Lab 1

Python: Introduction

[Compulsory, but no submission/presentation]

Authors: Ragnar Nohre, modified and moved to English Johannes Schmidt

This lab's goal is to learn the basics of python programming. The lab is relatively long and requires to work with it beyond the scheduled lab sessions.

1.0 Intro

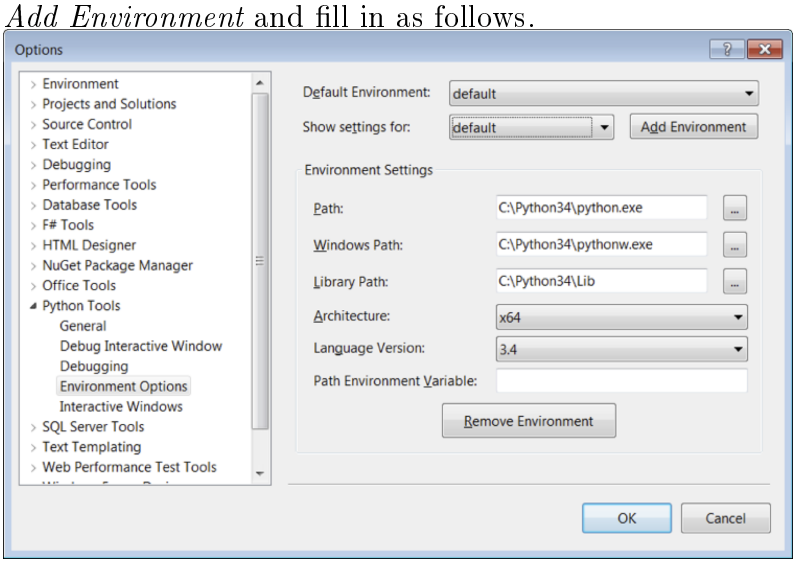
There is a very good tutorial on python.org. In the following we refer to it as the *web-tutorial*. This lab contains two parts. Part 1) is guidance on how to get started with python and the web-tutorial, and part 2) contains test questions and further aspects of python.

The purpose is that you work with the web-tutorial *before* you move to the corresponding test questions. If you choose to skip the web-tutorial you will learn less.

1.0.1 Install python

On the computers of the School of Engineering python should be available (version 3.5 or newer). As of 2024, the newest version is 3.13. You can work with Visual Studio (Code), but you need not. A simple text editor together with python from the command line does it as well.
It can happen that Visual Studio seems to have python available, but you

still encounter problems with the *helloworld* program. Open *TOOLS->Options* and choose *Python Tools -> Environment Options*, click *Add Environment* and fill in as follows.



If Visual Studio lacks python you should be able to install it via the Software Center.
You can also install python on your own **private** computer. It is for free and very simple. Go to **python.org**, download python via *downloads* and install it. If you install Visual Studio 2017, you can install python along with it. We need version *version 3.5* or newer. Mac computers have already a version of python installed. If it is python 2.x, you need to install python 3.5 or newer.

If you have Windows and Visual Studio 2015 on your private computer you should install a plugin that allows you to create and debug python projects in Visual Studio 2015. Google *python visual studio 2015* and you will find the link to *microsoft.github.io/PTVS*. Install that on your private computer.

1.0.2 Open a command line window and test

In this introduction you shall open a command line window (do this even if you use an IDE such as Visual Studio):

1. Open a command line window (in Windows search for *Windows Powershell* in the start menue and on Mac write *terminal* in spotlight).

2. Write the command `python3` (and hit return) in the terminal window. If this does not work, try `python` without the 3.
   (a) You should now see a pythonprompt (`>>>`), which means that you are in python's interpreter (interactive mode). Verify that the stated python version is 3.x (and not 2.x).
   (b) Write `3+2` and hit return. Verify that the interpreter answers `5`.
   (c) Execute the command `print("Hello, world!")`, and verify that the interpreter prints the text *Hello, world!*
   (d) Execute `quit()`, and verify that you quit the python interpreter and are back in the terminal.
3. Open an appropriate texteditor (e.g. notepad++ in Windows)
   (a) Create a new file. Write the text `print("Hello, world!!!")`.
   (b) Save the file under the name *hello.py* in an appropriate folder.
4. Back in the command line window. Navigate[1] to the latter folder where you saved the file `hello.py`.
5. Enter the command `python3 hello.py` (maybe skip the 3) and verify that your program is executed and *Hello, World!!!* is printed!

1.0.3 Test with Visual Studio

If you use a Windows computer with Visual Studio and python you can use Visual Studio:

1. Start Visual Studio
2. Create a new project with a name of your choice. Choose a python project (not iron python, etc.)
3. You should get a file with the ending `.py`. Write the text `print("Hello, world!")`
4. Press the RUN button (green triangle) and test run, verify that it works.

[1] In case you are unfamiliar with navigation in the command line: Write `cd` followed by the path of your file. Instead of typing the (sometimes long) path you can also drag a folder with the mouse from the explorer and drop it in the command line.

5. As python programmer you *always* want an interactive command line available, so you can test run and experiment with new commands that you are unsure about – before you write them in your program.
   (a) From Visual Studio's *TOOLS menue*, choose *Python* and then *Python Interactive Window*.
   (b) Verify that a window with a python interpreter becomes visible.
   (c) Test run the command `3+2` and verify that the result is `5`.

1.0.4 The web tutorial

On *python.org* you find a tutorial to learn python's basics.

Visit *python.org*. Move over *Documentations*, choose *Python 3.x Docs*. Click on the link **Tutorial** *start here*. The link leads to the tutorial as a *web document*.

Part 1 to 9 of the web tutorial is important and part of this course. This lab covers approximately part 1 to 7.

The remainder of this lab is reading instructions and test questions. The following chapter numbers correspond with the numbering in the web document. Remember to first work with the web tutorial before you go to the test questions of this lab.

1.1 Whetting Your Appetite

This is a short chapter with some advertisement for python. Good to read.

1.2 Using the Python Interpreter

This chapter contains some guidance on how to start python in interactive mode. We have already done that, but you find here additional useful information.

1.3 Introduction

Introduction to python. Subjects, among others, are:

- Numbers and operations on numbers
- Strings and some string operations
- Lists (similar to arrays, stacks, fifo queues, etc.)
- Programming. While-loops without curly braces etc.

Read the chapter attentively! Type in and test run all commands in interactive mode. If Visual Studio works as it should you get *interactive intellisense help* even in the command line.

1.3.1 Test questions on number

1. If you divide two integers with the /-operator, is the resultat always an integer (as it is in the programming language C)?
2. How do you achieve integer division in python?
3. What is `2**-1` ?

1.3.2 Test questions on Strings

1. What is `15*"spam "` ?
2. What is the easiest way to write a multi-line string?
3. What is the easiest way to write a string that contains citation symbols (quotation marks)?
4. Assume that `s = "Hello, world!"`
   (a) What is then `s[0:5]` ?
   (b) What is then `s[:5]` ?
   (c) What is then `s[-1]` ?
5. How do you determine the length of a string?
6. Provoke an error that shows that in python strings are immutable!

1.3.3 Test questions on Lists

1. Assume we do the following

```
a = [100,1,2,3,4,5,6,7]
r = a
s = a[0:3]
t = a[:]

a[0] = 99
```

(a) What is then `r` ?

(b) What is then `s` ?

(c) What is then `t` ?

(d) In the statement `x = y`, will `y` be *copied* or will `x` and `y` refer to *the same object?*

2. How to you determine the length of a list?

3. What is `m[1][0]` if `m = [[1, 2], [3, 4]]`?

1.3.4 Test questions on First steps...

1. Assume that `a` och `b` are two variables. What does the following statement do?

```
a,b   =   b,a
```

2. Write a `while`-loop that prints all integers between 1 and 9 on the screen!

1.4 More Control Flow Tools

Read this even though you can already program in other languages. There might be some (subtle) differences...

1.4.1 Test questions on the if-statement

1. What do you think, why does python use `elif` instead of `else if` (as in C)?

1.4.2 for-statement

1. Assume that `animal = ["dog, "cat", "elefant"]`. Write a `for`-statement that prints all elements of `animal`!

2. The following code comes from the web-tutorial

```
for w in words[:]:
    if len(w) > 6:
       words.insert(0, w)
```

   Why do we loop here over `words[:]` and not over `words` ?

1.4.3 range() function

1. Use the `range()`-function to write a `for`-loop that prints all integers between 0 and 99!

2. One might think that the `range()`-function returns a list, but that is not the case. Use the `range`- and `list`-functions to create a list that contains all integers between 0 and 99!

1.4.4 break, continue, else

1. A `for`-loop can have an `else`-branch. When is that branch executed?

1.4.5 Test question empty statement

1. In a C-program one can see sometimes empty statement in the form of a single semicolon or empty curly braces (`;` or `{ }`). In python one uses neither semicolon nor curly braces. How do you write an empty statement in python?

1.4.6 Defining functions

1. Write a function named `hello` that takes an integer `n` as input parameter and prints the string *parrot* `n` times!

2. What is `f` if one writes `f = hello`?

3. A C-program stores global data in the *heap* and local variables on the *stack*. How does a python program do it?

4. How do you access a global variable from within a function?

5. Assume that `animal = ["dog, "cat", "elefant"]`. How can you add the string `"flea"` at the end of the list?

1.4.7 More on defining functions

This chapter is relatively long an contains a lot of information.

keyword arguments

Write a function named `print_all` that prints all parameters it is called with. It shall be possible to call the function with an arbitrary number of parameters.

lambda functions

1. Assume we write

```
f = lambda a,b: a+b
```

   What is then `f(3,2)`?

document string

1. What would roughly happen if we write

```
print(  print.__doc__ )
```

2. How do you set a function's document string?

1.4.8 Intermezzo: Coding Style

1. A python program must be indentated correctly.Should one indent with tabs or with spaces?

2. Which ASCII-symbol (tab or space) writes Visual Studio if one hits the TAB-key?

1.5 Data structures

Many important data structures are inbuilt in pythyon, for instance:

- Lists, which compound stack operations
- Tuples and sequences, which are like lists that can not be changed.
- Sets
- Dictionaries, which store pairs consisting of key and data.

It is these data structures that render python a powerful language.

1.5.1 More on Lists

1. Assume that `a = [1,2,3,1,1,1,5,7,2]` is a list.
   (a) Indicate a statement that counts the number of 1'1 in the list.
   (b) Will the list be modified if one executes the command `a.sort()` or does the sort-function return a new list?

using lists as stacks

What will the variable `stack` contain after executing the following?

```
stack = [ 1, 2, 3, 4 ]
stack.append(8)
stack.pop()
stack.pop()
```

using lists as queues

What will the variable `queue` contain after executing the following?

```
queue = [ 1, 2, 3, 4 ]
queue.append(8)
queue.append(9)
queue.pop(0)
queue.pop(0)
```

Why is it **bad** to use a list as a fifo-queue?

list comprehensions

Assume that `a = [0,1,2,3,4,5,6,7,8,9,10]` is a list. Use a *list comprehension* to create a new list with the powers of two 1,2,4,8,...1024

1.5.2 Test questions on Tuples and Sequences

Let `t = 1111, 2222, 3333` be a tuple.

1. Which of the following functions should one *not* be able to to call on `t`, and why?
   (a) t.count(1111)
   (b) t.sort()
   (c) t.append(4444)
   (d) t.pop(0)
2. What does `x,y,z = t` do if `t` is as above?

1.5.3 Test questions on Sets

1. Assume that `answ` contains a string that a user just entered. Rewrite the following C++-code

```
if (answ=="yes" || answ=="yeah" || answ=="jop" || answ=="ok")
     cout << "you answered yes" << endl;
```

   into python-code that does the test with a set instead of many if-statements.
2. Write a python statement that converts a string to the set of symbols that are contained in the string!
3. Write a python statement that converts a list to the set of elements that are contained in the list!

1.5.4 Looping Techniques

1. Create a list and write then a loop that prints all elements of the list.
2. Create a dictionary object and write then a loop that prints all key/data-pairs of the object.

1.5.5 Test questions on More on Conditions

1. Translate the following C++-code fragment to python

```
if (20 <= age && age <=65)
     cout << "vad jobbar du med?" << endl;
```

   without using any AND-operator.

2. Is there a (logical) AND-operator in python? How does it look like?

1.5.6 Test question on Comparing Sequences...

1. Which of the following conditions are true?
   (a) `(1,2,3) < (1,3,2)`
   (b) `(2,1,3) < (1,2,3)`
   (c) `(1,2,3) < (1,2,3,4)`
   (d) `(2,2,2) < (1,1,1,1)`
   (e) `"c++" < "python"`

1.6 Modules

This chapter deals with how to split up a python program into several files.

Assume that the file `prog.py` contains, among other things, a function `getLotto()`.

1. What do you need to write in order to call the function `getLotto()` from the command line?

A small exercise

Create a file `prog.py` with a function `getLotto()` that shall return a set of 7 different numbers between 1 and 35. Call then `getLotto()` from the command line.

Hints:

1. Let a while-loop add random numbers to a set as long as the set's size is smaller than 7.

2. Import `random` and call `random.randint(1,35)` to get suitable random numbers.

3. In your command line window you should also import the module `importlib`, and do a `importlib.reload()` every time you reload `prog.py`.

1.7 Input and Output

1.7.1 Test question on Fancier output formatting

1. What does the statement `"A={} och B={}".format(1,2)` return?

1.7.2 Test question on Reading and Writing Files

1. Write a code section that opens a text file for reading and print all its rows on the screen.