Dr Kirill Sidorov

Cardiff University
School of Computer Science and Informatics

LAB EXERCISE ONE

Values, expressions, variables, logic

Before you attempt these exercises, please make sure that you watched the video entitled *"Values, expressions, variables, logic"* on Learning Central.

You can work on these exercises in your own time or during the scheduled on-line lab sessions. If you find the first few exercises too easy — feel free to skip to the harder ones. If you get stuck — do not be shy to ask for help or advice from the teaching associates; they are here to help you. It is also ok to discuss the solutions with your peers (these labs are not assessed!), however make sure you understand everything by yourself.

Good luck!

1 Start the Python interpreter in interactive (REPL) mode. Depending on your setup you may need to type `python` or `python3` in your terminal.

```
$ python

Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

(The exact output may vary depending on the version, platform, *etc*.)
You can type

```
>>> quit()
```

to exit the interpreter.

2 Try using Python as an interactive calculator, by typing arithmetic expressions into REPL. For example, try the following:

```
>>> 2 + 3
>>> 4 * 12
>>> 2 ** 8
```

Feel free to try some expressions of your own.

3 What will happen if you execute the following?

```
>>> 5 / 0
```

What kind of error does this produce?

4 Use the `%` operator ("modulus", or "mod") to compute remainders after integer division. Try the following examples and make sure you *understand* the results you are getting.

```
>>> 12 % 10
>>> 5 % 9
>>> 7 % 7
>>> 0 % 7
>>> -1 % 7
>>> -8 % 7
>>> -15 % 7
>>> 1 % -7
>>> -5 % -7
```

If you are confused by how `%` works for negative operands, see `https://docs.python.org/3.8/reference/expressions.html#binary-arithmetic-operations` and note that:

- The modulo operator `%` always yields a result with the same sign as its second operand (or zero); the absolute value of the result is strictly smaller than the absolute value of the second operand.
- The floor division (`//` operator) and modulo operators are connected by the following identity: `x == (x // y) * y + (x % y)`

5 To develop some intuition for the order of operations, try computing the following expressions. In each case, make sure you completely understand how the interpreter computes the result. Compute the results *in your mind first*, before verifying them with Python.

```
>>> 3 * 2 + 7 % 3 ** 2
>>> 3 * (2 + 7) % 3 ** 2
>>> 3 * 2 + (7 % 3) ** 2
>>> (3 * 2 + 7) % 3 ** 2
>>> 3 * (2 + (7 % 3)) ** 2
>>> ((3 * 2 + 7) % 3) ** 2
>>> ((3 * (2 + 7)) % 3) ** 2
```

6 Python understands many various data types. Using the `type()` function, find out the types of the following values:

```
4
5.0
"Six"
True
```

7 Let's experiment with variables. Remember, the operator `=` ("assignment") binds a variable to a value. Try these simple commands:

```
>>> a = 5
>>> b = 4
>>> print(a)
```

```
>>> print(b)
>>> print(a + b)
>>> b = 3
>>> print(a + b)
```

What was the last result? Why?

8 To convert the temperature from Celsius to Fahrenheit the following formula is used:

$$F = \frac{9}{5}C + 32,$$

where C is the temperature in degrees Celsius, and F is the temperature in degrees Fahrenheit. Complete the expression (`F = ...`), and hence carry out the conversion in the example below:

```
>>> C = 21
>>> F = ...
>>> print(F)
```

9 Devise a formula for Fahrenheit to Celsius conversion (*i.e.* the other way round), and implement it in Python as in the previous exercise.

10 Many useful mathematical functions are found int the `math` module. You can load it this way:

```
>>> import math
```

You now have access to the various math functions described here:
https://docs.python.org/3.8/library/math.html
For example, to compute the square root, type:

```
>>> math.sqrt(4)
```

Complete the expression below to compute the hypotenuse `c` in a right triangle, given the catheti `a` and `b`.

```
>>> a = 3
>>> b = 4
>>> c = ...
>>> print(c)
```

11 Python understands Boolean (logical) values and supports a range of operations on them. The names of the logical values are `True` and `False`. Let's experiment with them. For each of the expressions below, first compute the result in your mind, and then verify it using Python.

```
>>> True
>>> not True
>>> False and True
>>> False or True
```

```
>>> (not False) and (False or True)
>>> (False or True) and (False or (True and True))
>>> ((not False) and (not True)) or ((not True) and (not False))
```

12 Comparison operations in Python result in Boolean values. Here they are: `<`, `>`, `==` (equals), `!=` (not equals), `<=`, `>=`. Like in the previous exercises, evaluate the expressions below in your mind first, before typing them into the Python interpreter to verify your results.

```
>>> 4 > 5
>>> (12 % 5) < 5
>>> 3 + 4 == 4 + 3
>>> ((1 > 2) or (3 < 4)) and (5 <= 5)
>>> (2 < 5) == (3 < 4)
```

Now, let's consider some expressions involving variables:

```
>>> x = 0; y = 1.2
>>> x >= 0 and y < 1
>>> x >= 0 or y < 1
>>> x > 0 or y > 1
>>> x > 0 or not y > 1
>>> -1 < x <= 0
>>> not (x > 0 or y > 0)
```

Again, make sure you understand how Python evaluates each expression and make sure you can predict and explain the results in each case.

13 Working with strings in Python is easy. Let's try some commands which manipulate the strings or return information about them. Feel free to apply these commands to other strings. As in all exercises, predict the result first before evaluating it with Python.

```
>>> "Hello" + ", " + "World!"
>>> a = "abra"
>>> b = "cad"
>>> s = a + b + a
>>> print(s)
>>> len(s)
>>> s.count("a")
>>> s[0]
>>> s[2]
>>> s[0:3]
>>> s[:5]
>>> s[5:]
>>> s[-5]
```