Learn how to programme (in Python)

Marion Weinzierl

Advanced Research Computing

@arc_du

March 26, 2020

Outline

Materials used and recommended

- Python Wiki - Python for Non-Programmers
- How to think like a Computer Scientist
- A Whirlwind Tour of Python
- Software Carpentry - Programming with Python

Introduction

Course Objectives

By the end of this course you should know

- how a basic computer program is written and executed,
- what basic data types and control statements are,
- how to get and process user input and data,
- how to structure your code using functions,
- what can lead to your program not working, and what to do about it,
- where to find further resources to practice your Python programming.

Programming and Programming Languages

Why do we want to program?

Programming and Programming Languages

Programming means: make the computer do the work for you!

Programming and Programming Languages

Programming means: make the computer do the work for you!

- Do the maths
- Boring repetitions
- Too complicated/extensive tasks
- Big data sets
- ...

Programming and Programming Languages

Steps:

- Write your code in high-level programming language.
- Translate into low-level (machine/assembly) language.
- Execute program.

Programming and ~~Programming~~ Scripting Languages

Steps:

- Write your code in high-level programming language.
- Interpret code and execute directly

Programming and ~~Programming~~ Scripting Languages

Steps:

- Write your code in high-level programming language.
- Interpret code and execute directly

Beware: Oversimplification!

How do you start?

We go the easy way (so you can build it up from there):

- Choose Python.
- Skip the installation bit: Jupyter Notebooks.
- HOWEVER: You have to type yourselves! Don't copy-paste (yet)![1]
- Have a play.

[1] If you try to copy-paste from these slides, you will sometimes get a syntax error because of the characters used in the PDF.

Hello World!

Hello World! (in C++)

helloworld.cpp:

```
#include <iostream>

using namespace std;

int main() {
   cout << "Hello World!\n";
   return 0;
}
```

compile:

```
g++ -o helloworld helloworld.cpp
```

execute:

```
./helloworld
```

Hello World! (in Python)

```
print("Hello World!")
```

Hello World!

```
name = "Marion"

print("Hello " + name + "!")
```

Hello World!

```
name = "Marion"

print("Hello " + name + "!")

mySum = 2+3

print("I can add: 2+3 = ", mySum)
```

Basics

Basic Data Types

- Strings: "Heinz", 'Banana', 'He said "Hello"'
- Integers: 1, 2, 3, 22222222, -777
- Floats: -1.2, 0.0, 2.7182
- Booleans: True, False

Basics: Variables

- "I reserve a space in memory for my data bit, and I call it by the name x"
- Syntax: $name = value$

Examples

```
print('He said "Hello"')

myString = 'He said "Hello"'

print(myString)

type(myString)
```

Examples

```
print(2+5)

print("2+5 = " + 2 + 5)

print("2+5 = " + str(2 + 5))

print("2+5 = ", 2 + 5)
```

Examples

```
type(true)

type(True)

1+True

type(1+True)

bool(True+False)

bool(True and False)

bool(True or False)

not(False)
```

Basic Operations

- String: concatenation with $+$
- Bool: and, or, not
- Numerical data: $+$, -, *, /, %, **, abs, ...
- Order of execution:
  1. ()
  2. **
  3. *,/
  4. $+$, -
  5. Left-to-right (except exponentiation!)

$\Rightarrow$ Use parenthesis to make sure!

Basics: Comments (and Documentation)

```
# This is my programme to demonstrate how to
# do simple calculations in Python.

myNumber = 2

myOtherNumber = myNumber+5

myNumber = myOtherNumber/2 # I have to divide
                           # by 2 here, as the
                           # results are
                           # otherwise rubbish

print(myNumber)
```

Debugging and Types of Errors

- Errors in computer programs are called “bugs” for historic reasons.
- For complex projects, you will usually spend more time testing and debugging than writing code.
- Three types of errors:
  - Syntax errors - written the code wrongly
  - Semantic errors - written the wrong code
  - Runtime errors - something’s wrong with the code (during execution)

Have a play!

You could try

- what happens if you add a float and an integer,
- what happens if you mix numbers and bools in arithmetic expressions,
- how setting parenthesis changes the result of a large arithmetic expression,
- to print statements that include variables of different data types,
- try to reproduce each of the error types,
- ...

Getting Data in and out

User Input

```
# Get some user input

x = input()

print(x)

type(x)    # This will be a string
           # if you don't convert it

# Get the user's name

name = input("What's you name?")

print("Hello " + name)
```

Reading and writing files

```
# Create a file object
myFile = open("testfile.txt", "w")
```

Two things to note here:

- My object "myFile" is different from my file "testfile"!
- There are different modes:
  - read: r
  - (over-)write: w
  - append: a
  - read+write: w+ or r+

Reading and writing files

```
# Create a file object
myFile = open("testfile.txt", "w")

# Write - note special characters!
myFile.write("This is some text. \n \
And some more.")
myFile.write("\n\nI can also add numbers \
like this: %d %d \n" %(22, 333))

myFile.write(str(222))

# Don't forget to close the file
myFile.close();
```

see also f-strings

Reading and writing files

```
# Create a file object (this time for reading)
myFile = open("testfile.txt", "r")

# Read it and print it to screen
print(myFile.read())

# Try this:
print(myFile.read(7))
print(myFile.readline())
print(myFile.readlines())

# Don't forget to close the file
myFile.close();
```

What do we have here?

```
myList = [1, 2, 3, 4, 5] # A list!

print(myList)

print(myList[3])  # Note: [] not ()

print(myList[0]) # Start with 0!

print(myList[-1]) # Go backwards

print(myList[1:4]) # Include first,
                   # exclude last

print(myList[:2]) # More slicing
```

Have a Play!

https://www.w3schools.com/python/python_lists.asp

For Arithmetic Operations Better: Arrays

```
import array as arr

myList = [1, 2, 3, 4, 5]

myArray = arr.array('i', myList)
```

Alternative import:

```
from array import *

myList = [1, 2, 3, 4, 5]

myArray = array('i', myList)
```

see also numpy arrays, (here a tutorial)

Repetitions and Conditions

While-Loop

```
mySum = 0

while mySum < 100:

    mySum = mySum + 3 # Mind the indentation!

print(mySum)
```

For-Loop

```
mySum = 0

for i in range(5): # i goes from 0 to 4

    mySum += i # mySum = mySum + i

print(mySum)
```

For-Loop

```
mySum = 0

i = 0

mySum = mySum + i

i = 1

mySum = mySum + i

i = 2

mySum = mySum + i
⋮
```

For-Loop – Real-Life Examples

```
for i in range(1, 100):

  filename = "myfile"+str(i)+".dat"

  #Do stuff with that file
```

```
for i in range(100):

  currentFile = myFileList[i]

  #Do stuff with that file
```

```
for myFile in myFileList:
  #Do stuff with that file
```

If-Statement

```
num1 = float(input("Give me a number!"))
num2 = float(input("Another number!"))

if num1 > num2:
  print("Your first number is bigger than \
        your second number.")
else:
  print("Your first number is not bigger \
        than your second number.")
```

(There will be some annoying whitespace if you type it like that, just remove the linebreaks and the whitespace in the messages in your code.)

Do we trust the user...?

```
try:
    num1 = float(input("Give me a number!"))
    num2 = float(input("Another number!"))
    if num1 > num2:
        print("Your first number is bigger than \
              your second number.")
    else:
        print("Your first number is not bigger \
              than your second number.")
except:
        print("This wasn't a valid input, \
          I'm afraid.")
```

Have a play!

You could try

- writing numeric data into a file using a loop, and reading them into a list,
- defining a 2d list using a 'lists in a list' notation [[...],[...]] and try accessing its elements,
- doing nested loops,
- thinking about what type of errors *try-except* is able to catch (see slide 22),
- ...

Functions

Functions

We've seen functions (==useful code blocks that we need again and again, so we give them a name), for example *print*, *read(5)*, *open("testfile.txt", "w")*, etc.

$\Rightarrow$ Somewhere someone must have written some code for that...

Can we do that as well?

Functions

- Function Definition
- Arguments
- Return statement
- Function call
- Scope of variables

Functions

```
def myFunction(parameter1, parameter2):
    mySum = parameter1 + parameter2
    return mySum

def anotherFunction(parameter):
    print("Here you go: ", parameter)

myVar = myFunction(22, 55)

anotherFunction(myVar)
```

Functions

```
def myFunction(parameter1, parameter2):
    mySum = parameter1 + parameter2
    return mySum

def anotherFunction(parameter):
    print("Here you go: ", parameter)

myVar = myFunction(22, 55)

anotherFunction(myVar)
```

Variables defined inside a function are only available in that function.

Functions

```
def myFunction(parameter1, parameter2):
    mySum = parameter1 + parameter2
    return mySum

def anotherFunction(parameter):
    print("Here you go: ", parameter)

myVar = myFunction(22, 55)

anotherFunction(myVar)
```

Variables defined inside a function are only available in that function.

We should really give the functions and variables better names!

Flow of Execution (and debugging)

```
import pdb
pdb.set_trace()

def myFunction(parameter1, parameter2):
    mySum = parameter1 + parameter2
    return mySum

def anotherFunction(parameter):
    print(parameter)

myVar = myFunction(22, 55)

anotherFunction(myVar)
```

Advanced: Recursion

https://realpython.com/python-thinking-recursively/

Have a play!

You could try

- thinking about how to best name the variables and functions in the examples above, and why meaningful names are crucial,
- writing your own functions, and let one be called from within the other,
- writing a code that uses all that you have learned today,
- use the Python debugger to step through your code,
- ...

How to continue from here?

- Resources from the materials slide
- Research Methods Cafe
- Programmers are social!