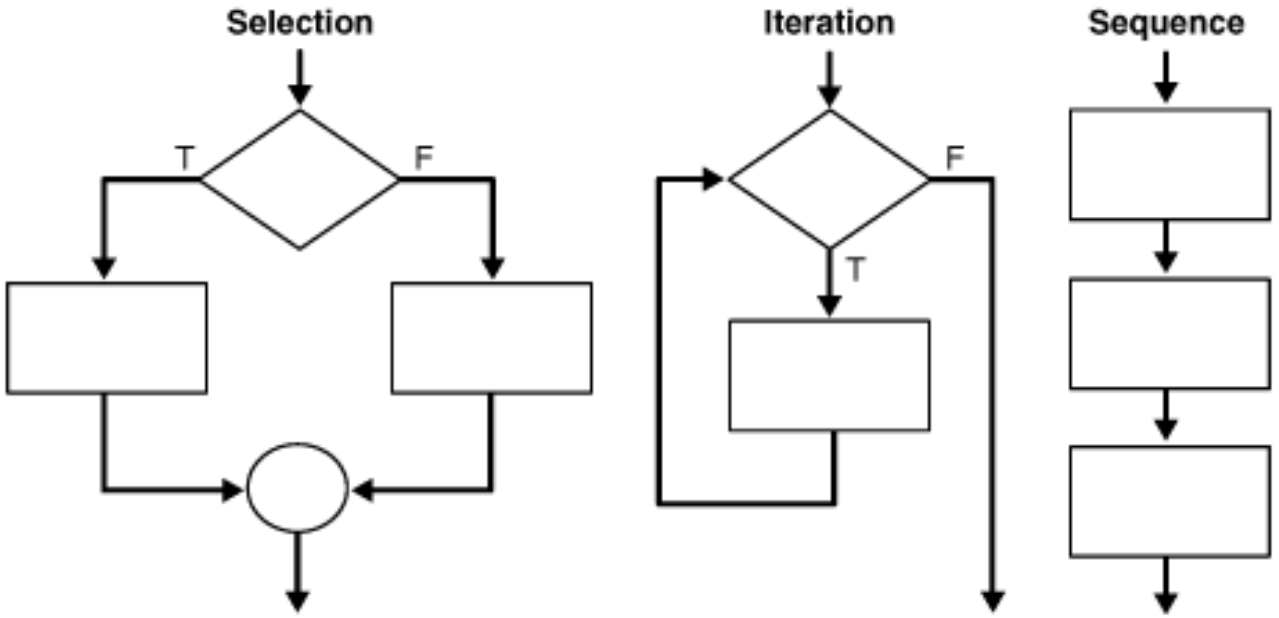
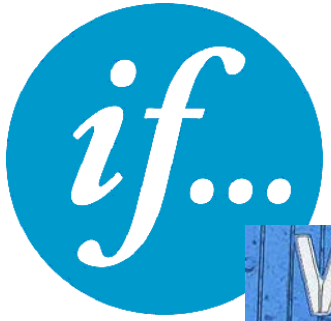


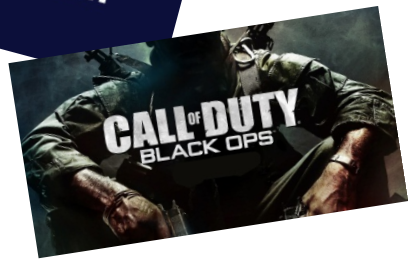
Control structures in Scala

CS Scala course 2012

Overview: control structures

- if
- while
- for
- match
- try
- Function calls

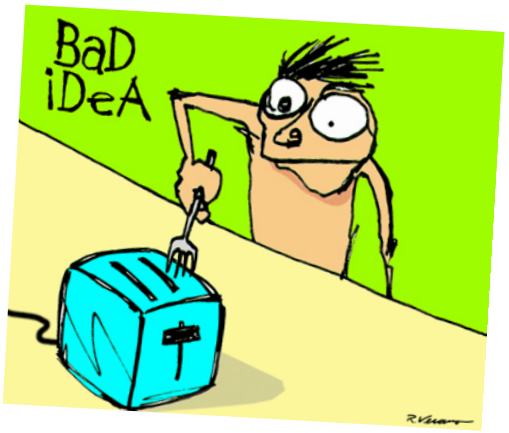




+ Exercise

Main ideas

- Few built-in control structures
  - Others could be find in libraries
- They work much like imperative counterparts
  - But they also have values (most of them)



Main ideas (2)

- In Java:
  - Expressions have values
  - Statements carries out an action

- In Scala:
  - In Scala, almost ALL constructs have values

if

Can be written in Java style:

```
if (n > 0) {
  r = r * n;
  n -= 1;
}
```

if (2)

But if returns a value!

```
var s = 0
if (x > 0) s = 1 else s = -1

val s = if (x > 0) 1 else -1
```

- Better because of *val*
- Semicolon optional

if (3)

if statements must have some value =>

omitted else returns **Unit** (≈ Java void)

if (x > 0) 1

equivalent to

if (x > 0) 1 **else** ()

Unit

while

Loops can be written in Java style

```
while (n > 0) {
  r = r * n;
  n -= 1;
}
```

There is also the do ... while loop

while (2)

There is no:

- break
- continue

Loops do NOT return a value =>

not used as often in Scala as in Java.

for

Scala has no direct analog of the Java for

for (*initialize*; *test*; *update*)

for (i **<-** *expr*)

<- to traverse all values of the right expression

for (i **<-** 1 **to** n)

the to method returns a Range (also: until)

for (2)

guard: an if inside the for

Example:

Filter out all numbers larger than 5

for (i <- expr; **if** i > 5)

for (3)

for (...) yield

- Creates a new collection of the same type as the original
- Contains the expressions after the yield, one for each iteration of the loop.

Example:

Double all elements larger than 5

```
val doubles = for (i <- expr; if i > 5 ) yield 2 * i
```

match

- Similar to switch statements
- Returns a value
- _ is used for default

```
val output = x match {
  case 1 => "one"
  case 2 => "two"
  case _ => "many"
}
```

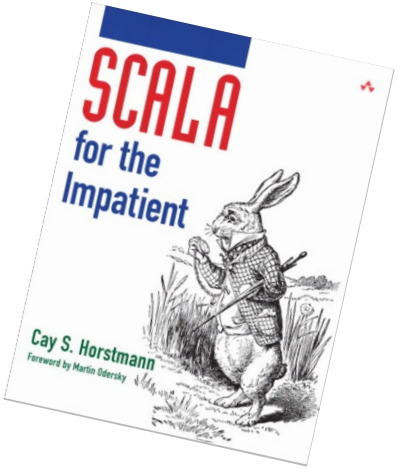
try

- Exceptions work as in Java
  - But you don't need to declare that a function might throw an exception

```
try {
  process(new FileReader(filename))
} catch {
  case _: FileNotFoundException => println(filename + " not found")
  case ex: IOException => ex.printStackTrace()
} finally { ... }
```

finally

Summary



- An if expression has a value
- A block has a value—the value of its last expression
- The Scala for loop is like an “enhanced” Java for loop
- Semicolons are (mostly) optional
- The void type is Unit
- Avoid using return in a function
- Exceptions work just like in Java or C++, but you use a “pattern matching” syntax for catch.