Foundations of Programming in Python

Philipp Gloor[1]

[1]University of Zurich

About me

Education

- 2012 – Bachelor of Science UZH in Physics
- 2016 – Master of Science UZH in Computational Science

Work

- 2014 – 2016 Software engineer CERN (remote)
- 2016 – now PDF Tools AG

Programming experience

C++, C#, Java, JavaScript, Python

Email

philipp.gloor@gmail.com

Round of introduction

- Name
- Occupation
- Programming experience? What language?
- Expectations

Learning targets

After this course…

- … you will know what programming is
- … you will know how to write a basic computer program
- … you will know the fundamental components of programming
- … you are able to run Python code
- … you are able to write a Python program based on a written out problem statement
- … you know where you can find more information to improve your programming skills

Introduction to Programming

What is a Computer Program

Modular System

- **Input**: Data input from keyboard, files, internet, etc...
- **Output**: Processed data is displayed or saved to a file
- **Assignment**: Values are assigned to variables
- **Conditional execution**: Statements are executed only if certain conditions are fulfilled
- **Loops**: Repeating statement or group of statements
- **Libraries**: Using existing implementations

Examples: Hello World I

Java

```
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World");
    }
}
```

C++

```
#include <iostream>
int main() {
    std::cout << "Hello World" << std::endl;
    return 0;
}
```

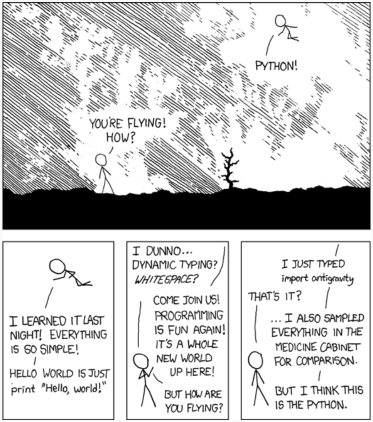
Examples: Hello World II

Python

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

Why Python?

- ”Simple” syntax
- High-level programming language
- Cross-platform
- Interpreted
- Object-oriented
- Many libraries available


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Source: https://xkcd.com/353/

Development Environment

- Integrated Development Environment (IDE)
- Collection of tools that are commonly used for software development
- Popular IDEs
    - Eclipse with pydev - `http://pydev.org`
    - JetBrains PyCharm - Community Edition available for free `http://jetbrains.com/pycharm/download`

Demo: Hello World

Options to run Python code:

- Directly in the Python prompt
- Write the code into a file and run python with the file
- Use IDE to run Python code

Fundamental Concepts

Values, Variables, Expressions, Operators, Comments

Values, Variables, Expressions, Operators, Comments

Values

- Numbers
  - 2
  - 1000000
  - -2
  - 3.2
  - 1.3333333
- Strings (Text)
  - ’Hello World’
  - ”Hello World”

Data Types

Strings

- ’Single quotes’ or ”double quotes” can be used to declare them
    - ’Hello World’
    - ”Hello World”
    - ”5”

Boolean

Binary data type

- True
- False

Variables I

- Variables hold values
- Similar to mathematics
    - x = 2
    - y = x + 2
- Values assigned using the `=` operator

Variables II

Examples

Use meaningful names

- Declaration

```
salutation = "Hello"
name = "Dennis Reynolds"
pi = 3.13159
```

- Usage

```
print(name)
```

Variables III

Keywords - reserved words

```
and, assert, break, class, continue, def, del, elif, else, except, exec,
finally, for, from, global, if, import, in, is, lambda, not, or, pass,
print, raise, return, try, while, yield
```

Variables IV

Variables and values can be combined

```
print(2+2)
a = 2
print(a+2)

salutation = "Hello"
name = "Dennis Reynolds"
print(salutation + " " + name)
```

Operators

Order of precedence

- ()
- **
- unary +,-
- *,/,%
- binary +,-
- `<, >, <=, >=, !=, ==`
- **`not`**
- **`and`**
- **`or`**

Comments

- Comments have no impact on the program
- Should explain the code
- A comment starts with a # character

Examples

```python
# Declaring the name
name = "Philipp"
print(name) # Prints Philipp
```

Functions

Functions I

- `print()` is a function that you have already used
- A function can take arguments which can be used inside the function

```
name = "Some name"
print(name) # Some name is used inside the print function
```

- Functions can also return a result
  - `return` statement

Examples

```
text = "Python programming language"
print(text) # Prints: Python programming language
text_length = len(text)
print(text_length) # Prints length of the string
```

Functions II

Type conversions

- `int('32')`: Converts a string that holds a number to an integer
- `int('Hello')`: This doesn’t work and it will throw a ValueError exception
- `float('313.333')`: Converts a string that hold a number to a float
- `str(32)`: Converts a number to a string

Examples

```
a = 20
b = 10
res = a + b
print("The sum of " + str(a) + " and " + str(b) + " is " + str(res))
```

Functions III

Rounding

```
a = 1.888
int(a) # = 1
int(round(a)) # = 2
int(a+5) # = 2
```

Math functions

```
import math
log_res = math.log(17.0)
sin_res = math.sin(45)
angle = 20
x = math.cos(20*math.pi/180))  # cos/sin etc take radians as arguments —>
     conversion from degree to radians necessary
```

- http://docs.python.org/library/math.html

Functions IV

User-defined functions

- A function encapsulates some functionality
- Reduces complexity

```
def my_function(param1, param2):
    print(param1)
    print(param2)
```

- Syntax is important
    - Indentation
    - The colon

Functions V

Examples

```
def line_separator():
    print('')

print("First Line")
line_separator()
print("Second Line")
line_separator()
print("Third Line")
line_separator()
print("Fourth Line")
```

- If we want to change the line separator to a dashed line we only need to change a single line of code

```
def line_separator():
print('------------------------------')
```

Functions VI

Examples

- If the line seperator should output two lines we can define a new function that calls the `line_separator()` function twice

```
def two_lines():
    line_separator()
    line_separator()

print ("First Line")
two_lines()
print("Second Line")
```

Functions VII

Parameters and arguments

- Arguments are passed when calling a function
- Value of arguments is assigned to parameters

```python
def print_sum(number_1, number_2):
    result = number_1 + number_2
    print(result)

print_sum(1,3)
print_sum(10,5)
```

Functions VIII

Parameters and arguments

- Variables are valid within a scope
- Variables that are defined in a function can only be seen inside that function
- Scope can be identified by indentation

```python
def concatenation(param1, param2):
    concat = part1 + part2
    print(concat)

concatenation("Hello", "World")
print(concat) # NameError: name 'concat' is not defined
```

Functions IX

Conclusion

- A function can be called multiple times
- If some code can be reused, put it in a function so you need to write less
    - Higher factorization
    - Less redundancy
    - Better maintenance
- Functions can also call other functions

Naming Conventions & Debugging

Naming Conventions I

How to name your functions and variables (PEP8)

- Naming convention is a set of rules for choosing names of functions and variables
- Every programming language has different naming conventions
- Python
  - No spaces in variable and function names
  - Variable and function names are in lowercase and _ is used to separate words

```python
length_in_cm = 15

def say_hello():
    print("Hello")
```

Debugging I

Finding and resolving ”bugs”

- Programming is a complex activity
- Mistakes happen all the time
- A mistake made in programming is called a bug
- The process of finding and resolving bugs is called debugging

Debugging II

Errors

- Syntax error
    - Incorrect syntax of a statement: `print(Hello World)` instead of `print("Hello World")`
- Runtime error
    - Error that occurs during the execution of a program
    - e.g. division by 0
- Semantic errors
    - Program does not deliver correct results
    - No error messages (code is syntactically correct)
    - Fixing semantic errors can be extremely complicated (good software design is important)

Debugging III

Techniques

- Reading code
- Print variables with `print()` to examine values (a poor man's debugger)
- Go through the program step by step -> **Debugger**!

Conditionals

Conditionals I

- Boolean algebra is a part of mathematics
- Often used in programming
- A boolean expression is either true or false

```
5 == 5 # —–> True
5 == 6 # —–> False
6 > 4 # —–> True
5 >= 8 # —–> False
```

Conditionals II

Examples

if

- The expression if defines a condition
- If the condition is true, subsequent statements will be executed
- If the condition is false, subsequent statements will not be executed
- There has to be at least one statement after the condition

```python
x = 10
if x > 0:
    print(str(x) + ' is positive')
if True:
    # This statement will always be executed
    print('Yes')
if False:
    # This statement will never be executed
    print('No')
```

Conditionals III

else

- Expression else is executed if the if condition is false
- Can only be used in combination with an if expression

```python
if x == 0:
    print(str(x) + ' is zero')
else:
    print(str(x) + ' is not zero')
```

Conditionals IV

Examples

%-operator (remainder after division)

```python
def print_parity(x):
if x % 2 == 0:
    print(str(x) + ' is even')
else:
    print(str(x) + ' is odd')

print_parity(2)
print_parity(3)
```

Functions with Return Values

Lists

Iteration

Dictionaries

Persistence

- Introduction to Programming
- Fundamental Concepts
  - Values, Variables, Expressions, Operators, Comments
  - Functions
  - Naming Conventions & Debugging
  - Conditionals
  - Functions with Return Values
  - Lists
  - Iteration
  - Dictionaries
- Persistence