

TYPES, VALUES, AND CONTROL FLOW

Benedict R. Gaster / @cuberoo_

TYPES

A type is a collection of related things, often called values

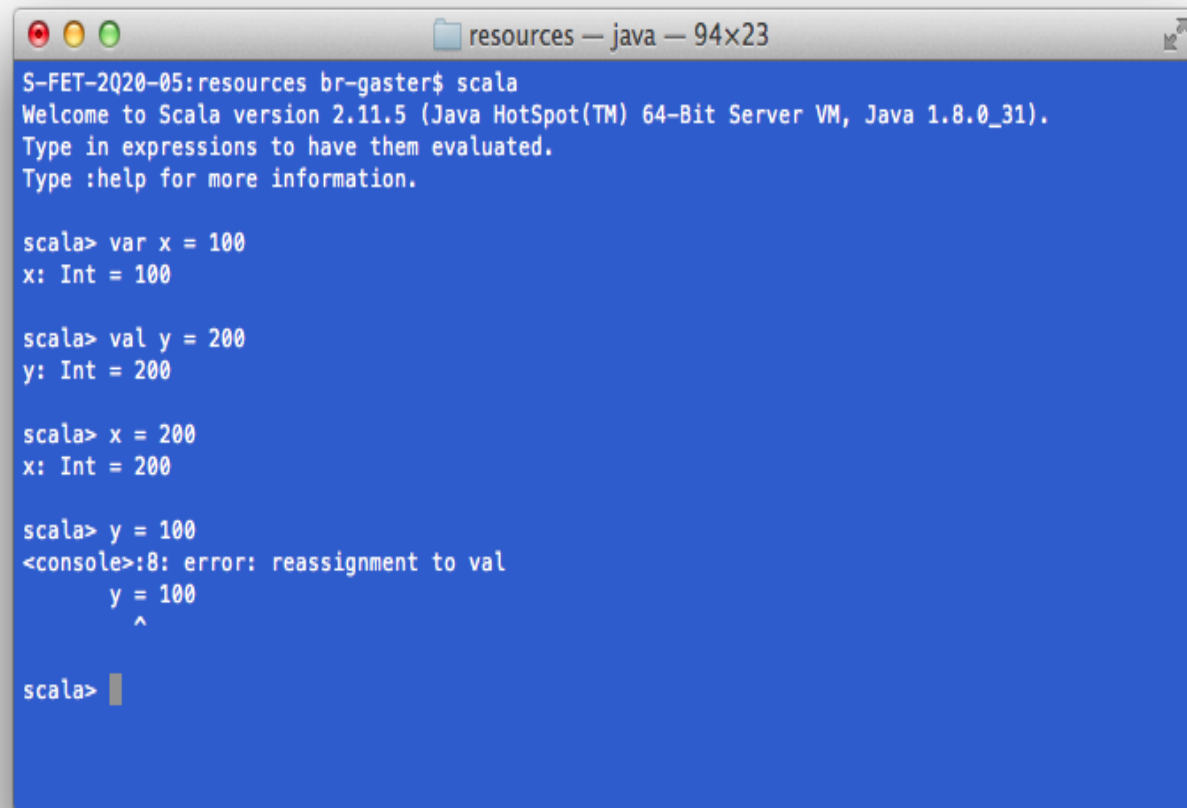
Name	Sets of values	Literal values
Char	characters	'Z', '#'
Int	integers	1, 2, 3
String	sequence of characters	"hello, world"
Double	floating point numbers	2.34, 0.22e23,

DEFINITIONS

- A variable is a name that refers to a thing
- A value is a name that refers to a thing too!
- The difference is that variables are mutable while values are immutable

```
var x = 100 // mutable, i.e. can change later  
val y = 100 // immutable, i.e. never changes
```

TRY IT OUT



```
resources — java — 94x23
S-FET-2Q20-05:resources br-gaster$ scala
Welcome to Scala version 2.11.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_31).
Type in expressions to have them evaluated.
Type :help for more information.

scala> var x = 100
x: Int = 100

scala> val y = 200
y: Int = 200

scala> x = 200
x: Int = 200

scala> y = 100
<console>:8: error: reassignment to val
      y = 100
       ^

scala>
```

TYPES AGAIN

- In Scala all values have a type
 - although you do not always need to write it down
- If an expression, **e**, evaluates to a value of type **T**, then **e** has type **T**, written **e : T**
- All correct (i.e. well-formed) expressions have a type, which can, often, be inferred by the compiler.

LISTS

A list is a sequence of things of the same type:

```
List(10,20,30,40,50,60,60) : List[Int]  
List("Hello", "World!") : List[String]
```

In general `List[T]` is a list of things of type `T`

MORE ON LIST TYPES

The type of a list says nothing about its length

```
List(10,20) : List[Int]  
List(20,30,40,50,60) : List[Int]
```

Both have type List[Int] but

```
List(10,20).length == 2  
List(20,30,40,50,60).length == 5
```

SOME LIST METHODS

Numerous predefined **methods**

What it is	What it does
<code>List() or Nil</code>	Empty List
<code>val cl = 'a' :: 'b' :: Nil</code>	Creates a new list with values 'a' and 'b'
<code>cl:::List('c','d')</code>	Concatenates two lists, giving List('a','b','c','d')
<code>cl.head</code>	Returns the first element of a list (in this case 'a')

TUPLES

A tuple is a sequence of things of the different type:

```
(10, 'a', "hello") : (Int, Char, String)
('1') : (Int)
("first", List(10, 20)) : (String, List[Int])
```

MORE ON TUPLES

The type of a tuple encodes its length

```
(10, 'a') : (Int, Char)  
( 'a', 'b' ) : (Char, Char)
```

have different types but have the same length

FUNCTIONS

In mathematics a function is a **relation** between a set of inputs and a set of valid outputs

You might remember some from school, e.g.

$$f(x) = x^2$$

defines a function from the set of integers to the set of integers, whose value is the square of its input

SCALA FUNCTIONS

In Scala a function is a **relation** between a tuple of inputs and a thing, which might be a tuple but can be other things too.

Returning to our function for squaring integers

```
def f(x : Int) : Int = {  
  x * x  
}
```

SCALA FUNCTIONS

In general, we should give our functions meaningful names

```
def square(x : Int) : Int = {  
    x * x  
}
```

SCALA FUNCTIONS

“def” starts a function definition

function name

parameter list in parentheses

function's result type

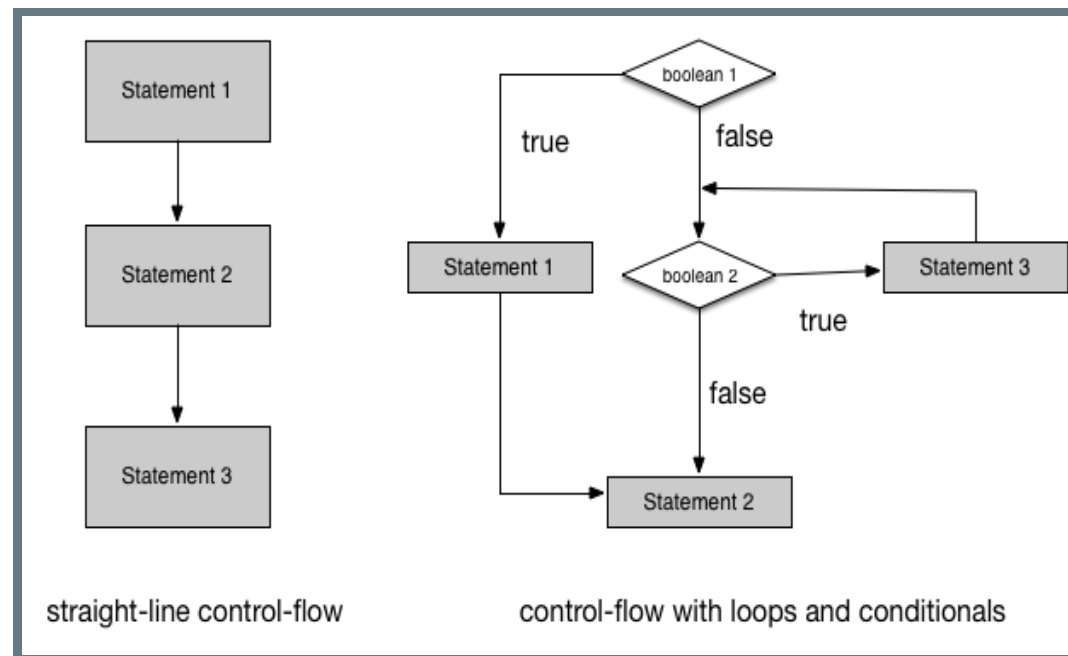
equals sign

```
def max( x : Int, y : Int) : Int = {  
  if (x > y)  
    x  
  else  
    y  
}
```

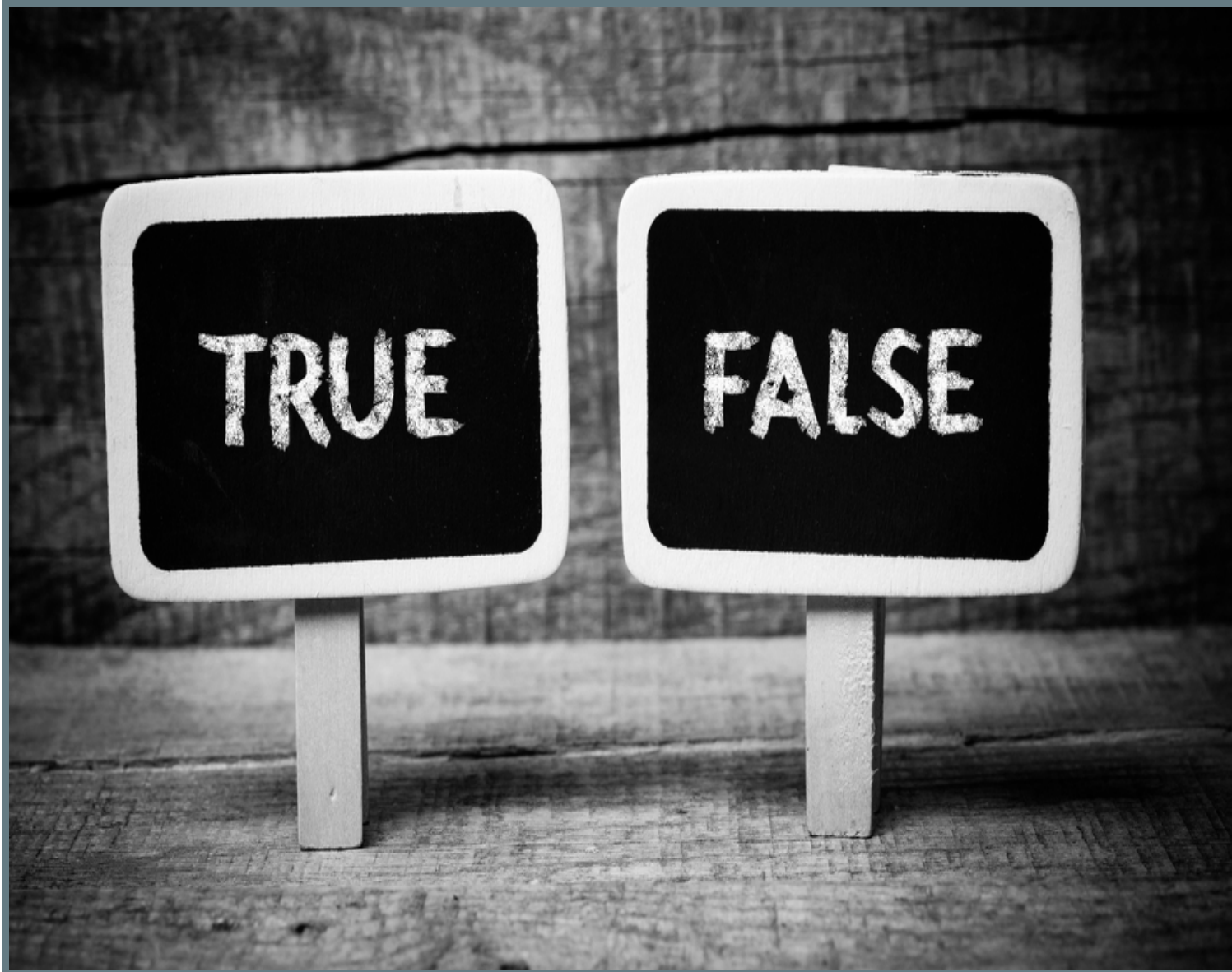
function body in curly braces

CONTROL FLOW

- Sequence of statements that are actually executed in a program
- Conditionals and loops, enable us to choreograph control flow



CONDITIONALS



IF-THEN-ELSE

The if statement is a form of branching

- Evaluate a boolean expression
 - If true, then execute some statements
 - If false, then execute some other statements

```
if (boolean expression) {  
    statement_i  
}  
else {  
    statement_j  
}
```

EXAMPLE (IS EVEN)

```
def isEven( x : Int ) : Bool = {  
    if (x % 2 == 0) { // % returns the remainder of an integer division  
        true  
    }  
    else {  
        false  
    }  
}
```

EXAMPLE (MAXIMUM)

```
def max( x : Int, y : Int ) : Int = {  
  if (x > y) {  
    x  
  }  
  else {  
    y  
  }  
}
```

EXAMPLE (IS EVEN, AGAIN)

We could have defined isEven as

```
def isEven( x : Int ) : Bool = {  
  (x % 2 == 0)  
}
```

Can you tell me why?

LOOPS



WHILE LOOP

The while loop is a common repetition structure

- Evaluate a boolean expression
 - If true, then execute some statements
 - Repeat

```
while (boolean expression) {  
    statement 1  
    ...  
    statement n  
}
```

EXAMPLE (GCD)

```
def gcd( x : Long, y : Long ) : Long = {  
  var a = x  
  var b = y  
  while (a != 0) {  
    val temp = a  
    a = b % a  
    b = temp  
  }  
  b  
}
```

EXAMPLE (SUM ELEMENTS OF LIST)

```
def sumList( xs : List[Int] ) : Int = {  
  var sum = 0  
  var ts  = xs  
  while (ts.length > 0) {  
    sum = sum + ts.head  
    ts  = ts.tail  
  }  
  sum  
}
```


ITERATION

- repetition of a mathematical or computational procedure applied to the result of a previous application, typically as a means of obtaining successively closer approximations to the solution of a problem

ITERATION IN SCALA

- While loops
- For loops
- Recursion loops
- Container functions, e.g. sum all elements of list

EXAMPLE (SUM ELEMENTS OF LIST)

```
def sumList( xs : List[Int] ) : Int = {  
    xs.sum  
}
```

SCALA FOR LOOPS

A for loop is another common repetition structure

- Iterate over a range
- Most basic example is over a range of integers
- More generally applies to any collection type, e.g. lists

```
for (i <- 1 to 4)
  println("Iteration " + i)
```

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
```

```
for (i <- 1 until 4)
  println("Iteration " + i)
```

```
Iteration 1
Iteration 2
Iteration 3
```

EXAMPLE (SUM ELEMENTS OF LIST AGAIN)

```
def sumList( xs : List[Int] ) : Int = {  
  var sum = 0  
  for (i <- xs) {  
    sum = sum + i  
  }  
  sum  
}
```

```
sumList(List(1,2,4,5))  
res17: Int = 12
```

EXAMPLE (SUM ELEMENTS OF LIST AGAIN)

Filter elements using a predicate

```
def sumEvensList( xs : List[Int] ) : Int = {  
  var sum = 0  
  for (i <- xs if isEven(i))  
    sum = sum + i  
  sum  
}
```

```
sumEvensList(List(1,2,4,5))  
res17: Int = 6
```

CONTROL FLOW SUMMARY

- Sequence of statements that are actually executed in a program
- Conditionals and loops enable us to choreograph the control flow

Control Flow	Description	Example
straight-line programs	all statements are executed in order	
conditionals	some statements are executed depending on conditions	if-else
loops	some statements are executed repeatedly on conditions	while for