



From Python to Scala: A tour on Static Typing

Nancy Hitschfeld
Matías Toro

Computing hailstones

Starting with a number n , the next number in the sequence is $n/2$ if n is even, or $3n+1$ if n is odd. The sequence ends when it reaches 1. Here are some examples:

2, 1

3, 10, 5, 16, 8, 4, 2, 1

4, 2, 1

$2n, 2n-1, \dots, 4, 2, 1$

5, 16, 8, 4, 2, 1

7, 22, 11, 34, 17, 52, 26, 13, 40, ...? (where does this stop?)

Termination is still an
open question

Variable declaration

Computing hailstones

Scala

```
var n: Int = 3
while(n != 1){
  println(n)
  if(n % 2 == 0)
    n = n/2
  else
    n = 3 * n + 1
}
println(n)
```

Python

```
n = 3
while n != 1:
    print(n)
    if n % 2 == 0:
        n = n / 2
    else:
        n = 3 * n + 1
```

Similar **basic** semantics

Scala requires
parentheses around
the conditions of the
if and **while**

Scala (<3) uses curly
braces around blocks,
instead of
indentation.

Types

Scala

```
var n: Int = 3  
...
```

Python

```
n = 3  
...
```

The most important semantic difference between the Python and Scala code is the declaration of the variable `n`, which specifies its type: `Int`.

“A type is any property of a program that we can establish without executing the program.”

[Krishnamurthi]

“Abstractions that represent sets of values, and the operations and relations applicable to them.”

[Bruce]

Some Types

A **type approximates** the set of values that an expression reduces to.

Int (for integers like 5 and -200. 32 bit signed value. Range -2147483648 to 2147483647)

Long (64 bit signed value. Range -9223372036854775808 to 9223372036854775807)

Boolean (for true or false)

Double (for floating-point numbers. 64 bit IEEE 754 double-precision float)

Char (for single characters like 'A' and '\$'.)

String represents a sequence of characters, like a Python string.

Unit represents the absence of a meaningful value.

Functions are also typed

We can annotate the type of the arguments and the return type:

```
def equal(a: Int, b: Int): Boolean = {  
    a == b  
}
```

Function takes two integer as arguments...

... and returns a boolean

The last expression is always returned

We say that the `equal` function has type
`(Int, Int) => Boolean`

What is the type of a hailstone function?


Why Types?

Type checking

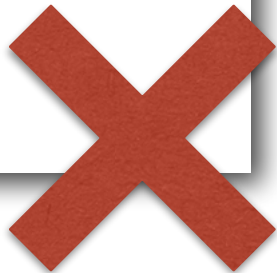
The process of verifying and enforcing the constraints of types.

```
def equal(a: Int, b: Int): Boolean = {  
    a == b  
}
```

```
var n: Int = 3  
while(!equal(n, 1)) {  
    ...  
}
```



```
var n: Int = 3  
while(!equal(n, "hola")) {  
    ...  
}
```



Static Checking vs Dynamic Checking

Static checking: “the bug” is found automatically before the program even runs.

Dynamic checking: “the bug” is found automatically when the code is executed.

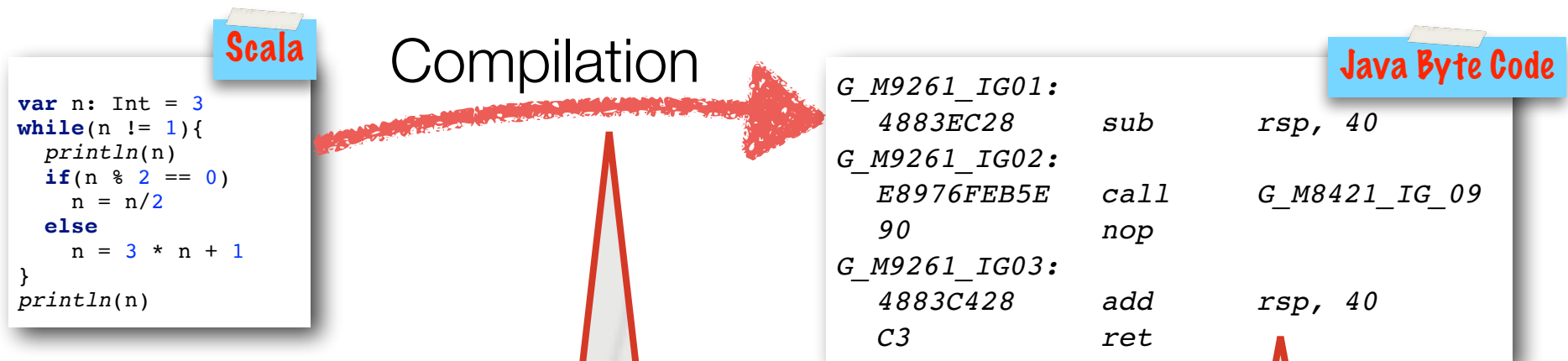
No checking: the language doesn’t help you find the error at all. You have to watch for it yourself, or end up with wrong answers.

Catching a bug statically is “better” than dynamically.

Catching it dynamically is better than not catching it at all.

What is a compiler?

A compiler is a program that translates computer code written in one programming language (**source**) to another language (**target**).



A good place ***type check!***

This is what
actually "runs"

Static Typing vs Dynamic Typing

Statically-typed languages

Scala

Java

C++

C#

Require variable/signatures to be annotated with type information

Checking of types is done at **compile time** (before running the program)

No type errors during runtime.

"5" * "6"

Fails during compilation

Dynamically-typed languages

Python

Javascript

Ruby

Groovy

No type annotation on variables/signatures

Checks of *safety* are deferred until **runtime** (while the program is running)

Runtime errors

"5" * "6"

Fails at runtime

Static Typing vs Dynamic Typing

Statically-typed languages

Scala

Java

C++

C#

- + early error detection
- + form of documentation
- + enable certain optimizations
- + more efficient
- necessarily conservative
- expressiveness \implies complexity

Dynamically-typed languages

Python

Javascript

Ruby

Groovy

- + simple
- + all programs can be run
- + incremental/agile
- runtime errors
- lack of documentation
- less efficient

```
if(true) 5*6 else "5"*"6"
```

Exercises

```
var n: Int = 5
if (n) {
  n = n + 1
}
```

- a) static error
- b) dynamic error
- c) no error

```
val big: Double = 1/5
```

- a) static error
- b) dynamic error
- c) no error, wrong answer

Exercises

```
val sum: Int = 0  
val n: Int = 0  
val average: Int = sum / n
```

- a) static error
- b) dynamic error
- c) no error

```
val sum: Double = 7  
val n: Double = 0  
val average: Double = sum / n
```

- a) static error
- b) dynamic error
- c) no error, wrong answer

Type Inference

Although you can declare a type when creating a new variable:

```
val x: Int = 1  
val y: Double = 1
```

In Scala you generally don't have to declare the type when defining value binders:

```
val a = 1  
val b = List(1, 2, 3)  
val m = Map(1 -> "one", 2 -> "two")
```

Mutability vs immutability

Good programmers try to avoid *unexpected change*.

Immutability: intentionally forbidding certain things from changing at runtime.

Benefits:

Thread safety

Atomicity of failure

Absence of hidden side-effects

Protection against null reference errors

Ease of caching

Prevention of identity mutation

Avoidance of temporal coupling
between methods

Support for referential transparency

Protection from instantiating logically-
invalid objects

Protection from inadvertent corruption
of existing objects

Like **final**
in Java

Mutability vs immutability

In Scala you define immutable variables with the **val** keyword, and mutable variables with **var**.

```
var x = 1  
val y = 1  
x = x + 1  
y = y + 1
```

Static or
dynamic
error?

Mutability vs immutability

In Scala you define immutable variables with the **val** keyword, and mutable variables with **var**.

```
var n: Int = 3
while(n != 1) {
  println(n)
  if(n % 2 == 0)
    n = n / 2
  else
    n = 3 * n + 1
}
println(n)
```

```
def hailstone(n: Int): Unit = {
  println(n)
  if (n % 2 == 0)
    hailstone(n / 2)
  else if (n != 1)
    hailstone(3 * n + 1)
}
hailstone(3)
```

Immutable version

Hacking vs engineering

Hacking:

Writing lots of code before testing any of it.

Keeping all the details in your head, assuming you'll remember them forever, instead of writing them down in your code.

Assuming that bugs will be nonexistent or else easy to find and fix.

Engineering:

Write a little bit at a time, testing as you go (test-first programming).

Document the assumptions that your code depends on.

Defend your code against stupidity – especially your own! Static checking helps with that.

The goal of this course

Learn to write software that is:

Safe from bugs.

Easy to understand.

Ready for change.

Why Scala?

Why Scala?

Scala stands for scalable language.

from small scripts to large systems

Easy to start with Scala

seamless interop with all Java libraries

The fusion of OOP and FP in a statically-typed language

FP: build things quickly from simple parts

OOP: structure larger systems, adapt to new demands

Result

new kinds of programming patterns and component abstractions

legible and concise programming style

This course will
focus on OOP

Create a “Hello World” project with sbt

Go to an empty folder and run the command

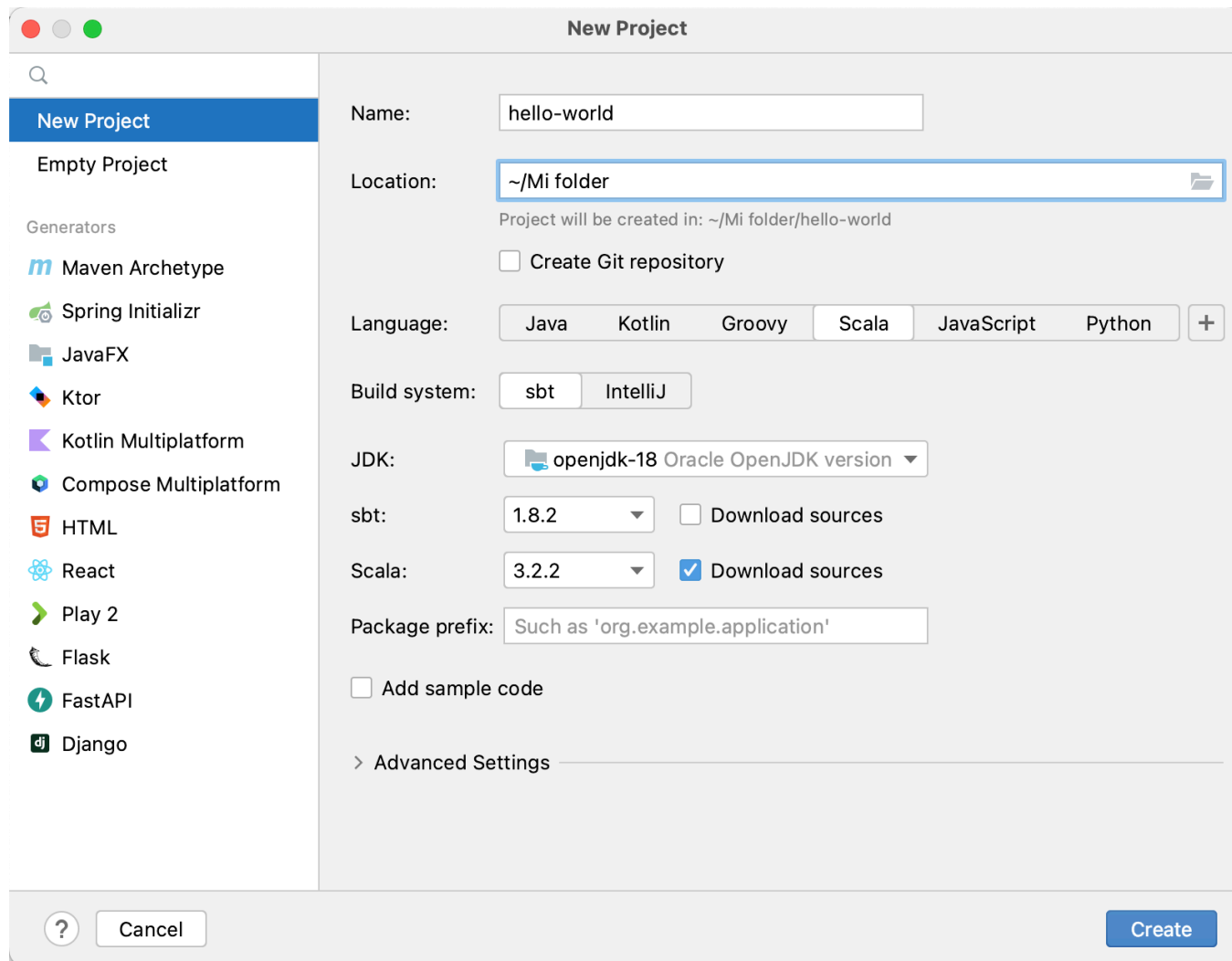
```
sbt new scala/hello-world.g8
```

When prompted, name the application `hello-world`.
This is the structure of what just got generated:

```
hello-world
- project (sbt uses this for its own files)
  - build.properties
- build.sbt (sbt's build definition file)
- src
  - main
    - scala (all of your Scala code goes here)
      - Main.scala (Entry point of program) <-- this is all we need for now
```

Alternatively create the project using IntelliJ

Click on File > New > Project



Main.scala (Scala 2)

Every method must be declared inside an *object* or *class* (more on this later)

The arguments passed by the command line

```
object Main {  
  def main(args: Array[String]) = {  
    println("Hello, World!")  
  }  
}
```

The return type is optional

Main.scala (Scala 3)

```
@main def hello: Unit =  
  println("Hello world!")  
  println(msg)  
  
def msg = "I was compiled by Scala 3. :)"
```

Compiling and running the project with sbt

Go to an empty folder and run the commands

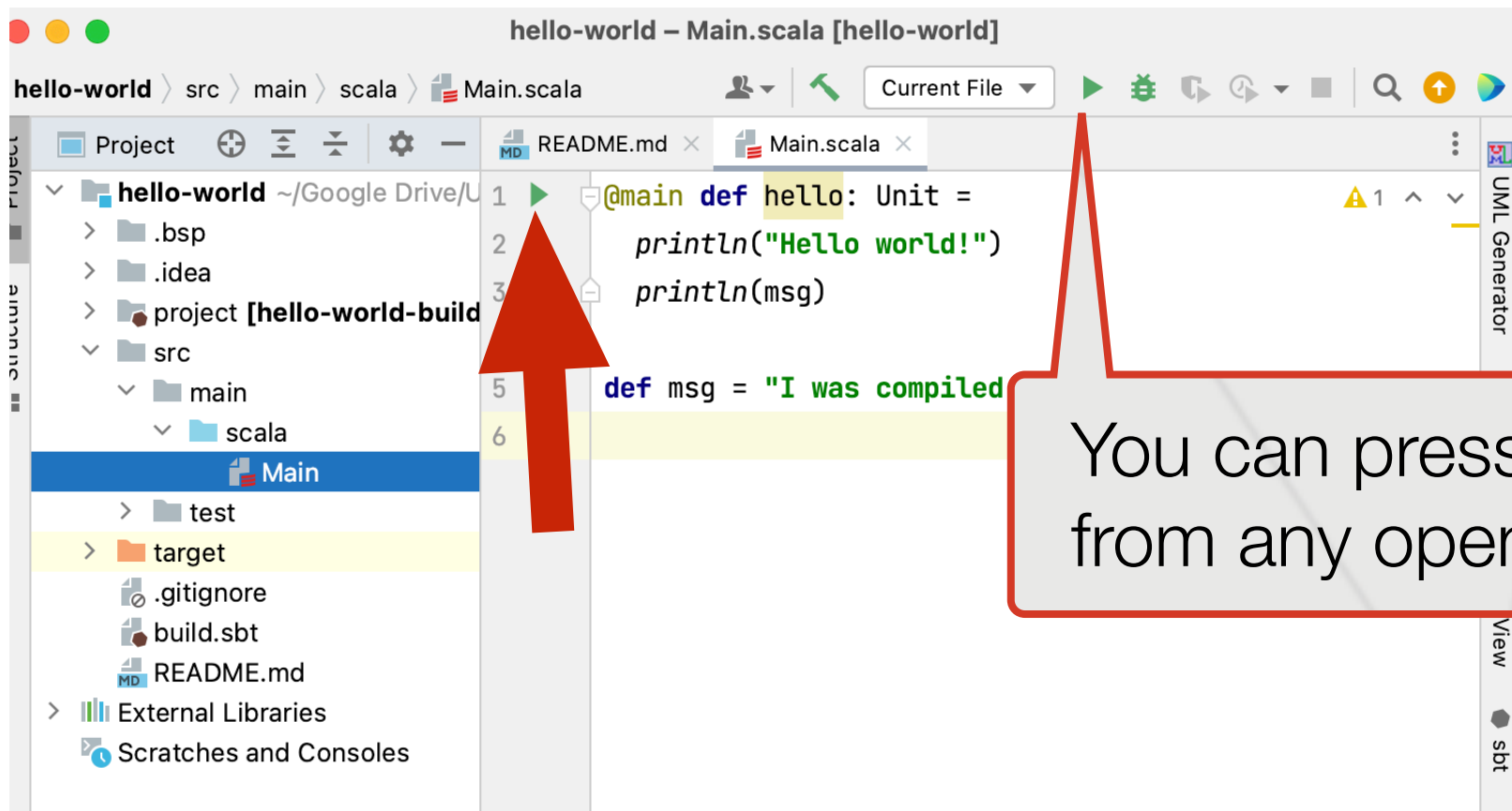
```
cd hello-world  
sbt
```

Inside the sbt console you can type the `~run` command to run the program (and re-run on every file save).

```
sbt:hello-world> ~run  
[info] compiling 1 Scala source to ../hello-world/target/scala-3.2.1/classes  
...  
[info] running hello  
Hello world!  
I was compiled by Scala 3. :)  
[success] Total time: 1 s, completed Jan 25, 2023, 10:39:37 AM
```

Compiling and running the project with IntelliJ

Open the Main.scala file and click the ▶ button





dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Ask for user input

```
import scala.io.StdIn.readLine

object helloInteractive {

  def main(args: Array[String]) = {
    println("Please enter your name:")
    val name = readLine()

    println("Hello, " + name + "!")
  }
}
```

The build.sbt file

This is where we configure the requirements of this project

The version of Scala

```
val scala3Version = "3.2.1"
```

```
lazy val root = project
```

Where are the source files

```
.in(file("."))
```

```
.settings(
```

The name of the project

```
  name := "hello-world",
```

```
  version := "0.1.0-SNAPSHOT",
```

The version

```
  scalaVersion := scala3Version,
```

```
  libraryDependencies += "org.scalameta" %% "munit" % "0.7.29" % Test
```

```
)
```

The dependencies

This is a library to write unit test

License



Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

You are free to:

- Share: copy and redistribute the material in any medium or format
- Adapt: remix, transform, and build upon the material for any purpose, even commercially

The licensor cannot revoke these freedoms as long as you follow the license terms



Attribution: you must give appropriate credit



ShareAlike: if you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original

Complete license: <https://creativecommons.org/licenses/by-sa/4.0/>



dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

f @ in / DCCUCHILE