Compilation and More

Compiling a program can be done with `nim c`. Compiling and then running a program is possible using the `-r` flag, i. e. `nim c -r filename.nim`.

Variables

Nim is statically typed, hence the type of a declaration must be provided when defining a variable (much like Java and unlike Ruby). A variable can be defined as follows: —

```
# If value is unknown
var <name>: <type>

# If value is known
var <name>: <type> = <value>
```

Nim can, however, infer the type of a declaration, so that the following line of code is also valid: —

```
var <name> = <value>
```

It is, however, strongly recommended to always define the type of a declaration. The following are a handful of examples of variable declaration: —

Immutable Values and Declarations

Variables, as the name would already suggest, are *variable*; their values can change over the course of the program. At times, however, this is not intended and a static value is, instead, required. There are two ways of declaring these immutable, static values: `const` and `let`.

Const

Constants, which are defined by `const`, must be declared and known at compile time. Thus, a constant cannot be dependent on the output of, for example, a function. Therefore, the following declaration would result in a compilation error: —

```
var five: int = -5
const hyper: int = five + 5
```

Data Types

If Statements

If statements are part of *control flow*, i. e. only executing a certain piece of code when a condition is fulfilled. If statements in Nim are written as follows: —

```
if <condition>:
        <indented block of code>
```

It is, once again, important to intend the code not using tabs, but using spaces. If statements can actually be nested, so that the following would be possible: —

```
var Κλεόφιλος: string = "Marv"
if Κλεόφιλος ≠ "Marvin":
    echo "Οὐκ ἐστιν ὁ Κλεόφιλος ἀληθῶς ὁ Κλεόφιλος."
    if Κλεόφιλος & "in" == "Marvin":
      echo "Marvin has returned."
```

Else and Else If

Case Statements

Case statements work quite similarly to those found in Ruby. Their syntax differs somewhat, however: —

```
var 🎄: string = "Christmas"
case 🎄
of "Christmas":
        echo "Merry Christmas!"
of "Weihnachten":
        echo "Fröhliche Weihnachten!"
else: echo "Det här språket kan jag inte förstå."
```

Unlike if statements (and unlike Ruby's case statements), case statements *must cover all possible cases*; therefore, when nothing is supposed to happen in the `else` portion of the case statement, the following must be written: —

```
var 🎄: string = "Christmas"
case 🎄
of "Christmas":
        echo "Merry Christmas!"
```

Loops

Nim has a number of different loops, just like Ruby does too. The most commonly used ones are the for- and while-loops.

For-loop

The for-loop is pretty straight-forward and looks as follows, wherein the `iterable` is any object which one can iterate through: —

```
for <loopVariable> in <iterable>:
        <loop body>
```

Procedures (Functions)

Procedure are basically the Nim equivalent of what's known as "functions" in other programming languages. Apparently, this is the proper name for functions in programming languages, as they differ from mathematical functions: —

> *As mentioned in the beginning of this section, procedures are often called functions in other languages. This is actually a bit of a misnomer if we consider the mathematical definition of a function. Mathematical functions take a set of arguments (like `f(x, y)`, where `f` is a function, and `x` and `y` are its arguments) and always return the same answer for the same input.*
> *Programmatic procedures on the other hand don't always return the same output for a given input. Sometimes they don't return anything at all*

Declaring a procedure

Modules

Unlike Ruby, but like other programming languages such as Java or C, modules can be loaded to extend the functionality of the language. Some of the most commonly used modules include: —

- `strutils`
  - Additional functionality when dealing with strings
- `sequtils`
  - Additional functionality when dealing with sequences
- `math`
  - Mathematical functions (logarithms, square roots, ...), trigonometry (sin, cos, ...)
- `times`
  - Measure and deal with time

However, there are much more packages that can be found in either the standard library or

Exercises

Collatz conjecture