Lerntraining Software

Python

Sophie Aerdker

Ruhr-Universität Bochum

Mai 2018

Table of Contents

Overview

Python is ...

- interpreted
- interactive
- object-oriented
- a beginners language!

This tutorial is mostly inspired by
`https://www.tutorialspoint.com/python/index.htm`.
Much of what you learn here you will find there!

This tutorial

- Introduction to basic programming techniques in `python` as well as an outlook to advanced methods
- There are exercises between some sections, where you can try out what you have learned bevor
- The slices are english, as most of the documentation so that you become familiar with the technical terms
- It will be a tight program, so don't worry if you don't understand everything directly! If you are intersted in programming, you can read a lot more in detail. I will present some literature later.

For you at home, here `python` is already installed!

- Check if `Python` is already installed: open a terminal and type "python"
- Linux:
- Windows:

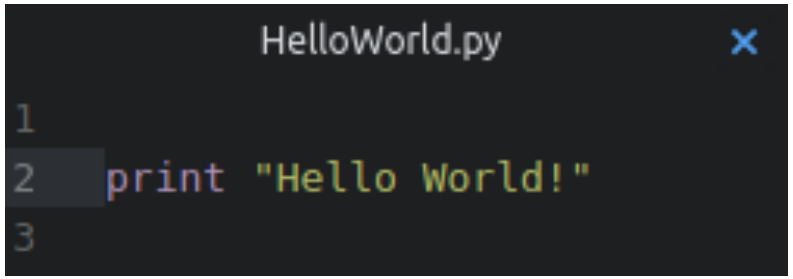
Using *Anaconda*:

- Linux:
- Windows:

Or use an IDE like *eclipse* or *Visual Studio*

Hello World!

- open an editor
- type: `print "Hello World!"`
- save it as "HelloWorld.py" under ...
- open a terminal and go to your directory with cd ...
- type: `python HelloWorld.py`



Simple Calculations

- now type e.g. `x = 5` and `y = 10` under your print statement
- type `print` and a calculation using +,-,*,/
- save the file, go to your terminal and type `python HelloWorld.py` or press ↑
- What is the result of `x/y`?

```
4   x = 5
5   y = 10
6   print x+y
7   print x*y
8   print x/y
9
```

Variable Types

Python has five basic data types:

- Numbers, like 5 and 10
- Strings, like "Hello World!"
- Lists,
- Dictionary, sometimes very useful but is not presented here
- Tuple

Data types can be stored in variables:

- `x = 10`
- `gravConstant = 9.81`
- `s = "Hello World!"`
- `name = "Sophie"`

Basic numerical types with some examples

Type	Examples	Comment
int	3, -42	signed integer $\leq 2,147,483,647$
long	51924361L	signed integer $> 2,147,483,647$
float	3.14, 3.0+e10, 0.	floating point real values
complex	42.0j, 2.+0.3j	complex numbers, imaginary unit j

Integer Division

The statement `5/10` is interpreted as an integer! Thus, its integer division is 0. Instead type `5./10.` to obtain a float-type value.

Boolean

The result of a comparison which is `True` or `False` is called Boolean. `True` and `False` are special versions of 1 (or any non-zero/null value) and 0, respectively. You can use them in arithmetic contexts.

Arithmetic and Comparison Operators

Operator		Examples
+	Addition	5+10 = 15
-	Subtraction	10-5 = 5
*	Multiplication	10*5 = 50
/	Division	10/5 = 2, 5/10 = 0, 5./10. = 0.5
**	Power	10**5 = 10,000
%	Modulus	10%5 = 0, 5%10 = 5
//	Floor Division	9.//2. = 4.0
==	equal	5==10 is False, 5==5 is True
!=	not equal	5!=10 is True, 5!=5 is False
>	greater than	10 > 5 is True
<	less than	10 < 5 is False
<= or >=		10 >= 5 is True, 5 <= 5 is True

Asignment Operators

Operator	Description	Example
=	Assigns values from the right side to the left side	x = 5+10
+=	Adds right operand to the left one AND assigns the result to the left operand	x += 1 is equivalent to x = x + 1
-=	x -= 1 is equivalent to x = x-1	
*=	x *= 2 is equivalent to x = x*2	
/=	x /= 2 is equivalent to x = x/2	
**=	x **= 2 is equivalent to x = x**2	
%=	x %= 2 is equivalent to x = x%2	
//=	x //= 2 is equivalent to x = x//2	

Other Operators

Bitwise operators

which perform bit by bit operations like binary AND, binary OR or shifting

Logical operators

`not` , `or` , `and`

Membership operators

`in` and `not in`

test the membership in a *sequence* such as lists or strings

Identity operators

`is` and `is not`

compare the memory locations of two objects, you can often use them like == and != for example in *if-statements*

Strings and string formatting

- In Python there is no difference between 'chars' and "strings", single and double quotes are treated the same.
- You can create strings simply by putting characters in quotes: str = `"Hello World!"`

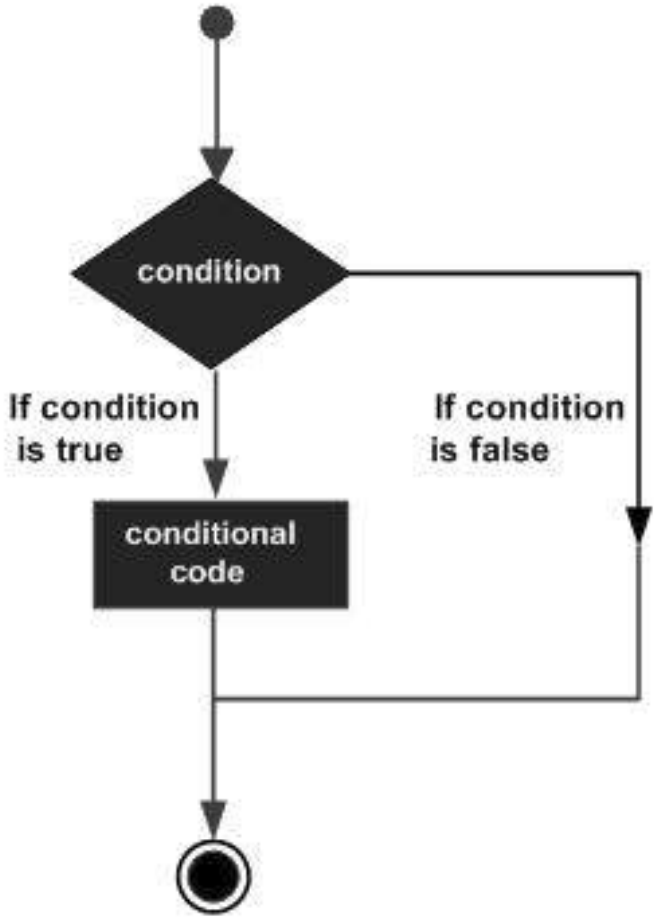
You can format strings using the string formatting operator '%':

```
obj = "Circle"
rad = 2
area = 12.566
print "The area of a %s with radius %i is: %f " %(obj, rad, area)
```

```
sophie@sophie-pc:~/Documents/Uni/SOWAS_Leitung/Le
The area of a Circle with radius 2 is: 12.566000
```

You can find a table with format symbols here: `https://www.tutorialspoint.com/python/python_strings.htm`

If-statements



use `if` and `else` conditions to execute a specific code if a condition is TRUE or to jump to the next (or another conditional) code otherwise

Example

Some `if`, `elif`, `else` statements to compare the values x and y:

```
print "x = %f and y = %f" %(x,y)
if (x > y):
    print "x is greater than y!"
elif(x < y): print "x is smaller than y!"
else:
    print "x is equal y!"
    if x and y >= 5.0:
        print "x and y are greater than 5!"

print "Finish..."
```

Output for different values of x and y:

```
sophie@sophie-pc:~/Documents/Un
x = 2.000000 and y = 3.000000
x is smaller than y!
Finish...
sophie@sophie-pc:~/Documents/Un
x = 6.000000 and y = 6.000000
x is equal y!
x and y are greater than 5!
Finish...
```

Syntax

The conditional code has to be intended or stands in a line (only possible for one statement) with the condition. IDEs and many editors do this automatically.

Exercises

- Write the program `EvenOdd.py` which returns wether a variable is even or odd!
  Use operators and condition statements and print the value as well as the result!
- Write the program `CharInString.py` which returns wether the string `"Hello World!"` contains a specific letter (a so called char)!
  Use membership operators and the program should be case sensitive to keep it simple.
- Bonus: Use a string method (`https://www.tutorialspoint.com/python/python_strings.htm`) to make `CharInString.py` case insensitive

Lists

A list contains several items like strings or numbers, you can easily create a list with [] and ,-separated items:

- `fruits = ["apple", "banana", "strawberry"]`
- `numbers = [1,1,2,3,5,8]`

The items in a list can be of different types:

- `book = ["title", "physics", 1997]`

You can access an item in a list by its index:

- `fruits[1]` is `"banana"`
- `numbers[3]` is 3
- `fruits[0]` is `"apple"`
- `numbers[-1]` is 8

Index

The index of a list starts with 0! You can use negative indices, too!

Manipulating Lists

You can use the index to change an item at the index position:

- `fruits[0] = "mango"` results in `["mango", "banana", "strawberry"]`

You can delete an item at a specific position:

- `del fruits[0]` results in `["banana", "strawberry"]`

List slicing with ':' :

- `numbers[1:4]` results in `[1,2,3]`
- `numbers[2:]` results in `[2,3,5,8]`

Merge lists:

- `numbers` + [13,21,34] results in `[1,1,2,3,5,8,13,21,34]`

Sequences

Strings

Lists and Strings both are sequences, thus accessing substrings and slicing strings works the same way!

List methods

There are some useful methods for lists:

- `len(fruits)` returns the length of the list: 3
- `max(numbers)` returns the maximum value: 8
- `min(numbers)` returns the minimum value: 1
- `list.sort([func])` sorts the list `list`, using an optional sort function
- `list.count(obj)` returns how often `obj` occurs in `list`
- `list.append(obj)` appends `obj` to `list`
- `list.remove(obj)` remove `obj` at any position from `list`

You can find more methods and their decription on `https://www.tutorialspoint.com/python/python_lists.htm` or in the python documentation

for-loops

If you want to repeat a statement while a specific condition is True, you can use loops:

- `while`-loops:
  repeats one or more statements while a condition is true; the condition is tested before executing the conditional code
- `for`-loops:
  iterates over the items of any sequence like strings or lists and executes the conditional code

Examples

for-loop over a list:

```
fruits = ["apple", "banana", "strawberry"]
for fruit in fruits:
    print "I like: %s" %fruit
```

```
sophie@sophie-pc:~/
I like: apple
I like: banana
I like: strawberry
```

while-loop:

```
index = 0;
while index < 4:
    print "Hello World!"
    index += 1
```

```
sophie@sophie
Hello World!
Hello World!
Hello World!
Hello World!
```

Examples

for-loop using `range()`:

```
for x in range(0,10,2):
    print x
```

```
soph
0
2
4
6
8
```

The `range()` method creates a list with the parameters `(begin, end, step)`, where the end-value is not included! Default values are: `begin = 0` and `step = 1`, thus `range(4)` wohuld create the list `[0,1,2,3]`.

Exercises

- Write the program `Faculty.py` that calculates the faculty of a certain number!
- Write the program `FibonacciSeries.py` that returns the Fibonacci Series! Use a list to store your result beginning with `Fibonacci = [0,1]`, the series should not exceed 10,000.
- Calculate the sum of all even numbers in your Fibonacci Series! You can reuse your code from exercise one.

There are many more mathematical problems or "number games" on `https://projecteuler.net/archives` you can solve with the few programming skills (but sometimes a lot of logical thinking) you have achieved here!

Functions

You can organize code, that performs a single action with help of functions. This makes your code more readable and reusable.
You already saw a lot of "build-in" functions, like the `print` statement or the methods to manipulating lists, like `max()`.
You can easily define your own functions using the `def` keyword:

```
def isEven(var):
    if var%2 == 0: return True
    else: return False
```

Calling a function:

```
x = 4
if isEven(4) is True: print "%i is even!"
else: print "%i is odd!"
```

Exercises

- Rewrite your `Faculty.py` program with a function `faculty(n)`!
- Write the function `faculty(n)` *recursive*. This means that the function calls itself! Take care of defining an end condition (like `faculty(0)` returns 1) , otherwise you get stuck in an infinite loop!

```
def isPrim(n):
    if n == 1 or n == 0: return False
    i = 2
    while i**2 <= n:
        if isPrim(i) == True:
            if n%i == 0: return False
        i += 1
    return True
```

Example:
Recursive function to calculate wether n is prime! But: this is not really efficient.

Modules

For the purpose of organize and reuse your code you can create *Modules* which are basically files containing Python code like functions, variables or *classes*. You can *import* such modules (or single functions) using the `import` statement:

main.py

```
import recursivePrime
from FacultyFunction import faculty

print faculty(4)
print recursivePrime.isPrim(4)
```

Directory path

Your main method and modules must be in the same directory or you have to define a PYTHONPATH.

Namespaces

To avoid typing the module name before each function you imported with the `import` statement or to confuse different functions with similar names, you can use *namespaces*.
Or you can import all items from one module into the current namespace using * (although this is not recommended and should be used wisely)

```
import recursivePrime as prim
import math as m

print prim.isPrim(4)
print m.exp(1)
```

```
from math import *

print exp(1)
print sin(2*pi)
```

Command Line Arguments

For a convenient use of your programs you wouldn't want to open your python file, change the variables, save the file and then run it over the command line. Instead it would be better to run the code and define the variables over the command line:

```
sophie@sophie-pc:~/repositories/SOWAS_Lerntraining$ python CommandLinePrime.py 5
5 is a prime number!
```

This works with the `sys`-module you can easily import. This system module provides the list `sys.argv` which contains command line arguments as strings.
Another way is to read the keyboard input during runtime of your program using the `raw_input` or `input` methods.

sys-module example

```
import sys
from recursivePrime import isPrim

n = int(sys.argv[1]) #sys.argv is a list with strings
#you entered in the command line after 'python', thus
#sys.argv[0] is always the file name! The function int()
#parses the string to an integer

if isPrim(n) is True: print "%i is a prime number!" %n
else: print "%i is not a prime number!" %n
```

Exercises

- Write a new command line based faculty program using the sys module! Import your old `faculty` function to calcuate it!
- Import the math module, and do some random calculation using its member functions. You can find them in the python documentary:
  `https://docs.python.org/2/library/math.html`

Opening and closing files

Reading and writing files

Matplotlib