Lecture 1

COMP 3717- Mobile Dev with Android Tech

第1讲

COMP 3717 - 使用Android技术进行移动开发

Welcome Everyone

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

欢迎各位

- 讲师：Charles Tapp
  - 办公时间：需预约
  - 联系方式： ctapp2@bcit.ca
  - Slack
    - 相关信息已发布在学习中心

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)

讲座与实验

- 讲座幻灯片将在课前打印并发布在D2L（学习中心）上课前
- 实验作业将在讲座结束后立即发布，并于下次讲座前一晚截止提交（为期一周）

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.

评分结构

- 细分：
  - 作业：10%
  - 实验：30%
  - 期中考试：30%
  - 期末考试：35%

*期中和期末考试的平均分必须达到50%或以上才能通过课程

- 示例：如果你的期末考试得49%，期中考试得49%，即使作业得到100%，你仍将无法通过该课程。

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

出勤

- 讲座与实验课
  - 我们将检查参与情况和出勤情况，必须亲自参加是必需的

- 如果因正当理由将缺席某节课或实验课，请告知我

Course Materials

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

课程资料

- Android Studio（必需）
  - 最新稳定版本
    - 独角鲸 2025.1.2
- 物理安卓设备（推荐）
  - 非必需，但如果您有的话建议使用

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

什么是安卓?

- 安卓是全球最流行的操作系统
  - 基于 Linux
- 最初主要为触摸屏移动设备设计
- 安卓每年收入大约为 200 到 300 亿美元，其中大部分来自应用商店

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

什么是 Android? （续）

- SDK（软件开发工具包）
  - 最新版本：Android 16（API 级别 36）
    - https://developer.android.com/about/versions/16
  - SDK 工具
    - 构建工具
    - 平台工具
    - 模拟器
    - Google Play 服务
- Gradle 是 Android Studio 中使用的构建软件
  - 在创建构建时为我们自动执行并管理构建过程

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)

Android 操作系统无处不在

- 智能手机
- 平板电脑
- 智能手表（Wear OS）
- 电视（Android TV，Google TV）
- Android Automotive（不同于 Android Auto）
- Amazon Fire OS（电视、平板电脑）
- 数码相机
- 冰箱
- 游戏机（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

语言及其他 SDK

- 原生开发
  - 使用 Android Studio 的 Kotlin 和 Java
- 可用来构建 Android 应用的跨平台工具
  - Jetpack Compose（Google）
  - Flutter（Google）
  - React (Meta)
  - Xamerin (Microsoft
  - Unity
  - 等等

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

- Google 推荐的 Android 应用开发语言
- 结合了面向对象和函数式编程特性
  - 函数式
    - 通过创建函数来转换数据
  - 面向对象编程（OOP）
    - 数据存储在对象中

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 与 Java

- 比 Java 更简洁和高效
- 可在任何使用 Java 的场景中使用
  - 例如服务器端代码
- 大多数从 Java 转向 Kotlin 的人，都不会再回头

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 基础

- 所有程序都需要一个 入口点 来运行，也就是主函数

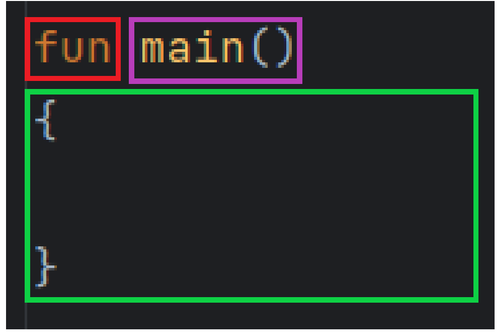
```
fun main(){

}
```

- 在为 Android 构建时，我们不会直接使用 main 函数

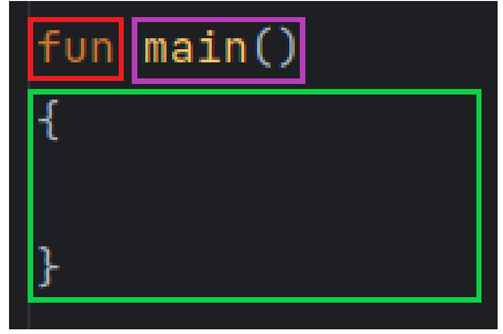
Kotlin basics (cont.)

- 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



Kotlin 基础（续）

- fun 是用于创建函数的关键字
- main 是我们函数的名称
  - 我们按照 Kotlin 语法提供括号 ( )
- 花括号内的部分称为 代码块，我们的逻辑就放在其中



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

Kotlin 基础（续）

- *println* 是我们在主函数内调用的一个函数

- 这里我们传入了一个 字符串 但我们也可以传入任意类型，因为 *println* 是一个重载函数

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

- *println* 向控制台输出一行并换行

Variables

- We can create a variable like so

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

- variable name
- variable type
- variable value

变量

- 我们可以这样创建一个变量

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

- 变量名
- 变量类型
- 变量值

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
```

变量（续）

- 在这里，我们可以使用 *println* 将一些变量输出到控制台

```
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'

}
```

基本数据类型

- 整型、双精度浮点数、长整型、单精度浮点数、布尔值、字符串和字符

```
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 it's use cases, but it is best to use a specific type (i.e. String) for now

基本数据类型（续）

- 每个变量类型都可以是 *Any* 类型，因为每个 Kotlin 类都以 *Any* 作为超类

```
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* 有自己的使用场景，但目前最好使用特定类型（例如 String）

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'
```

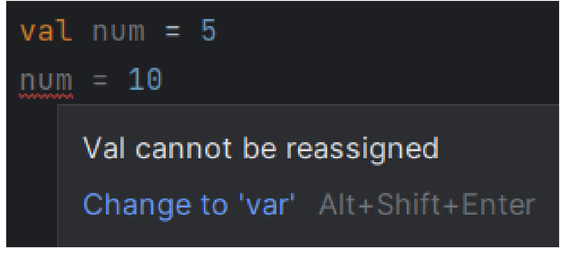
类型推断

- Kotlin 在我们初始化变量时可以为我们确定类型

```
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

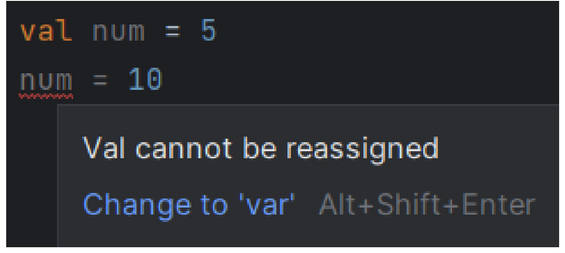
- Val
  - Read only
  - Can only be assigned once
- Var
  - Mutable
  - Can be reassigned



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

Val 与 Var

- Val
  - 只读
  - 只能赋值一次
- Var
  - 可变
  - 可以重新分配



- 除非需要使用 *var*，否则最好在任何地方都使用 *val*

Const

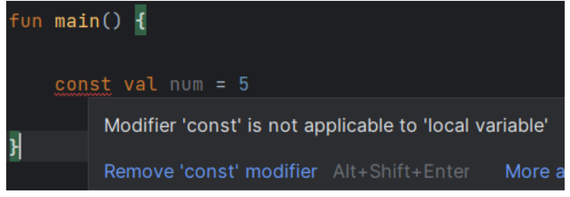
- Const values are read only variables known before code execution
  - Better for performance
- Since they are determined at compile time you can only define primitive types and strings as const
- Anything besides primitives and strings may have runtime side effects contradicting how constants work

常量

- 常量是代码执行前已知的只读变量
  - 性能更优
- 由于它们在编译时确定，因此只能将基本类型和字符串定义为常量
- 除了基本类型和字符串之外的任何内容都可能存在运行时副作用，这与常量的工作方式相矛盾

Const (cont.)

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



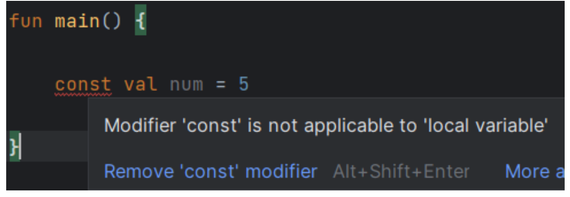

```
const val num = 5

fun main() {

}
```

常量（续）

- 常量变量不能在函数内部声明
  - 类级别的变量




```
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"
```

空安全

- 空安全特性的设计初衷是为了避免空引用
  - 过去常见的错误！
- Kotlin 允许类型可为空或不可为空
- 可为空

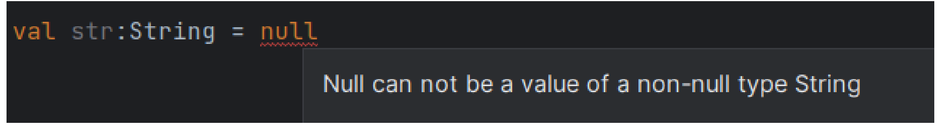
```
val str:String? = null
```

- 不可为空

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

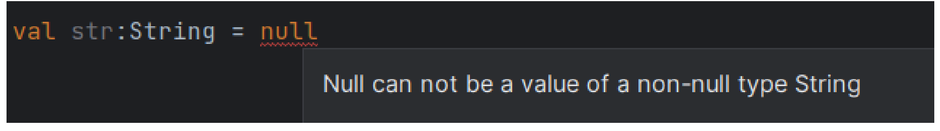
Null Safety (cont.)

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



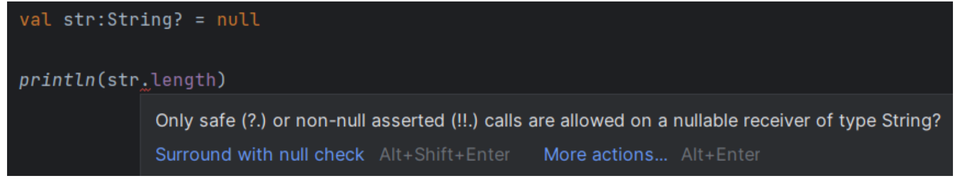
空安全（续）

- 你可以看到，我们无法将 null 赋值给非 null 类型



Null Safety (cont.)

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



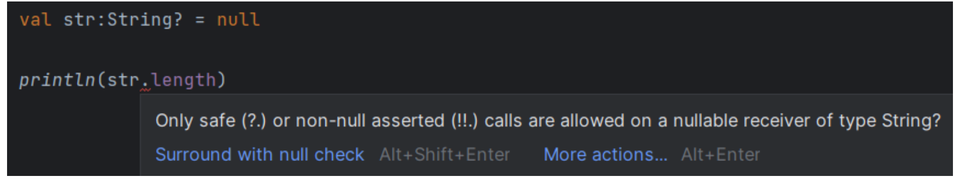
- We have to use certain operators to protect us from a null exception

```
val str:String? = null

println(str?.length)
```

空安全（续）

- 当我们尝试使用可空类型的属性（*length*）时，会得到一个错误



- 我们必须使用某些操作符来防止空异常

```
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

空安全（续）

```
val str:String? = null

println(str?.length)
```

- 运行此代码将在控制台打印 null，而不是抛出空指针异常

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

Process finished with exit code 0
```

- 这就是 Kotlin 如何保护我们免受空指针异常的方式

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
}
```

空安全（续）

- !! 操作符
  - 仅在确定不会为空时使用

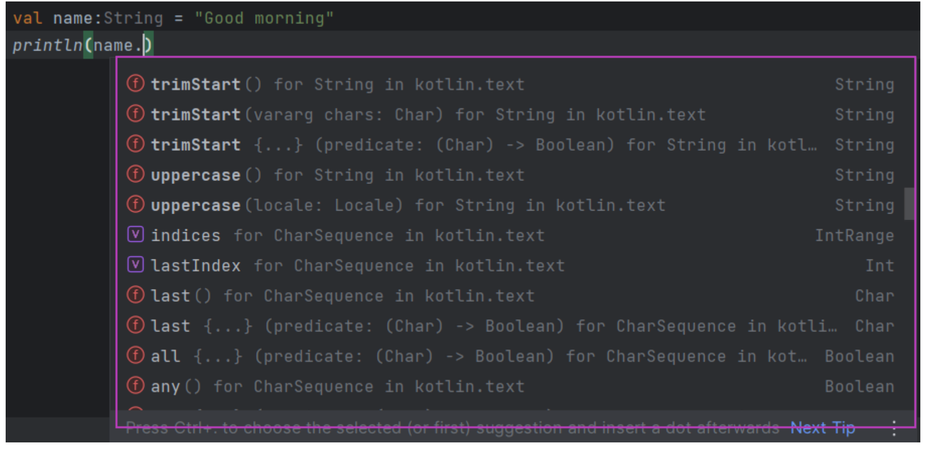

```
private var str:String? = "Hello World"

private fun doSomething() : Int{

    return str!!.length
}
```

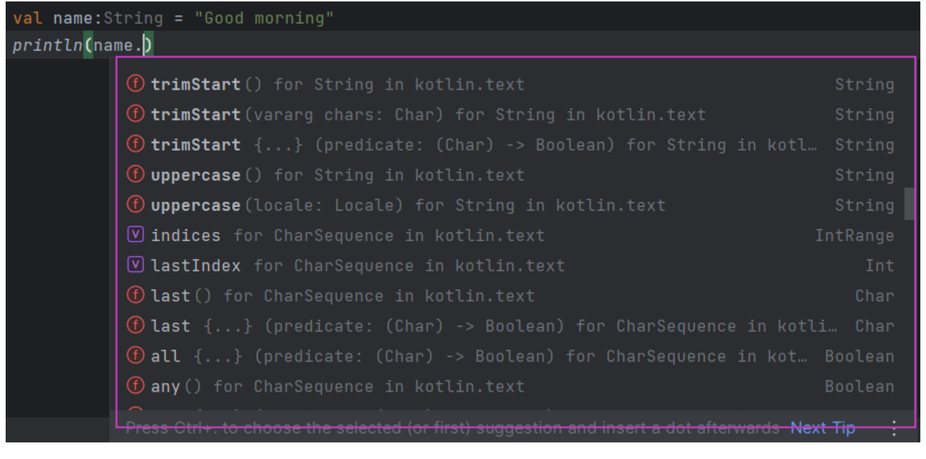
Strings

- Strings have many properties and methods we can work with



字符串

- 字符串具有许多 属性 和 方法 可供我们操作



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
```

字符串（续）

- 以下是一些常见示例

```
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
```

字符串（续）

- 我们还可以使用 [ ] 方括号 打印字符串中字符的索引

```
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 Create breakpoint : 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
```

字符串（续）

- 在 Kotlin 中仍然可能出现越界错误，但编译器会尽量提前给你警告

```
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 Create breakpoint : 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
```

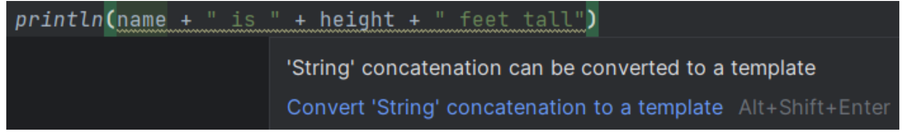
String Template

- 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



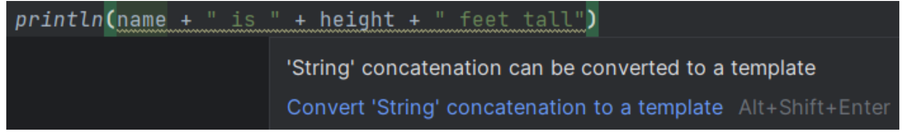
字符串模板

- 如果要在字符串中连接两个变量，请不要这样做

```
val name = "Jerry"
val height = 5

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

- 编译器会要求你将其转换为模板



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
```

字符串模板（续）

- 要使用字符串模板，需将表达式按如下方式包裹：***${expression}***
- 如果只有一个变量，可以像下面这样省略**{ }**花括号

```
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
```

字符串（续）

- 在连接变量时，你也可以使用 format 方法使用*%s*操作符

```
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()
```

多行字符串

- 按下 **shift + "** 三次，然后按 **enter**，即可创建一个多行字符串

```
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
```

多行字符串（续）

- 在这里我们可以创建一个多行字符串，甚至包含缩进

```
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\bi

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


Process finished with exit code 0
```

多行字符串（续）

- *trimIndent()* 并非必需，但你可能为了格式化效果而更倾向于使用它
  - 它会从每一行中移除共同的最小缩进
  - 如果首行和末行为空行，则将其移除

- 以下是不使用 *trimIndent()* 的相同示例

```
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\bi

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


Process finished with exit code 0
```


SEE THE TIME ON THE CLOCK?
IT'S BREAK TIME!


SEE THE TIME ON THE CLOCK?
IT'S BREAK TIME!

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
```

字符串比较

- 在 Kotlin 中，为了比较两个变量的值，我们使用 == 运算符
  - *.equals* 所做的事情相同

```
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
```

字符串引用相等性

- 要比较两个变量的内存位置，我们使用 ===

```
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

字符串常量池内存

- 请注意，当我们对两个相同值但不同的字符串使用=== 时，结果为真

```
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
```

- 这是由于字符串常量池内存的存在，它与堆内存不同

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

字符串常量池内存（续）

- 它会检查字符串常量池中是否已存在该字符串的值内存
- 如果存在，则指向内存中的该字符串
- 因此，在前面的示例中，堆上仅创建了一个字符串

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
```

字符串池内存（续）

- 在此示例中，*name2* 是在堆上创建的新对象，因此比较它们的内存位置将返回 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
```

算术运算符

```
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.e
3.7682887362833544

Process finished with exit code 0
```

Math

- 要使用诸如求平方根之类的数学运算，我们可以使用*kotlin.math*包

```
val num1 = 14.2

val result = kotlin.math.sqrt(num1)

println(result)
```

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

数学（续）

- *kotlin.math* 包含了所有常用的数学运算

```
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
```

如果..否则语句

- 在此示例中，我们使用了一个if..else 语句，并使用了一些比较和逻辑运算符

```
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
```

如果表达式

- 在这里，我们通过提供一个 返回值 将相同的语句转换为表达式


```
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
```

如果表达式（续）

- 无论什么 最后出现的内容 就是表达式的返回值，因此 可以 先执行其他逻辑可以事先进行

```
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
```

如果表达式（续）

- 当 if 表达式的每个代码块只需要一行代码时，你不需要括号，这样看起来更简洁一些

```
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
```

表达式（续）

- 为了使前面的代码看起来更加简洁，你可以将整个表达式放在一行代码中

```
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 语句

- Kotlin 中的 switch 语句版本


```
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
```

当表达式

- When 语句也可以作为表达式

```
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\b
average

Process finished with exit code 0
```

表达式（续）

- 你还可以使用 when 语句/表达式来检查 *range*

```
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\b
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)
```

表达式（续）

- when 表达式 不需要初始值

```
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)
```

布尔值

- 可空布尔值可以是 true、false 或 null
- 使用它们时，不能这样做

```
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
```

布尔值（续）

- 你必须明确检查它的值

```
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
```