Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

JS 2 K - Part 1

JavaScript to Kotlin

Variables and Constants

JavaScript

```
const name = "Jon Snow";
let isAlive = true;
let role; // Declared but not initialized
```

Kotlin

```
// Types can be inferred from initialized variables
val name = "Jon Snow"
var isAlive = true

// If the variable is declared but not initialized, a type annotation is required:
var role: String
```

Strings

JavaScript

```
const house = "Stark";
const motto = `
Winter
is
comming
`;
```

Kotlin

```
val house = "Stark"
val motto = """
Winter
is
comming
"""
```

String Interpolation

JavaScript

```
const action = `Attacking using a ${weapon}`;

const result = `Looks like you will ${user.getFate()}`;
```

Kotlin

```
const action = "Attacking using a $weapon"

const result = "Looks like you will ${user.getFate()}" // curly brackets are only
```

Functions : Named Functions

JavaScript

```
function double(num) {
    return num * 2;
}
double(2); // 4


// Default values
function shout(message, postfix = "!!!") {
    return `${message.toUpperCase()}${postfix}`;
}
shout("hey"); // HEY!!!
```

Kotlin

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

// Default values
fun shout(message: String, postfix = "!!!"): String {
    return "${message.toUpperCase()}$postfix"
}
```

Functions : Named Parameters

Kotlin

```
fun reformat(str: String,
             normalizeCase: Boolean = true,
             upperCaseFirstLetter: Boolean = true,
             divideByCamelHumps: Boolean = false,
             wordSeparator: Char = ' ') {
    ...
}

reformat("SomeString", normalizeCase = false, divideByCamelHumps = true)
```

Function Expressions

JavaScript

```
const double = (num) => num * 2; // Single line has implicit return

const square = (num) => {
    const result = num * num;
    return result; // Multi line: No implicit return
}
```

Kotlin

```
val double = { num:Int -> num * 2 }

val square = { num: Int ->
    val result = num * num
    // The last expression in a lambda is always considered the return value:
    result
}
```

“arrow functions”
Vs
lambdas

Function Expressions

JavaScript

```
const carModels = cars.map((car) => car.model );
const oldEnough = users.filter((user) => user.age >= 21 );
```

Kotlin

```
val carModels = cars.map { it.model }
val oldEnought = users.filter { it.age >= 21 }
```

“arrow functions”
Vs
lambdas

Flow Control

JavaScript

```
if (number > 0) {
    console.log("Positive number");
} else {
    console.log("Negative number");
}
```

Kotlin

```
if (number > 0) {
    print("Positive number")
} else {
    print("Negative number")
}
```

if / else

Flow Control

Kotlin by JetBrains

if / else

JavaScript

```
let result;
if (number > 0) {
    result = "Positive number";
} else {
    result = "Negative number";
}
```

Kotlin

```
val result = if (number > 0) {
    "Positive number"
} else {
    "Negative number"
}
```

Flow Control

JavaScript

```
const result = number > 0 ? "Positive number" : "Negative number";
```

Kotlin

if / else

```
val result = if (number > 0) "Positive number" else "Negative number"
```

JavaScript

Flow Control

Kotlin by JetBrains

```
switch (selectedFruit) {
  case "orange":
    console.log("Oranges are 59 cents a pound.");
    break;
  case "apple":
    console.log("Apples are 32 cents a pound.");
    break;
  case "cherry":
    console.log("Cherries are one dollar a pound.");
    break;
  case "mango":
  case "papaya":
    console.log("Mangoes and papayas are 3 dollars a pound.");
    break;
  default:
    console.log(`Sorry, we are out of ${selectedFruit}.`);
}
```

switch/when

Flow Control

```
when(selectedFruit) {
    "orange" -> print("Oranges are 59 cents a pound.")
    "apple" -> print("Apples are 32 cents a pound.")
    "cherry" -> print("Cherries are one dollar a pound.")
    "mango", "papaya" -> print("Mangoes and papayas are 3 dollars a pound.")
    else -> print("Sorry, we are out of $selectedFruit.")
}
```

switch/when

Flow Control

Kotlin by JetBrains

Loops

JavaScript

```javascript
for (let i = 1; i<=10; i++) {
    console.log(i);
}
// 1 2 3 4 5 6 7 8 9 10


const places = ["New York", "Paris", "Rio"];
for (const place of places) {
  console.log(`I Love ${place}`);
}
// I Love New York
// I Love Paris
// I Love Rio
```

Kotlin

```kotlin
for (i in 1..10) {
    print(i)
}
// 1 2 3 4 5 6 7 8 9 10


val places = listOf("New York", "Paris", "Rio")
for (place in places) {
    println("I Love $place")
}
// I Love New York
// I Love Paris
// I Love Rio
```

References

Sources: https://dev.to/cassiozen/kotlin-for-js-devs-part-1-5bld


Thanks.