First steps in Kotlin

Like Python, Kotlin has an interactive mode, where you can try out things or just compute something interactively.

```
>ktc
Welcome to Kotlin version 1.0.6-release-127
Type :help for help, :quit for quit
>>> println("Hello World")
Hello World
>>> println("This is fun!")
This is fun!
```

To write a Kotlin script, create a file with name, say, test.kts, and run it by saying `kts test.kts` from the command line.

Later, we will see how to compile large programs (so that they start faster).

Typing

What is the biggest difference between Python and Kotlin?

Python is dynamically typed, Kotlin is statically typed.

In Python and in Kotlin, every piece of data is an object. Every object has a type. The type of an object determines what you can do with the object.

Dynamic typing means that a variable can contain objects of different types:

```
# This is Python
def test(a, b):
  print a + b

test(3, 15)
test("Hello", "World")
```

The + means different things.

Static typing

Static typing means that every variable has a known type.

If you use the wrong type of object in an expression, the compiler will immediately tell you (so you get a compilation error instead of a runtime error).

```
>>> var m : Int = 17
>>> m = 18
>>> m = 19.0
error: the floating-point literal does not
  conform to the expected type Int
```


Dynamic and static typing

Dynamic typing: Flexible, concise (no need to write down types all the time), useful for quickly writing short programs.

Static typing: Type errors found during compilation. More robust and trustworthy code. Compiler can generate more efficient code since the actual operation is known during compilation.

Java, C, C++, and Kotlin are all statically typed languages. But Kotlin is the only modern language among them: Kotlin uses type inference, and you don't have to write types all over your program.

C++11, Scala, Swift, and many modern functional programming languages have type inference.

```
>>> var t = 18.0
>>> ::t
var Line_0.t: kotlin.Double
```

Block structure

Programming languages make use of blocks of instructions: the body of a function, the body of a `while`-loop, the two alternatives of a conditional statement.

In Python, block structure is indicated by indentation.

In Kotlin, Java, and C, block structure is indicated by curly braces.

In Kotlin, a block is either a single expression, or a sequence of expressions surrounded by curly braces.

In Java and C, every statement is terminated by a semicolon. In Kotlin, the semicolon can usually be omitted at the end of a line.

Functions

In mathematics:

$$f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}, \quad f(a, b) = a + b$$

In Kotlin:

```
fun f(a: Int, b: Int) : Int = a + b
```

Argument types

Result type

Function definition

The function definition is a block: A single expression or a sequence surrounded by curly braces.

If the function definition consists of a single expression, then no `return` statement is needed.

val and var variables

Kotlin has two kinds of variables:

`val` variables can never change their value. After the variable name has been defined, it always keeps its current value:

```
>>> val u = 17
>>> u = 18
java.lang.IllegalAccessError:
  tried to access field Line16.u
```

`var` variables are like variables in Python—their value can change as often as you want.

It is easier to reason about programs if they use `val` variables.

The extreme case—programs without `var` variables—leads to a style called functional programming.

Functions with and without result

If the function definition is a block in curly braces, you return the result using `return`:

```
>>> fun f(a: Int, b: Int) : Int {
...   val s = a + b
...   return s
... }
>>> f(3, 5)
8
```

If the function returns no result, just omit the result type:

```
>>> fun greet(name: String) {
...   println("Hello $name, how are you?")
... }
>>> greet("Otfried")
Hello Otfried, how are you?
```