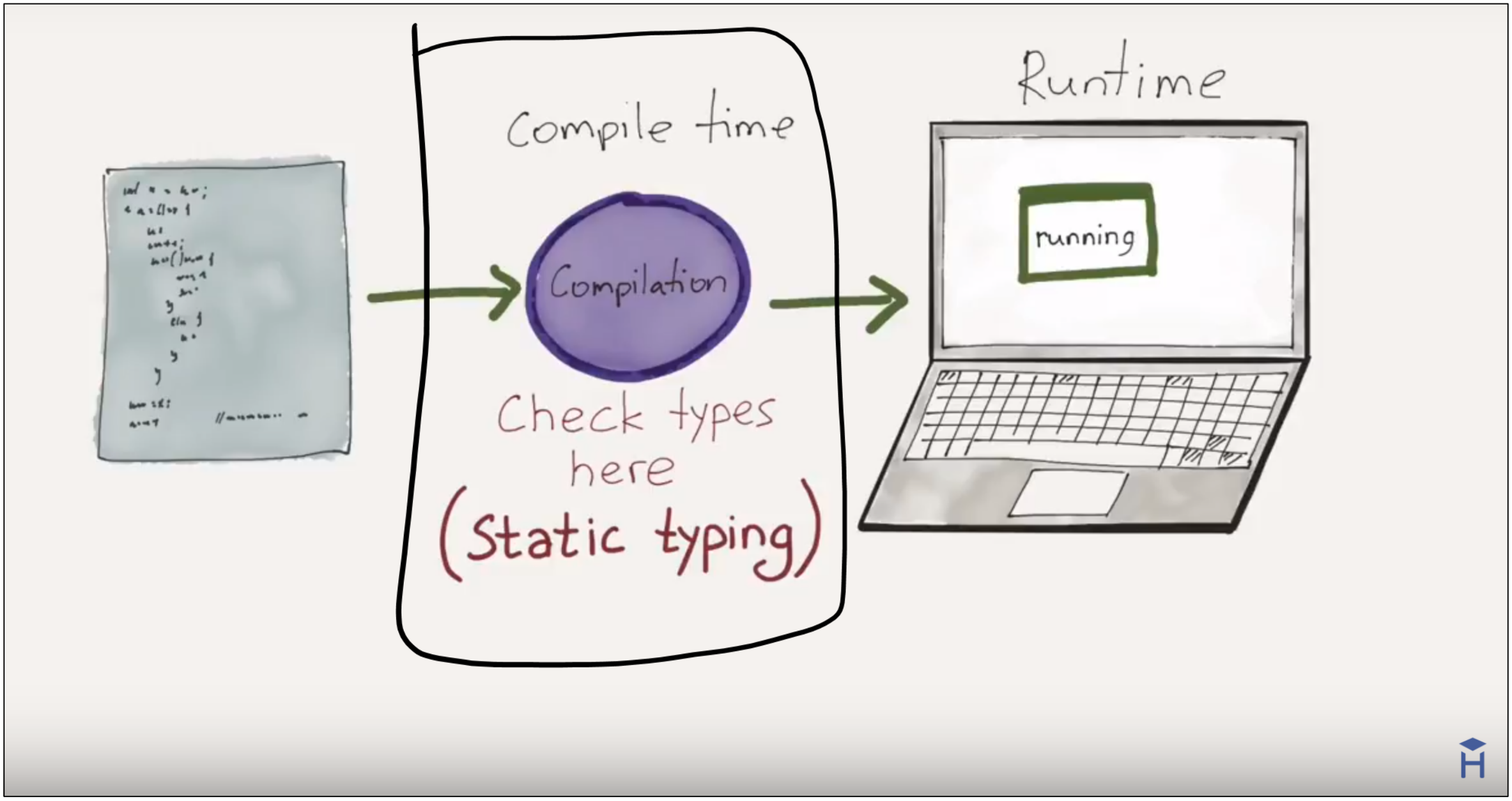
Kotlin: Basic Types



STATIC TYPING

"Variable declarations are mandatory before usage, else results in a compile-time error"

Static Typing – Example


```
String greeting = "Hello!";
int someRandomInteger = 100;
double aDoubleVariable = 2.2;
```

A type is assigned to each variable.

In Java, if we don't assign a type, we get a compiler error
→ Java is statically typed.

Types determine the operations we can perform on the variables.

https://howtoprogramwithjava.com/dynamic-typing-vs-static-typing/

Static Typing – Example

Kotlin

*In Kotlin, you don't have to specify the type of each variable explicitly, even though Kotlin **is** statically-typed.*

Here, Kotlin determines the type from the initialisation.

```
fun main(args : Array<String>)
{
    var someRandomInteger = 100
    var aDoubleVariable = 2.2
    println (someRandomInteger)
    println (aDoubleVariable)
}
```

http://petersommerhoff.com/dev/kotlin/kotlin-for-java-devs/#type-inference

Static Typing – Example

Kotlin

However, you can choose to explicitly define a data type.

```
fun main(args : Array<String>)
{
    var someRandomInteger : Int = 100
    var aDoubleVariable : Double = 2.2
    println (someRandomInteger)
    println (aDoubleVariable)
}
```

Static Typing – Example

With Kotlin, you have to <u>either</u> define a type or initialise the variable (kotlin then determines the type!).

```
fun main(args : Array<String>)
{
    var someRandomInteger //compile error
    var aDoubleVariable : Double = 2.2
    println (someRandomInteger)
    println (aDoubleVariable)
}
```

http://petersommerhoff.com/dev/kotlin/kotlin-for-java-devs/#type-inference

Static Typing – Example

Kotlin

```
fun main(args : Array<String>)
{
    var someRandomInteger : Int = 100
    var aDoubleVariable : Double = 2.2

    someRandomInteger = 2.65      //compile error
    aDoubleVariable = 233         //compile error

    println (someRandomInteger)
    println (aDoubleVariable)
}
```

http://petersommerhoff.com/dev/kotlin/kotlin-for-java-devs/#type-inference

Kotlin

- runs on Java Virtual Machine.
- is an evolution of the Java syntax but is more concise and has cleaner syntax.
- is not syntax compatible with Java; but is interoperable with Java.
- relies on some Java Class Libraries e.g. Collections framework.
- is a statically-typed programming language.

Based on : https://en.wikipedia.org/wiki/Kotlin_(programming_language)

Basic Types

Numbers, characters and booleans.

Basic Types

In Kotlin, everything is an object in the sense that we can call member functions and properties on any variable.

Basic Types - Numbers

Type	Bit width
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

Basic Types - Numbers

Type	Bit width
Double	64
Float	32
Long	64
Int	32
Short	16
Byte	8

```
val doubleNumber: Double = 100.45
val floatNumber: Float = 100.45f
val longNumber: Long = 100
val intNumber: Int = 100
val shortNumber: Short = 100
val byteNumber: Byte = 100
```

Explicitly defining a numeric type

Basic Types - Numbers

Type inference

```
val doubleNumber = 100.45
val floatNumber = 100.45f
val longNumber = 100L
val intNumber = 100
val shortNumber = 100
val byteNumber = 100
```

Basic Types - Numbers

Type inference

```
val doubleNumber = 100.45
val floatNumber = 100.45f
val longNumber = 100L
val intNumber = 100
val shortNumber = 100
val byteNumber = 100
```

```
println("doubleNumber type is: " + doubleNumber.javaClass)
println("floatNumber type is:  " + floatNumber.javaClass)
println("longNumber type is:   " + longNumber.javaClass)
println("intNumber type is:    " + intNumber.javaClass)
println("shortNumber type is:  " + shortNumber.javaClass)
println("byteNumber type is:   " + byteNumber.javaClass)
```

Console

```
<terminated> Config - Main.kt [Java Appl
doubleNumber type is: double
floatNumber type is:  float
longNumber type is:   long
intNumber type is:    int
shortNumber type is:  int
byteNumber type is:   int
```

Basic Types - Numbers

```
val oneMillion = 1_000_000
val threeThousand = 3_000
val creditCardNumber = 1234_4321_5678_8765

fun main(args : Array<String>)
{
  println("" + oneMillion + " - the type is: " + oneMillion.javaClass)
  println("" + threeThousand + " - the type is: " + threeThousand.javaClass)
  println("" + creditCardNumber + " - the type is: " + creditCardNumber.javaClass)
}
```

You can use underscores to make number constants more readable.

```
Console
<terminated> Config - Main.kt [Java Application] C:\Progra
1000000 - the type is: int
3000 - the type is: int
1234432156788765 - the type is: long
```

Basic Types – Numbers: Explicit Conversions

In Kotlin, there are no implicit widening conversions for numbers i.e. smaller types (e.g. Byte) are not subtypes of bigger ones (e.g. Int)

→ smaller types are NOT implicitly converted to bigger types.

Basic Types – Numbers: Explicit Conversions

In Kotlin, there are no implicit widening conversions for numbers i.e. smaller types (e.g. Byte) are not subtypes of bigger ones (e.g. Int)

→ smaller types are NOT implicitly converted to bigger types.

```
val byteNumber: Byte = 10          //static type check: OK
val intNumber: Int = byteNumber //syntax error
```

BUT, we can use explicit conversions to widen numbers

```
val byteNumber: Byte = 10          //static type check: OK
val intNumber: Int = byteNumber.toInt()  //OK
```

Basic Types – Numbers: Explicit Conversions

Every number type supports the following conversions:

- `toByte(): Byte`
- `toShort(): Short`
- `toInt(): Int`
- `toLong(): Long`
- `toFloat(): Float`
- `toDouble(): Double`
- `toChar(): Char`

```
//Explicit Conversion
val intNumber: Int = byteNumber.toInt()
val floatNumber: Float = byteNumber.toFloat()
```

Basic Types – Characters

```
val aChar = 'a'
val bChar: Char = 'b'

fun main(args : Array<String>)
{
  println("" + aChar + " - the type is: " + aChar.javaClass)
  println("" + bChar + " - the type is: " + bChar.javaClass)
}
```

Console

```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\jre1.8.0
a - the type is: char
b - the type is: char
```

Basic Types – Booleans

```
val aFlag = true
val bFlag: Boolean = false

fun main(args : Array<String>)
{
  println("" + aFlag + " - the type is: " + aFlag.javaClass)
  println("" + bFlag + " - the type is: " + bFlag.javaClass)
}
```

Console

```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\jre1.8.0_77\bin\
true - the type is: boolean
false - the type is: boolean
```

Basic Types – Escape Characters

Special characters can be escaped using a backslash:

\t \b \n \r \’ \“ \\ \$

```
val aFlag= true
val bFlag: Boolean = false

fun main(args : Array<String>)
{
  println("" + aFlag + " - the type is: \n\t\t" + aFlag.javaClass)
  println("" + bFlag + " - the type is: \n\t\t" + bFlag.javaClass)
}
```

Console

```
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\jre1.8.0
true - the type is:
                boolean
false - the type is:
                boolean
```

Local Variables – val (read-only)

Assign-once (read-only) local variable:

```
val a: Int = 1  // immediate assignment
val b = 2       // `Int` type is inferred
val c: Int      // Type required when no initializer is provided
c = 3           // deferred assignment
```

```
a = 1, b = 2, c = 3
```

Target platform: JVM Running on kotlin v. 1.1.50

Local Variables – val (read-only)

Assign-once (read-only) local variable:

```
val a: Int = 1  // immediate assignment
val b = 2       // `Int` type is inferred
val c: Int      // Type required when no initializer is provided
c = 3           // deferred assignment
c = 4           // Syntax error: cannot reassign value
```

Target platform: JVM Running on kotlin v. 1.1.50

Local Variables – var (mutable)

Mutable variable:

```
var x = 5 // `Int` type is inferred
x += 1    // Mathematical operations permitted on Int
```

```
x = 6
```

Target platform: JVM Running on kotlin v. 1.1.50

Local Variables – var (mutable)

Mutable variable:

```
var x = 5 // `Int` type is inferred
x += 1    // Mathematical operations permitted on Int
x = 5     // Allowed to reassign value
```

```
x = 5
```

Target platform: JVM Running on kotlin v. 1.1.50