

Lecture 1

COMP 3717- Mobile Dev with Android Tech

Welcome Everyone

- Instructor: Charles Tapp
 - Office hours: By appointment
 - Contact Info: ctapp2@bcit.ca
 - Slack
 - Info posted on learning hub

Lectures and Labs

- Lecture slides will be printed and posted on D2L (Learning Hub) before class
- Lab assignments will be available right after lecture and due the night before next lecture (1 week)

Grading Structure

- Breakdown:
 - Assignment: 10%
 - Labs: 30%
 - Midterm: 30%
 - Final: 35%

*Average of midterm and final exams must be 50% or higher to pass course

- Ex. If you get 49% on the final and 49% on the midterm, but get 100% on your assignments, you will fail the course.

Attendance

- Lectures & Labs
 - We will be checking for participation and attendance, in-person participation is required
- Let me know if going to miss a class or lab via a valid reason

Course Materials

- Android Studio (Required)
 - Latest stable version
 - Narwhal 2025.1.2
- Physical android device (Recommended)
 - Not required but recommended if you have one

What is Android?

- Android is the most popular operating system in world
 - based on Linux
- Was designed mainly for touch screen mobile devices
- Android makes roughly 20-30 billion a year with the Play Store being responsible for most of it

What is Android? (cont.)

- SDK (Software developer Kit)
 - Latest: Android 16 (API level 36)
 - <https://developer.android.com/about/versions/16>
 - SDK Tools
 - Build Tools
 - Platform Tools
 - Emulator
 - Google Play Services
- Gradle is the build software used in Android Studio
 - Automates and manages the build process for us when creating builds

The Android OS is everywhere

- Smartphones
- Tablets
- Smartwatches (Wear OS)
- TVs (Android TV, Google TV)
- Android Automotive (Different then, Android Auto)
- Amazon Fire OS (TVs, tablets)
- Digital cameras
- Refrigerators
- Gaming consoles (Ouya)

Languages & other SDK's

- Native development
 - Kotlin and Java with Android Studio
- Cross platform tools that allow you to build for android
 - Jetpack Compose (Google)
 - Flutter (Google)
 - React (Meta)
 - Xamerin (Microsoft)
 - Unity
 - etc

Kotlin

- Google's preferred language for Android app development
- Combines objected oriented and functional programming features
 - Functional
 - Data is transformed by creating functions
 - OOP
 - Data is stored in objects

Kotlin vs Java

- More concise and streamlined than Java
- Can be used in any situation you would use Java
 - Such as server-side code
- Most people who make the switch from Java to Kotlin, don't turn back

Kotlin basics

- All programs need an *entry point* to be run which is the main function

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

- When building for Android, we don't use the main function directly

Kotlin basics (cont.)

- **fun** is the keyword for creating a function
- **main** is the name of our function
 - We provide parenthesis () as part of the Kotlin syntax
- Inside the curly braces is called the **code block**, where our logic goes

```
fun main()  
{  
  
}  

```

Kotlin basics (cont.)

- *println* is a function we are calling inside our main function
- Here we are passing in a *string* but we could pass in any type because *println* is an overloaded function

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

- *println* prints a new line to the console

Variables

- We can create a variable like so

```
fun main() {  
    val name: String = "Jerry"  
}
```

- variable name
- variable type
- variable value

Variables (cont.)

- Here we can use *println* to display some variables to the console

```
val name: String = "Jerry"  
val height: Int = 5  
  
println("$name is $height feet tall")
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...  
Jerry is 5 feet tall  
  
Process finished with exit code 0
```

Basic Datatypes

- Int, Double, Long, Float, Boolean, String and Char

```
fun main() {  
  
    val num: Int = 5  
    val d: Double = 5.5  
    val l: Long = 500L  
    val f: Float = 2.3F  
    val b: Boolean = true  
    val s: String = "Good Morning"  
    val c: Char = 'g'  
  
}
```

Basic Datatypes (cont.)

- Every variable type can be type *Any* because every Kotlin class has *Any* as a superclass

```
val num: Any = 5
val d: Any = 5.5
val l: Any = 500L
val f: Any = 2.3F
val b: Any = true
val s: Any = "Good Morning"
val c: Any = 'g'
```

- *Any* has its use cases, but it is best to use a specific type (i.e. String) for now

Type Inference

- Kotlin can determine the type for us when we initialize a variable

```
val num = 5
val d = 5.5
val l = 500L
val f = 2.3F
val b = true
val s = "Good Morning"
val c = 'g'
```

Val vs Var

- Val
 - Read only
 - Can only be assigned once
- Var
 - Mutable
 - Can be reassigned

```
val num = 5
num = 10
```

Val cannot be reassigned
Change to 'var' Alt+Shift+Enter

- It's good practise to use *val* everywhere unless you need to use *var*

Const

- Const values are read only variables known before code execution
 - Better for performance
- Since they are determined at compile time you can only define primitive types and strings as const
- Anything besides primitives and strings may have runtime side effects contradicting how constants work

Const (cont.)

- Const variables cannot be declared inside a function
 - Class level variable

```
fun main() {  
    const val num = 5  
}
```

Modifier 'const' is not applicable to 'local variable'

Remove 'const' modifier Alt+Shift+Enter More a

```
const val num = 5  
  
fun main() {  
  
}
```

Null Safety

- Null safety was created to avoid null references
 - A common bug in the past!
- Kotlin allows types to be null or not null

- Nullable

```
val str:String? = null
```

- Non null

```
val str:String = "Good Morning"
```


Null Safety (cont.)

- You can see that we can't assign null to a non null type

```
val str:String = null
```

Null can not be a value of a non-null type String

Null Safety (cont.)

- When we try to use a property (*length*) of a nullable type we get an error

```
val str:String? = null

println(str.length)
```

Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type String?
Surround with null check Alt+Shift+Enter More actions... Alt+Enter

- We have to use certain operators to protect us from a null exception

```
val str:String? = null

println(str?.length)
```

Null Safety (cont.)

```
val str:String? = null  
  
println(str?.length)
```

- Running this code will print null to the console instead of a null exception

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...  
null  
  
Process finished with exit code 0
```

- This is how Kotlin protects us from null pointer exceptions

Null Safety (cont.)

- The !! Operator
 - Only use if you know it won't be null

```
private var str:String? = "Hello World"

private fun doSomething() : Int{

    return str!!.length
}
```

Strings

- Strings have many **properties** and **methods** we can work with

```
val name:String = "Good morning"
println(name.)
```

<input type="radio"/>	trimStart()	for String in kotlin.text	String
<input type="radio"/>	trimStart(vararg chars: Char)	for String in kotlin.text	String
<input type="radio"/>	trimStart {...}	(predicate: (Char) -> Boolean) for String in kotl...	String
<input type="radio"/>	uppercase()	for String in kotlin.text	String
<input type="radio"/>	uppercase(locale: Locale)	for String in kotlin.text	String
<input checked="" type="checkbox"/>	indices	for CharSequence in kotlin.text	IntRange
<input checked="" type="checkbox"/>	lastIndex	for CharSequence in kotlin.text	Int
<input type="radio"/>	last()	for CharSequence in kotlin.text	Char
<input type="radio"/>	last {...}	(predicate: (Char) -> Boolean) for CharSequence in kotli...	Char
<input type="radio"/>	all {...}	(predicate: (Char) -> Boolean) for CharSequence in kot...	Boolean
<input type="radio"/>	any()	for CharSequence in kotlin.text	Boolean

Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards **Next Tip** :

Strings (cont.)

- Here are some common examples

```
val name = "Good morning"  
println(name)  
println(name.length)  
println(name.uppercase())  
println(name.lowercase())  
println(name.isEmpty())
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...  
Good morning  
12  
GOOD MORNING  
good morning  
false  
  
Process finished with exit code 0
```

Strings (cont.)

- We can also print the character index of the string using [] brackets

```
val name = "Good morning"  
println(name[0])  
println(name[5])  
println(name[9])
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...  
G  
m  
i  
  
Process finished with exit code 0
```

Strings (cont.)

- You can still will get out of bounds errors in Kotlin, but the compiler will try and give you a heads up

```
val name = "Good morning"  
println(name[12])
```

Index is always out of bounds

⋮

"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 12

at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:48)

at java.base/java.lang.String.charAt(String.java:1513)

at com.bcit.lecture1.MainKt.main(Main.kt:6)

at com.bcit.lecture1.MainKt.main(Main.kt)

Process finished with exit code 1

String Template

- If you want to concatenate two variables in a string don't do this

```
val name = "Jerry"  
val height = 5  
  
println(name + " is " + height + " feet tall")
```

- The compiler will want you to convert it to a template

```
println(name + " is " + height + " feet tall")
```

'String' concatenation can be converted to a template

Convert 'String' concatenation to a template Alt+Shift+Enter

String Template (cont.)

- To use a *String* template you wrap your expression like so *`${expression}`*
- If you just have a single variable you can omit the `{ }` braces like below

```
val name = "Jerry"  
val height = 5  
  
println("$name is $height feet tall")
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...  
Jerry is 5 feet tall  
  
Process finished with exit code 0
```

Strings (cont.)

- You can also use the format method when concatenating variables using the %s operator

```
val str = "%s is %s feet tall"

println(str.format(...args: "Alex", 4))
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Alex is 4 feet tall

Process finished with exit code 0
```

Multiline Strings

- Pressing **shift + "** three times, then **enter**, will create a multiline string

```
val str = """  
  
""".trimIndent()
```

Multiline Strings (cont.)

- Here we can create a multi-line string, even with indents

```
val name = "Jerry"

val str = """
    $name and %s went
on the roller coaster and
    had a fantastic time
""".trimIndent()

println(str.format( ...args: "Alex"))
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
    Jerry and Alex went
on the roller coaster and
    had a fantastic time

Process finished with exit code 0
```

Multiline Strings (cont.)

- *trimIndent()* is not needed but you may prefer it for formatting purposes
 - It removes a common minimal indent from each line
 - And removes first and last line if they are blank
- Here is the same example without *trimIndent()*

```
val name = "Jerry"

val str = """
    $name and %s went
on the roller coaster and
    had a fantastic time
    """

println(str.format(...args: "Alex"))
```

```
"C:\Program Files\Android\Android Studio\jbr\bin

    Jerry and Alex went
on the roller coaster and
    had a fantastic time

Process finished with exit code 0
```



String Comparison

- To compare the values of two variables in Kotlin we use the `==` operator
 - `.equals` does the same thing

```
val name = "Jerry"  
val name2 = "Sarah"  
  
println(name == name2)  
println(name.equals(name2))
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
```

```
false
```

```
false
```

```
Process finished with exit code 0
```


String referential equality

- To compare the memory location of two variables we use `===`

```
val name = "Jerry"  
val name2 = "Sarah"  
  
println(name === name2)
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...  
false  
  
Process finished with exit code 0
```

String pool memory

- Notice that when we use `===` on two different strings with the same value, it is true

```
val name = "Jerry"  
val name2 = "Jerry"  
  
println(name === name2)
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...  
true  
  
Process finished with exit code 0
```

- This is because of the string pool memory, which is different than heap memory

String pool memory (cont.)

- It checks if there is already a value of that string in the string pool memory
- If there is, then it points to that string in memory
- So, in the previous example, only one string is created on the heap

String pool memory (cont.)

- In this example, *name2* is a new object created on the heap, so comparing their memory locations will return false

```
val name = "Jerry"  
val name2 = String("Jerry".toCharArray())  
  
println(name == name2)
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...  
false  
  
Process finished with exit code 0
```

Arithmetic operators

```
val num1 = 10
val num2 = 3

println(num1 + num2)
println(num1 - num2)
println(num1 * num2)
println(num1 / num2)
println(num1 % num2)
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
13
7
30
3
1

Process finished with exit code 0
```

Math

- To use math such as finding the square root we can use the *kotlin.math* package

```
val num1 = 14.2  
  
val result = kotlin.math.sqrt(num1)  
  
println(result)
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe"  
3.7682887362833544  
  
Process finished with exit code 0
```

Math (cont.)

- *kotlin.math* has all the mathematical operations commonly used

```
val num1 = 14.2

val result = kotlin.math.sqrt(num1)

println(kotlin.math.round(result))
println(kotlin.math.floor(result))
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
```

```
4.0
```

```
3.0
```

```
Process finished with exit code 0
```

If..else statement

- In this example we are using an *if..else statement* using a few comparison and logical operators

```
val num1 = 5
val num2 = 7
val num3 = 4

if (num1 < num2 || num3 >= num1){
    println("True")
} else {
    println("False")
}
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
True

Process finished with exit code 0
```


If expression

- Here we are turning that same statement into an expression by providing a **return value**

```
val num1 = 5
val num2 = 7
val num3 = 4

val value = if (num1 < num2 || num3 >= num1 ){
    "True"
} else {
    "False"
}

println(value)
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
True

Process finished with exit code 0
```

If expression (cont.)

- Whatever **comes last** in the expression is what is returned, so **any other logic can go beforehand**

```
val num1 = 5
val num2 = 2
val num3 = 6

val value = if (num1 >= num2 && num3 < num1 ){
    println("Sponge")
    "True"
} else {
    println("Star")
    "False"
}

println(value)
```

```
"C:\Program Files\Android\Android Studio\jbr\b
Star
False

Process finished with exit code 0
```

If expression (cont.)

- When the if expression only needs one line of code for each block, you don't need brackets which has a slightly cleaner look

```
val num1 = 5
val num2 = 2
val num3 = 6

val value = if (num1 >= num2 && num3 < num1 )
    "Sponge"
else if (num2 < 4)
    "Star"
else
    "Squirrel"

println(value)
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Star

Process finished with exit code 0
```

If expression (cont.)

- To give the previous code an even cleaner look you can put the whole expression on one line of code

```
val num1 = 5
val num2 = 2
val num3 = 6

val value = if (num1 >= num2 && num3 < num1) "Sponge" else if (num2 < 4) "Star" else "Squirrel"

println(value)
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\java.exe" ...
Star

Process finished with exit code 0
```

When statement

- Kotlin's version of the switch statement

```
val species = "W"  
  
when (species) {  
    "C" -> println("Crab")  
    "S" -> println("Squirrel")  
    "W" -> println("Whale")  
    else -> println("Human")  
}
```

```
"C:\Program Files\Android\Android Studio\jbr\bin\jav  
Whale  
  
Process finished with exit code 0
```

When expression

- The When statement can also be an expression

```
val species = "S"  
  
val result = when (species) {  
    "C" -> "Crab"  
    "S" -> "Squirrel"  
    "W" -> "Whale"  
    else -> "Human"  
}  
  
println(result)
```

```
"C:\Program Files\Android\Android Studio\  
Squirrel  
  
Process finished with exit code 0
```

When expression (cont.)

- You can also check *range* using the when statement/expression

```
val height = 165

val result = when (height){
    in 120 ≤ .. ≤ 150 -> "short"
    in 151 ≤ .. ≤ 180 -> {"average"}
    in 181 ≤ .. ≤ 210 -> "tall"
    else -> {"unknown"}
}

println(result)
```

```
"C:\Program Files\Android\Android Studio\jbr\t
average

Process finished with exit code 0
```

When expression (cont.)

- The when expression **doesn't need an initial value**

```
fun main() {  
  
    val num1 = 4  
    val num2 = 5  
  
    val value = when{  
        num1 > num2 -> "num1 is greater than num2"  
        num1 < num2 -> "num1 is less than num2"  
        else -> "num1 is equal to num2"  
    }  
  
    println(value)  
}
```


Booleans

- Nullable Booleans are either true, false or null
- When working with them you can't do this

```
val isSponge:Boolean? = null

val str = if (isSponge) "sponge is true" else "sponge is false"

println(str)
```

Booleans (cont.)

- You have to specifically check it's value

```
val isSponge:Boolean? = null

val str = if (isSponge == true) "sponge is true" else "sponge is false or null"

println(str)
```

```
"C:\Program Files\Android\Android Studio\jbr\b
sponge is false or null

Process finished with exit code 0
```

