 'o'
                  .equals ("Hello") // true
                  .uppercase() //HELLO
                  .lowercase() // hello
                  .substring (2) //llo
       fir = sal[0] //'H'
```

=> &&, ||, !=, ==, >, <, >=, <= present too

Notes

Appointment

```
=> if(a==b) {
       ___
     ___ ___ ___

   } else {

       ___ ___
     ___ ___ ___

   }
```

```
=> val text = if(ac=="free") "Free" else "Paid"
```

```
=> val tip =
            if (true) {
                println(5)
                3 // Ignored
                3 // Ignored
                2 // Assigned to tip
            } else { 5 }
```

```
=> if(    ){

   } else if(      ){

   }
```

Notes

Appointment

```
=> checking types
     val a = 35
     print(val is Int) // true
```

JANUARY 2008					
Mon		05	12	19	26
Tue		06	13	20	27
Wed		07	14	21	28
Thu	01	08	15	22	29
Fri	02	09	16	23	30
Sat	03	10	17	24	31
Sun	04	11	18	25	

```
=> val password = "ABC"

   when{
          password ==" "  -> {
                print(1)
          }
          password.length < 7  ->{
                print (2)
          }
          else -> {
               println (" OK")
          }

      }

=> when(num) {
          1 -> {
               print(" one")
          }

=>  val a = when(num) {
                1 -> {
                     "Hello"
                }
```

Notes

```
        print (a) //"Hello"
=) to check ranges
            2,3,4,5 -> { }
            2..5  -> { }
```

Appointment

```
⇒ While ( ){

  }

⇒ for(elements of list){
     ⁼
  }

Ex    val letters = listOf("A", "B", "C")
      for(l in letters){
          println(" The letter is $l ")
      }

      for(i in 1..5)

      for(i in 1 until 5) → i=1,2,3,4
      for(i in 5 downTo 1) → i=5,4,3,2,1
      for(i in 1..5 step 2) → i = 1,3,5
```

=> top-level variables — Global variables
— declared outside of all functions
(like C/C ++)

=> Local functions can also be defined in Kotlin. (A function defined within a~~nd~~ function)

=> Returning values

type of ret. value

```
fun double(value: Int) : Int {
        return value * 2
}
```

~~=> absolute (-123)~~ //123

=> Single line functions.

```
fun triArea(wid: Double, len: Double) : Double
                    = width x len/2
```

// is equivalent to ↓

```
fun triArea(wid: Double, len: Double) : Double {
        return w x l/2
}
```

Notes

Appointment

Sun 07 14 21

⇒ Named & default arguements

```
fun cheer (now : String = "Hello",
                   who : String = "World") {

   cheer ()          // Hello World
   cheer ("Hi")      // Hi World
   cheer (who = "Yash") // Hello Yash
```