Visualize ComplexCity: Introduction to Programming I

28.02.2014

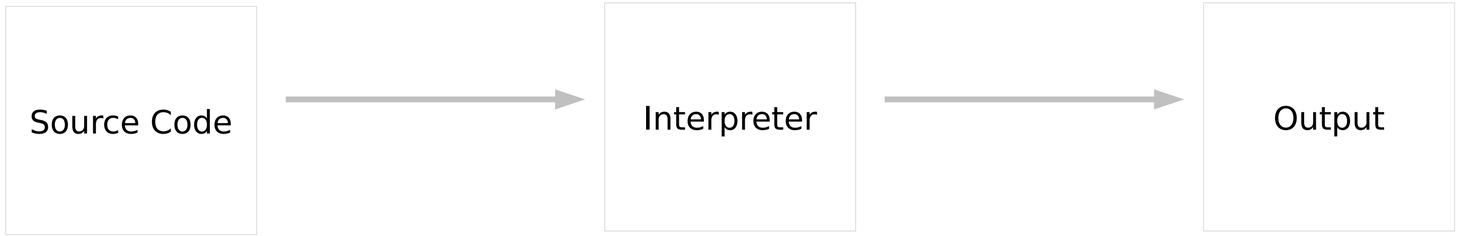
Chair of Information Architecture, ETH Zürich

Programming is the interaction between the programmer and the computer. The computer evaluates the formal text you write and executes the instructions of the text.

Programming means to solve problems and to give the instructions how to solve them to the computer.

Sorting: [9,3,5,2,8] -> [2,3,5,8,9]

Python is a **high-level programming** language. This means, we can write a **human readable** set of instructions. But an **interpreter is needed** to translate the text (code) to a machine readable format.



Python?

Introduction to Programming I

There are two ways/modes to interact with the interpreter.

- Interactive mode
- Script mode

You need to have the following software installed on your laptop:

- Python
- Your favourite programming IDE, e.g. Ninja-IDE (http://ninja-ide.org/)
- Tkinter Python library (https://wiki.python.org/moin/TkInter)
- Python Image library (http://www.pythonware.com/products/pil/)
- To test, if a libraries is installed, open the Python console and type:

  import Tkinter, Image

Let us start with you first programm. It shall print *Hello World!* to the prompt. We will be doing this by using the **interactive mode**.

- Go to your editor.
- Go to the prompt.
- Enter: *print('Hello World')*
- Hit *Enter*

We an achieve the same result by using the **script mode** in the following way.

- Go to your editor.
- Open a new file.
- Write *print('Hello World')* into the file.
- Press the run button.

Values are for example: *1*, *2* or *'Hello World!'*. Each value has a **type**, the numbers are of **numeric** type (in this special example of type **integer**). The *'Hello World!'* is a so called **string**. You can find out a type of a value by entering, for example, *type(2)*. This should print *<type 'int'>*.

There are many types present in Python. And you could also define your own types.

The basic types are:

- Boolean
- Numeric
- String

Types

Python Basics

Boolean types can have two values, either *True* or *False.*

Boolean

Data Types

Numeric types are numbers. There are two main types, **Integers** and **Floating Point** numbers.

Numeric types are numbers. There are two main types, **Integers** and **Floating Point** numbers.

- Integers are **whole numbers.**

  Examples: *-1, 43, 23, 1000*

- Floating Point numbers are **real numbers**.

  Examples: *2.7182, -3.1415, 3.0*

Strings are a string of letters. The interpreter knows that it is a string, when it is enclosed by quotation marks.

Example: *'Hello World!', '23', 'bar'*

A **variable** is a name that refers to a number. When you use an **assignment statement**, Python will create a new variable with a corresponding value.

Examples: *bar = 2*

foo = 2.0

anne = '2'

Operators represent a computation you want to do with values or variables. For example the operators *+, *, -, /* perform addition, multiplication, subtraction and division, respectively. It is **important** to ensure that the data type is correct when you use such an operation.

Example: 4/5 vs. 4.0/5.0

There are three basic data structures in Python.

- Lists
- Tuples
- Dictionaries

Data Structures

Data Types

A **list** is a sequence of values of any type. To initialize the list, put the value between two **brackets**.

Example: *foo = [42, [1,2,3], 'Hello World!' , 23]*

List

Data Structures

Lists are flexible and can be modified. You can add and remove elements. To access an element in the list, you have to use brackets.

Examples: *foo = [42, [1,2,3], 'Hello World!' , 23]*

foo[2] #access the third(!) element in the list.

foo.remove(42) #removes the first appearance of 42 in the list.

foo.append(42) #adds 42 to the end of the list.

Tuples are similar to lists, but they can not be altered after initialization. To initialize the tuple, put the values between **parentheses**. You can access the elements in a tuple the same way you would do it in a list.

Example: *bar = (42, [1,2,3], 'Hello World!', 23)*

bar[1] # accesses the second(!) element in the tuple.

Tuple

Data Structures

Dictionaries are so called hash tables. This means that every element has a corresponding key, and not an index like the elements in lists and tuples. To initialize the dictionary put the key-value pairs between curly brackets.

Example: *foo = {'answer':42, 'who':'brown fox'}*

Dictionary

Data Structures

Dictionaries are flexible and can be modified. You can add and remove elements. To access an element in the list, you have to use brackets.

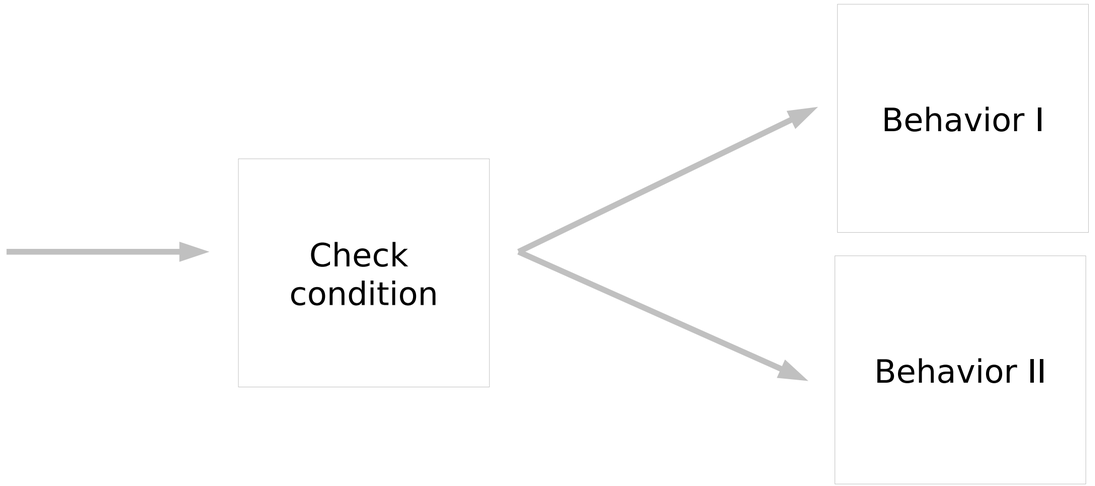
Examples:

```
foo = {'answer':42, 'who':'brown fox'}
bar = foo['answer']                  #sets bar to the value with the key answer.
foo['where'] = 'Ankh Morpork'        #add an element with the key where to foo.
foo.pop('who')                       #removes the elemenet with the key who.
```

Dictionary

Data Structures

Conditional Statements are an important construct in programming, since we always need to check some condition and let the program changing its behavior accordingly.



Conditional Statements

Python Basics

Boolean expressions are either true or false. The result is of type **Boolean**. Most of the time, relational operators are used.

Relational operators: *==, !=, <, >, <=, >=*

Do not confuse *=* with *==*, the first is the assignment operator, the other one the relational operator!

Logical operators are *or, and* and *not.* They are used to construct logical expressions.

Examples: *x < 2 and x >= 0*

x == 0 or x != 0

not (x == 0 or x > 100)

The simplest form of a **conditional statement** is the *if* statement. It checks if some condition is fulfilled and executes the code if it is *true*.

Example:
```
if aValue < 2:
    print('Hello World!')
```

If you want the program to execute an alternative, if a condition is not met you can add the *else* statement.

Example:
```
if aValue < 2:
    print('Hello World!')
else:
    print('Goodbye cruel world.')
```

Alternative Execution

Conditional Statements

It is also possible to chain conditional statements by using *elif*.

Example:

```
if aValue < 2:
    print('Hello World!')
elif aValue >100:
    print('Nice!')
else:
    print('Goodbye cruel world.')
```

Chained Conditionals

Conditional Statements

The second important construct in programming is **loops.** This makes it possible to repeat a statement multiple times.

There are two different kinds of loops, *for* loops and *while* loops.

A ***for*** **loop** iterates through a list and makes it possible to work with each element of the list.

Example:
```
for a in [1,32,4,2]:
    b = a * a
    print(b)
```

A ***for*** **loop** iterates through a list and makes it possible to work with each element of the list.

Example:

```
for a in [1,32,4,2]:
    b = a * a
    print(b)
```

Output:

```
1
1024
16
4
```

A ***while*** **loop** executes the corresponding sequence of instructions until some condition is not fulfilled anymore.

Example:

```
a = 1
while a < 16:
    a = a + a
    print(a)
```

A ***while*** **loop** executes the corresponding sequence of instructions until some condition is not fulfilled anymore.

Example:

```
a = 1
while a < 16:
    a = a + a
    print(a)
```

Output:

```
2
4
8
16
```

Very useful keywords when working with loop are *break* and *continue*.

- ***break*** terminates the loop and resumes after the loop construct.
- ***continue*** terminates the current iteration and returns to the beginning of the loop.

Example:

```
for a in [0,1,2,3,4,5,6,7,8,9]:
    if a < 3:
        print(a * 100)
    elif a == 5:
        continue
    elif a > 7:
        break
    else
        print(a)
```

Everything Together

Python Basics

Example:

```
for a in [0,1,2,3,4,5,6,7,8,9]:
    if a < 3:
        print(a * 100)
    elif a == 5:
        continue
    elif a > 7:
        break
    else
        print(a)
```

Output:

```
0
100
200
3
4
6
7
```

Everything Together

Python Basics

If you want to do the same computation in different parts of the code, you should define a **function**, so you only need to write the code once. You can then call that function if you need to do the computation.

A **function definition** start with the keyword *def*. This is followed by the function name and parameters needed for the computation. If you want to return the result of the computation, use the *return* keyword.

```
def functionName(parameters):
    instructions
    return variable
```

The following function takes one input and then returns the square of it.

```
def square(a):
    ret = a * a
    return ret
```

To call the function, you only need to state its name and set the input parameter.

Example: *square(3)*

Function: square

It is also possible to have mulitple inputs.

```
def multiply(a, b):
    ret = a * b
    return ret
```

To call the function, you only need to state its name and set the input parameters.

Example: *square(3, 4)*

Function: multiply

If a project gets bigger, it is not easily possible, to still remember which part which computations does. To make notes and **comments** in the programming code, we can use the **#**. The characters following the hash on that line will not be processed by the interpreter.

Example: *print('Hello World!') # prints Hello World! to the prompt.*

It is important to indent the code correctly, so that the interpreter knows what belongs to a loop, conditional statement, function, and so on. What is the output in the following code snippets?

```
for a in [1,3]:
    a = a * a
print(a)
```

```
for a in [1,3]:
        a = a * a
print(a)
```

The exercise of this weeks consists of two parts.

- First you will have to print patterns to the console by using the loop constructs.
- Second you will move around some data between a tuple, a list and a dictionary.

Exercise

Exercise

There exist some functions and operators, which can be useful to solve the first exercise.

- *range(start, end, step)*: it produces you a list of numbers, beginning with *start* up to, but excluding, *end*, with a stepsize of *step*. For example *range(0,21,10)* produces *[0,10,20]*.
- *%*: is the modulo operator, it calculates the rest of a division. For example *4%3* gives you *1* as a result.

Helper Functions & Operators

Exercise

Example: ###

00

#

Exercise I

Exercise

Example: ###

00

#

Code:
```
for a in range(3,0,-1):
    if a%2 == 0:
        symbol = '0'
    else
        symbol = '#'
    print(a*symbol)
```

Exercise I

Exercise

What the probably the easiest way would be to solve all the small exercises is to hard code all the lines in a list and then just print them. But this will not be accepted as a solution!

Not accepted solution:

```
for a in ['###','00','#']:
    print(a)
```

The following tips will help you with the second part of the exercise.

- *len(aList)* : this returns the length of the list *aList*. For example: *len([0,0,0])* returns *3*.
- *n * [0]* : this produces a list with all elements equal to zero with *n* elements in it. For example: *3 * [0]* produces *[0,0,0]*.

- *http://www.tutorialspoint.com/python*
- *http://docs.python.org/3/*
- *http://greenteapress.com/thinkpython/html/index.html*
- *And many more...*