# Mobile Application Development


Produced by

Dave Drohan (david.drohan@setu.ie)

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

Updated & Delivered by

Gongzhe Qiao (003969@nuist.edu.cn)

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


nuist.edu.cn


setu.ie

# Introducing Kotlin Syntax - Part 1.4

# 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

# Null

Using nullable values and checking for null

# Null Safety

- In Kotlin, the type system distinguishes between references that can hold **null** (nullable references) and those that cannot (non-null references)
- The Kotlin compiler makes sure you don't, by accident, operate on a variable that is null.

# Null Safety – a non-null reference

- A regular variable of type String can not hold **null**

```
var a: String = "abc"
a = null // syntax error
```

- Calling a method / accessing a property on variable a, is guaranteed not to cause an NullPointerException (NPE)

```
val l = a.length
```

# Null Safety – a nullable reference

- To allow nulls, we can declare a variable as a nullable string, written String?

```
var b: String? = "abc"
b = null // ok
```

```
val l = b.length // syntax error: as
                 // variable 'b'
                 // can be null
```

- However, there are many ways around this….

# Null Safety – a nullable reference

- To allow nulls, we can declare a variable as a nullable string, written String?

```
var b: String? = "abc"
b = null // ok
```

Option 1: you can explicitly check if b is null, and handle the two options separately:

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

# Null Safety – a nullable reference

- To allow nulls, we can declare a variable as a nullable string, written String?

```
var b: String? = "abc"
b = null // ok
```

Option 2: you can use the safe call operator ?. This returns b.length if b is not null, and null otherwise

```
b?.length
```

# Null Safety – a nullable reference

- To allow nulls, we can declare a variable as a nullable string, written String?

```
var b: String? = "abc"
b = null // ok
```

Option 3: you can use the !! Operator. This forces a call to our method and will return a non-null value of b or throw an NPE if b is null. Use sparingly!

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

# Null Safety – The Elvis Operator, ?:

- When we have a nullable reference b, we can say:

"if b is not null, use it, otherwise use some non-null value x"

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

# Null Safety – The Elvis Operator, ?:

- When we have a nullable reference b, we can say:

  "if b is not null, use it, otherwise use some non-null value x"

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

- Along with the complete if-expression, this can be expressed with the Elvis operator, written ?:

```
val l = b?.length ?: -1
```

- If the expression to the left of ?: is not null, the Elvis operator returns it, otherwise it returns the expression to the right.

# Nullable – nullable returns

- A reference must be explicitly marked as nullable (i.e. ?) when **null** value is possible.

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

Return null if the return value does not hold an integer:

# Comments

Single line, block, KDoc

# Comments – single line and block comments

Just like Java and JavaScript, Kotlin supports end-of-line and block comments.

```
// This is an end-of-line comment

/* This is a block comment
   on multiple lines. */
```

Unlike Java, block comments in Kotlin can be nested.

# Comments – KDoc (equivalent to JavaDoc)

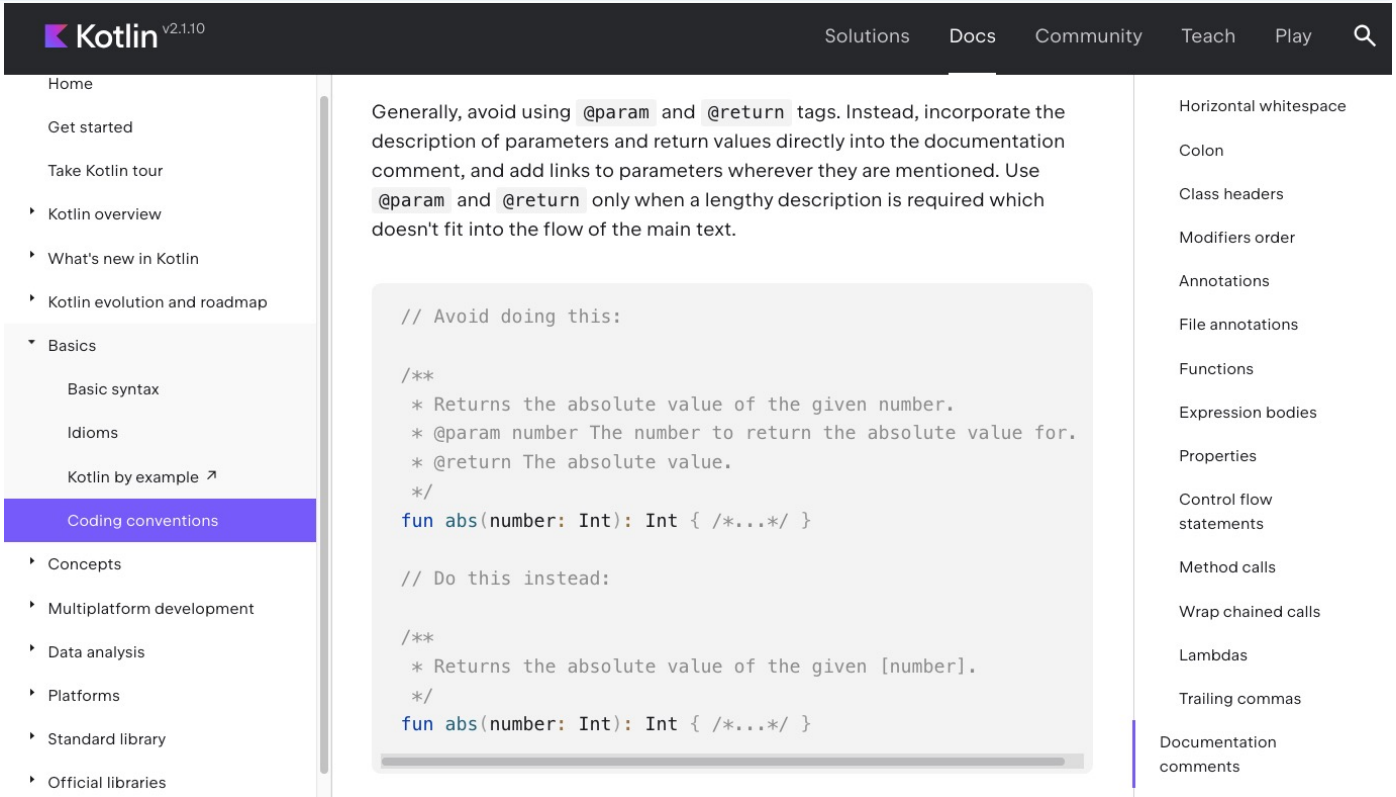
```
/**
 * A group of *members*.
 *
 * This class has no useful logic; it's just a documentation example.
 *
 * @param T the type of a member in this group.
 * @property name the name of this group.
 * @constructor Creates an empty group.
 */
class Group<T>(val name: String) {
    /**
     * Adds a [member] to this group.
     * @return the new size of the group.
     */
    fun add(member: T): Int { ... }
}
```

# Comments – KDoc

| Block tags | Currently supported KDoc block tags |
|---|---|
| @param <name> | Documents a value parameter of a function or a type parameter of a class, property or function. |
| @return | Documents the return value of a function. |
| @constructor | Documents the primary constructor of a class. |
| @receiver | Documents the receiver of an extension function. |
| @property <name> | Documents the property of a class which has the specified name. |
| @throws <class>, @exception <class> | Documents an exception which can be thrown by a method. |
| @sample <identifier> | Embeds the body of the function with the specified qualified name into the documentation for the current element, in order to show an example of how the element could be used. |
| @see <identifier> | Adds a link to the specified class or method to the **See Also** block of the documentation. |
| @author | Specifies the author of the element being documented. |
| @since | Specifies the version of the software in which the element being documented was introduced. |
| @suppress | Excludes the element from the generated documentation. Can be used for elements which are not part of the official API of a module but still have to be visible externally. |

For more info: http://kotlinlang.org/docs/reference/kotlin-doc.html

# However …




For more info: https://kotlinlang.org/docs/coding-conventions.html#documentation-comments


# 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


Thanks.