



# Mobile Application Development

---

Produced by Dave Drohan ([david.drohan@setu.ie](mailto:david.drohan@setu.ie))

Department of Computing & Mathematics  
South East Technological University  
Waterford, Ireland

Updated & Delivered by Gongzhe Qiao ([003969@nuist.edu.cn](mailto:003969@nuist.edu.cn))

Department of Computer Science  
Nanjing University of Information Science and Technology  
Nanjing, China

[nuist.edu.cn](http://nuist.edu.cn)



[setu.ie](http://setu.ie)



# Introducing Kotlin Syntax - Part 1.3

---



# Agenda

---

- ❑ Basic Types
- ❑ Local Variables (`val` & `var`)
- ❑ Functions
- ❑ Control Flow (`if`, `when`, `for`, `while`)
- ❑ Strings & String Templates
- ❑ Ranges (and the *`in`* operator)
- ❑ Type Checks & Casts
- ❑ Null Safety
- ❑ Comments



# Agenda

---

- ☐ Basic Types
- ☐ Local Variables (val & var)
- ☐ Functions
- ☐ Control Flow (if, when, for, while)
- ☐ **Strings & String Templates**
- ☐ **Ranges (and the *in* operator)**
- ☐ **Type Checks & Casts**
- ☐ Null Safety
- ☐ Comments



# Strings and String Templates

---

Escaped strings, raw strings, literals, templates



# Strings

- ❑ Strings are represented by the type **String**. Strings are **immutable**. Elements of a string are characters that can be accessed by the indexing operation: **s[i]**. A string can be iterated over with a **for-loop**:

```
1 fun main() {  
2   val str = "abcd"  
3   for (c in str) {  
4     println(c)  
5   }  
6 }
```

a  
b  
c  
d

# String Literals

---

- ❑ Kotlin has two types of string literals:
  - escaped strings** that may have escaped characters in them
- ❑ and
  - raw strings** that can contain newlines and arbitrary text.

# String Literals

- ❑ Kotlin has two types of string literals:
  - escaped strings** that may have escaped characters in them
- ❑ and
  - raw strings** that can contain newlines and arbitrary text.

An **escaped string** is very much like a Java string

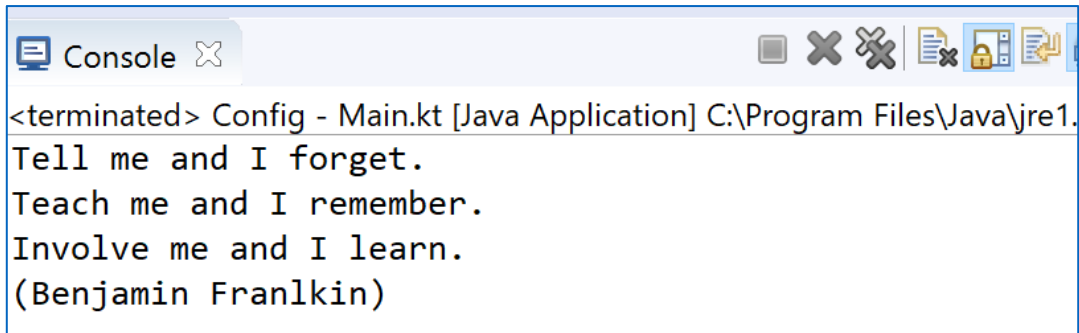
```
val s = "Hello, world!\n"
```

A **raw string** is delimited by a triple quote ("""), contains no escaping and can contain new lines and any other characters.

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



# String Literals



```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\jre1.  
Tell me and I forget.  
Teach me and I remember.  
Involve me and I learn.  
(Benjamin Franklkin)
```

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

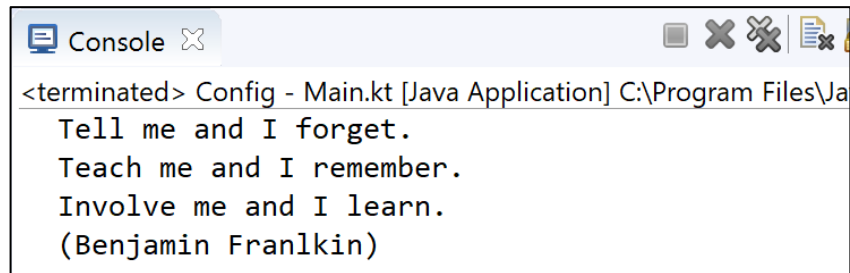
- ❑ You can remove leading whitespace with `trimMargin()`

# String Literals

- ❑ By default `|` is used as margin prefix, but you can choose another character and pass it as a parameter like `trimMargin(">")`.

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

print(text)
```



```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Ja
Tell me and I forget.
Teach me and I remember.
Involve me and I learn.
(Benjamin Franklin)
```

*Note the impact of the two spaces we put between > and the text*

# String Templates

- ❑ Strings may contain template expressions, i.e. pieces of code that are evaluated and whose results are concatenated into the string.
- ❑ A template expression starts with a dollar sign (\$) and consists of either a simple name:

```
val i = 10  
val s = "i = $i" // evaluates to "i = 10"
```

- ❑ or an arbitrary expression in curly braces:

```
val s = "abc"  
val str = "$s.length is ${s.length}" // evaluates to "abc.length is 3"
```

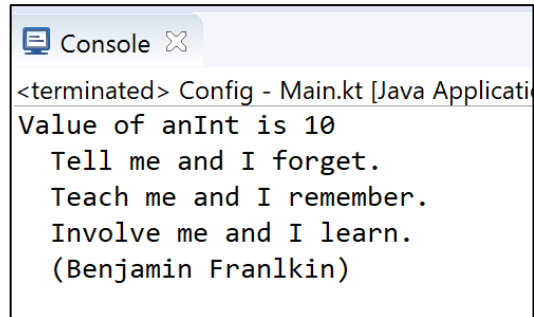
# String Templates

- ❑ Templates are supported both inside **raw strings** and inside **escaped strings**.

```
val anInt = 10
val aString = "Value of anInt is ${anInt}\n"

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

print(aString)
print(text)
```



Console

<terminated> Config - Main.kt [Java Application]

Value of anInt is 10  
Tell me and I forget.  
Teach me and I remember.  
Involve me and I learn.  
(Benjamin Franklin)

# String Templates

```
1 fun main() {  
2     var a = 1  
3     // simple name in template:  
4     val s1 = "a is $a"  
5  
6     a = 2  
7     // arbitrary expression in template:  
8     val s2 = "${s1.replace("is", "was")}, but now is $a"  
9     println(s2)  
10 }
```

s1

"a is 1"

a was 1, but now is 2

# Ranges

---

The **in** operator



# Range

Check if a number is within a range using *in* operator:

```
1 fun main(args: Array<String>) {  
2     val x = 10  
3     val y = 9  
4     if (x in 1..y+1) {  
5         println("fits in range")  
6     }  
7 }
```

fits in range

Check if a number is out of range:

```
1 fun main(args: Array<String>) {  
2     val list = listOf("a", "b", "c")  
3  
4     if (-1 !in 0..list.lastIndex) {  
5         println("-1 is out of range")  
6     }  
7     if (list.size !in list.indices) {  
8         println("list size is out of valid list indices range too")  
9     }  
10 }
```

-1 is out of range

list size is out of valid list indices range too

# Range

Iterating over a range:

```
1 fun main(args: Array<String>) {  
2     for (x in 1..5) {  
3         print(x)  
4     }  
5 }
```

12345

Iterating over a progression:

```
1 fun main(args: Array<String>) {  
2     for (x in 1..10 step 2) {  
3         print(x)  
4     }  
5     for (x in 9 downTo 0 step 3) {  
6         print(x)  
7     }  
8 }
```

135799630




# Type Checks & Casts



---



# is and !is operators

```
fun main(args: Array<String>) {  
    val aString = "I am a String"  
  
    if (aString is String) {  
        println("String length is: ${aString.length}")  
    }  
  
    if (aString !is String) { // same as !(aString is String)  
        print("Not a String")  
    }  
    else {  
        println("String length is: ${aString.length}")  
    }  
}
```

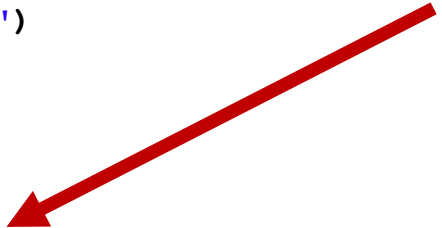




 Console 

```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\jre1.8.0_77\  
String length is: 13  
String length is: 13
```

# Smart Casts (an example using `if`)

```
fun main(args: Array<String>) {  
    demo ("I am a String")  
    demo (12)  
}  
  
fun demo(x: Any) {  
    if (x is String) {  
        println(x.length) // x is automatically cast to String  
    }  
    else{  
        println(x.javaClass)  
    }  
}
```





 Console 

```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\jre1.8.0_77\bin\java  
13  
class java.lang.Integer
```

# Smart Casts (an example using **when**)

```
fun main(args: Array<String>) {  
    demo (12)  
    demo ("I am a String")  
    demo (intArrayOf(1,2,3,4))  
}  
  
fun demo(x: Any) {  
    when (x) {  
        is Int -> println(x + 1)  
        is String -> println(x.length + 1)  
        is IntArray -> println(x.sum())  
    }  
}
```

 Console 

<terminated> Config - Main.kt [Java Application] C:\P

13

14

10



## References

---

Sources:

<http://kotlinlang.org/docs/reference/basic-syntax.html>

<http://petersommerhoff.com/dev/kotlin/kotlin-for-java-devs/>

<https://www.programiz.com/kotlin-programming>

<https://medium.com/@napperley/kotlin-tutorial-5-basic-collections-3f114996692b>

