Lerntraining Software

Python

Sophie Aerdker

Ruhr-Universität Bochum

Mai 2018

Table of Contents

Overview

Python is ...

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

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`

HelloWorld.py

```
1
2 print "Hello World!"
3
```

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`?

```
x = 5
y = 10
print x+y
print x*y
print x/y
```

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`

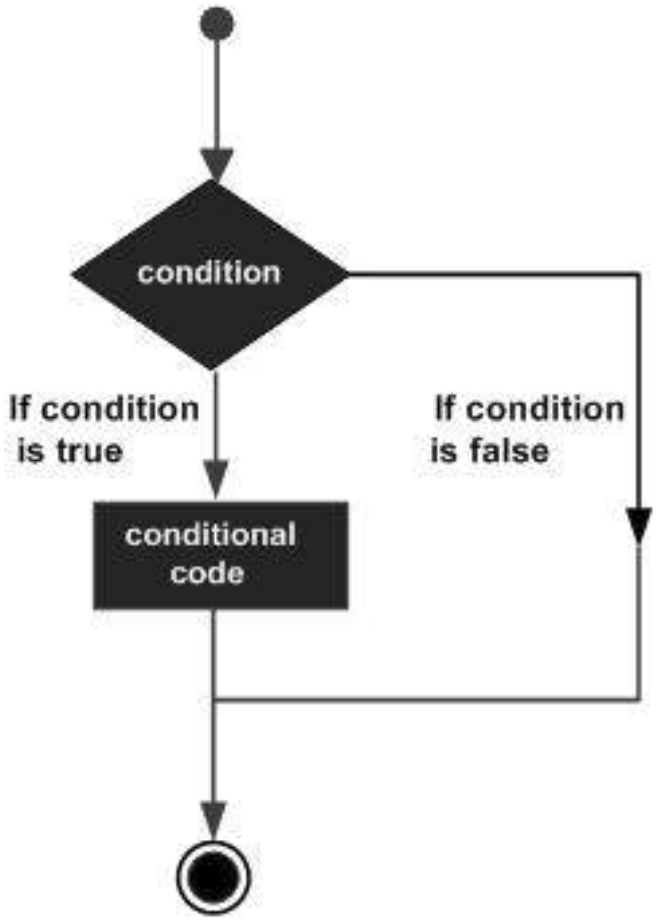
compare the memory locations of two objects, you can often use them like == and `!=` for example in *if-statements*

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 '%':
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"`
- `fruits[0]` is `"apple"`
- `numbers[3]` is 3
- `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 `fruits = ["mango", "banana", "strawberry"]`

You can delete an item at a specific position:

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

List slicing with ':' :

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

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, 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:

Exercises

- Rewrite your `Faculty.py` program with a function `faculty(var)`!
- Write the function `faculty(var)` *recursive*. This means that the function calls itself!