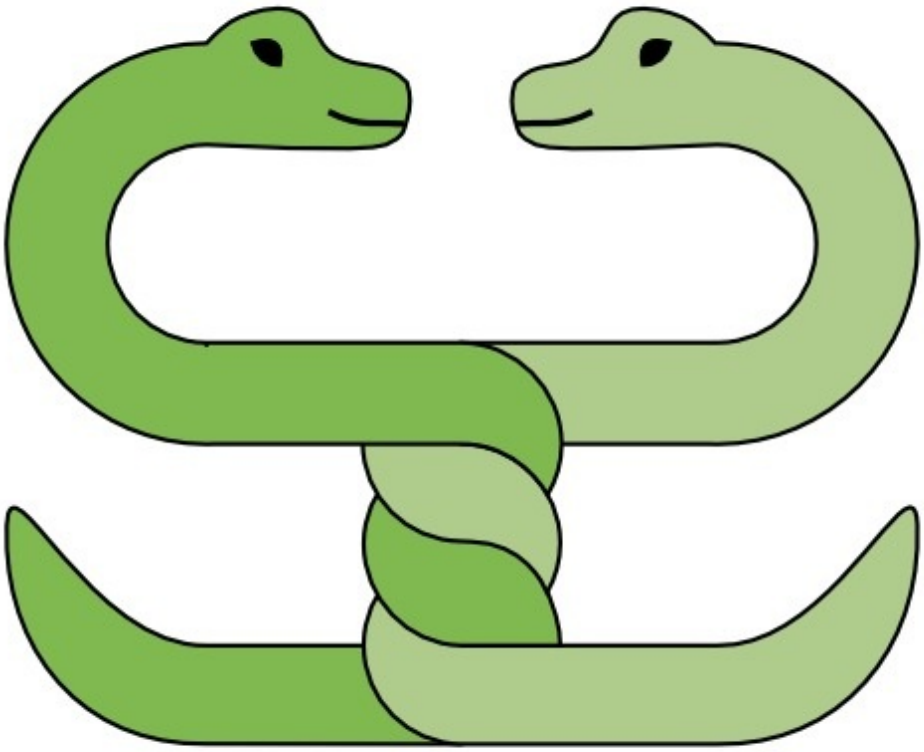
ACADEMIS

Python 3 Basics



tutorial

Tabela de conteúdos

Tutorial de Conceitos Básicos do Python 3

(c) 2015 Dr. Kristian Rother (krother@academis.eu)

com contribuições de Allegra Via, Kaja Milanowska e Anna Philips

tradução de Fabricio Biazzotto

O código fonte deste documento poderá ser obtido em
https://github.com/biazzotto/Python3_Basics_Tutorial_pt-BR

Introdução

A quem se destina este tutorial?

Este tutorial destina-se a programadores novatos. Você é o aluno que tinha em mente enquanto escrevia este tutorial se:

- já trabalhou um pouco com uma linguagem de programação diferente como R, MATLAB ou C.
- não possui nenhuma experiência prévia com programação
- conhece bem o Python e gostaria de ensinar os outros

Este tutorial funciona melhor se você seguir os capítulos e exercícios passo a passo.

Se for um programador experiente

Se você já é fluente em qualquer linguagem de programação, este tutorial pode ser muito fácil para você. Claro, você pode trabalhar com os exercícios para ter a sintaxe do Python em suas mãos. No entanto, este tutorial contém muito pouco material sobre os níveis de abstração mais elevados do Python, como classes, namespaces ou até mesmo funções.

Para um tutorial para não-iniciantes, recomendo os seguintes livros gratuitos online (em inglês):

- Learn Python the Hard Way - a bootcamp-style tutorial by **Zed Shaw**

- How to think like a Computer Scientist - a very systematic, scientific tutorial by **Allen B. Downey**
- Dive into Python 3 - explains one sophisticated program per chapter - **by Mark Pilgrim**

The dataset of U.S. baby names

The authorities of the United States have recorded the first names of all people born as U.S. citizens since 1880. The dataset is publicly available on http://www.ssa.gov/oact/babynames/limits.html . However for the protection of privacy only names used at least 5 times appear in the data.

Throughout this tutorial, we will work with this data.

Instalando o Python

O primeiro passo na programação é ter o Python instalado no computador. Basicamente você precisará de duas coisas:

- o próprio Python
- um editor de textos

Qual instalar, depende do seu sistema operacional.

No Ubuntu Linux

Por padrão, o Python já vem pré-instalado. Porém, neste tutorial iremos utilizar o Python3 sempre que possível. Você poderá instalá-lo a partir da linha de comando (Ctrl+Alt+T) com:

```
sudo apt-get install python3
sudo apt-get install ipython3
```

Para verificar se tudo correu bem, digite:

```
ipython3
```

No caso do editor de textos seria legal iniciarmos com o **gedit**. Por favor, certifique-se de mudar tabulações para espaços em *Preferências -> Editor -> marque 'Inserir espaços em vez de tabulações'*. Também é recomendável, nesta mesma tela, reduzir *Largura das tabulações:* para *4*.

No Windows

Uma maneira conveniente de instalar o Python, um editor e muitos pacotes adicionais em uma única etapa seria o **WinPython**.

Após a instalação do WinPython, é conveniente executar o *WinPython Control Panel*, e selecionar *Advanced -> Register distribution...*.

Outras distribuições Python

- **Python 3** - a instalação padrão do Python
- **Anaconda** - uma distribuição Python com diversos pacotes pré-instalados para aplicações científicas
- **Canopy** - outra distribuilção Python, que inclui editor e diversos pacotes pré-instalados

Outros editores

- **Idle** - o editor padrão do Python
- **Sublime Text** - um editor de textos muito poderoso para todos os sistemas operacionais
- **Notepad++** - um poderoso editor de textos para o Windows.
- **PyCharm** - um ambiente de desenvolvimento profissional para Python, com capacidade para lidar com grandes projetos. Você não precisará da maioria das funcionalidades por um longo tempo, mas é um editor muito bem feito.
- **vim** - um editor de textos baseado em console para sistemas Unix. A ferramenta preferida de muitos administradores de sistema.

Perguntas

Pergunta 1

Quais editores de texto estão instalados em seu sistema?

Pergunta 2

Qual a versão do Python que você está rodando?

First steps on the IPython Shell

There are two ways to use Python: The *interactive mode* or *IPython shell* and *writing programs*. We will use the interactive mode to store data on **U.S. baby names**.

In the first part of the tutorial, you will use the IPython shell to write simple commands.

Goals

The commands on the IPython shell cover:

- How to store a number?
- How to add two numbers together?
- How to store text?
- How to convert a number to text and back?

Using Python as a calculator

Warming up

Enter Python in the interactive mode. You should see a message

```
In [1]:
```

Complete the following calculations by filling in the blanks:

```
In [1]: 1 + ___
Out[1]: 3

In [2]: 12 ___ 8
Out[2]: 4

In [3]: ___ * 5
Out[3]: 20

In [4]: 21 / 7
Out[4]: ___

In [5]: ___ ** 2
Out[5]: 81
```

Enter the commands to see what happens (**do not** type the first part `In [1]` etc., these will appear automatically).

Exercises

Complete the following exercises:

Exercise 1:

There are more operators in Python. Find out what the following operators do?

```
3 ** 3
24 % 5
23 // 10
```

Exercise 2:

Which of the following Python statements are valid? Try them in the IPython shell.

```
0 + 1
2 3
4-5
6 * * 7
8 /
9
```

Exercise 3:

Which operations result in 8?

- [ ] `65 // 8`
- [ ] `17 % 9`
- [ ] `2 ** 4`
- [ ] `64 ** 0.5`

The Challenge: Boys' names total

In the table, you find the five most popular boys' names from the year 2000. What is the total number of boys in the top5?

name	number
Jacob	34465
Michael	32025
Matthew	28569
Joshua	27531
Christopher	24928

Storing numbers

The U.S. authorities record the names of all babies born since 1880. How many babies with more or less popular names were born in 2000? Let's store the numbers in *variables*.

Warming up

Let's define some variables:

```
In [1]: emily = 25952
In [2]: hannah = 23073
In [3]: khaleesi = 5
In [4]: emily
Out[4]: ______
In [5]: hannah + 1
Out[5]: ______
In [6]: 3 * khaleesi
Out[6]: ______
```

Let's change the content:

```
In [7]: emily = emily + 1
In [8]: emily
Out[8]: _____

In [9]: all_babies = 0
In [10]: all_babies = _____ + _____ + _____
In [11]: all_babies
Out[11]: 49031
```

Insert the correct values and variable names into the blanks.

Exercises

Complete the following exercises:

Exercise 1:

Which of the following variable names are correct? Try assigning some numbers to them.

```
Emily
EMILY
emily brown
emily25
25emily
emily_brown
emily.brown
```

Exercise 2:

Which are correct variable assignments?

- [ ] `a = 1 * 2`
- [ ] `2 = 1 + 1`
- [ ] `5 + 6 = y`
- [ ] `seven = 3 * 4`

The Challenge: Calculate an average

What is the average number of the top five boy names?

Calculate the number and store it in a variable.

Make a very rough estimate before you do the calculation.

Storing text

Warming up

So far, we have only worked with numbers. Now we will work with text as well.

```
first = 'Emily'
last = "Smith"
first
last

name = first + " " + last
name
```

What do the following statements do:

```
name[0]

name[3]

name[-1]
```

Exercises

Exercise 1:

Is it possible to include the following special characters in a string?

```
. 7 @ ? / & * \ Ä ß " ' á
```

Exercise 2:

What do the following statements do?

```
first = `Hannah`

first = first[0]

name = first

name = ""
```

Exercise 3:

Explain the code

```
text = ""
characters = "ABC"
text = characters[0] + text
text = characters[1] + text
text = characters[2] + text
text
```

The Challenge

first name	Andrew
last name	O'Malley
gender	M
year of birth	2000

Write the information from each row of the table into a separate string variable, then combine them to a single string, e.g.:

```
O'Malley, Andrew, M, 2000
```

Converting numbers to text and back

Now you know how to store numbers and how to store text. Being able to convert the two into each other will come in handy. For that, we will use *type conversions*.

Warming up

Now we are going to combine strings with integer numbers.

```
name = 'Emily Smith'
born = _____
____ = '15'

text = ____ + ' was born in the year ' + _____
year = born + _____
text
year
```

Insert into the following items into the code, so that all statements are working: `age`, `int(age)`, name, `str(born)`, `2000`

Questions

- Can you leave `str(born)` and `int(age)` away?
- What do `str()` and `int()` do?

Exercises

Exercise 1:

What is the result of the following statements?

```
9 + 9

9 + '9'

'9' + '9'
```

Exercise 2:

Change the statements above by adding int() or str() to each of them, so that the result is 18 or '99', respectively.

Exercise 3:

Explain the result of the following operations?

```
9 * 9
9 * '9'
'9' * 9
```

Exercise 4:

Write Python statements that create the following string:

```
12345678901234567890123456789012345678901234567890
```

The Challenge: A data record with types

field	value	type
first name	Andrew	string
last name	O'Malley	string
gender	M	string
year of birth	2000	integer
age	15	integer

Write the values from each row of the table into string or integer variables, then combine them to a single one-line string.

Writing Python programs

Using the interactive IPython alone is exciting only for a while. To write more complex programs, you need to store your instructions in *programs*, so that you can *execute* them later.

In the second part of the tutorial, you will learn a basic set of Python commands. Theoretically, they are sufficient to write any program on the planet (this is called *Turing completeness*).

Practically, you will need shortcuts that make programs prettier, faster, and less painful to write. We will save these shortcuts for the later parts.

Goals

The new Python commands cover:

- How to write to the screen.
- How to read from the keyboard.
- How to repeat instructions.
- How to store many items in a list.
- How to make decisions.

Writing to the screen

In this section, you will write your first Python program. It will be the most simple Python program possible. We simply will have the program write names of babies to the screen.

Warming up

Open a text editor window (*not a Python console*). Type:

```
print("Hannah")
print(23073)
```

Now save the text to a file with the name `first_program.py`.

Then run the program.

- In **Canopy**, you do this by pressing the 'play' button.
- In **IDLE**, you can execute a program by pressing F5.
- On **Unix**, you go open a terminal (*a regular one, not IPython*) and write:

  python3 first_program.py

Exercises

Exercise 1

Explain the following program:

```
name = "Emily"
year = 2000
print(name, year)
```

Exercise 2

Write into a program:

```
name = "Emily"
name
```

What happens?

Exercise 3

Which `print` statements are correct?

- [ ] `print("9" + "9")`
- [ ] `print "nine"`
- [ ] `print(str(9) + "nine")`
- [ ] `print(9 + 9)`
- [ ] `print(nine)`

The Challenge: Write a table with names

Many names of fictious characters that have been used as baby names. Write a program that produces an output similar to:

```
Harry      Hermione      Severus
Tyrion     Daenerys      Snow
Luke       Leia          Darth
```

Extra challenges:

- Use a single `print` statement to produce the output.
- Store the names in separate variables first, then combine them.
- Use string formatting.

Reading from the keyboard

Next, we will connect the keyboard to our program.

Warming up

What happens when you write the follwing lines in the IPython shell:

```
In  [1]: a = input()
In  [2]: a
```

Exercise 1

Which `input` statements are correct?

- [ ] `a = input()`
- [ ] `a = input("enter a number")`
- [ ] `a = input(enter your name)`
- [ ] `a = input(3)`

The Challenge: Enter a baby name

Write a program that asks for a *name* and an *age*, then writes a sentence with the entered data:

```
Bobby is 3 years old.
```

Extra challenge:

- Add 1 to the age entered.

Repeating instructions

So far, each Python instruction was executed only once. That makes programming a bit useless, because our programs are limited by our typing speed.

In this section you will learn the `for` statements that repeats one or more instructions several times.

Warming up

What does the following program do?

```
for char in 'Emily':
    print(char)
```

What advantages does this (apparently more complex) program have over this one:

```
print('E')
print('m')
print('i')
print('l')
print('y')
```

Exercises

Exercise 1

What does the following program do?

```
text = ""
characters = "Hannah"
for char in characters:
    text = char + text
print(text)
```

Exercise 2

Write a for loop that creates the following output

```
000
111
222
333
444
555
666
777
888
999
```

Exercise 3

Write a for loop that creates the following string variable:

```
000111222333444555666777888999
```

Exercise 4

Write a for loop that creates the following output

```
1
3
6
10
15
21
28
```

Exercise 5

Write a for loop that creates the following string

```
"1 4 9 16 25 36 49 64 81 "
```

Exercise 6

Add a single line at the end of the program, so that it creates the following string:

```
"1 4 9 16 25 36 49 64 81"
```

Exercise 7

Which of these `for` commands are correct?

- [ ] `for char in "ABCD":`
- [ ] `for i in range(10):`
- [ ] `for num in (4, 6, 8):`
- [ ] `for k in 3+7:`
- [ ] `for (i=0; i<10; i++):`
- [ ] `for var in seq:`

The Challenge: count characters

Write a program that calculates the number of characters in `Emily Smith`.

Extra challenge

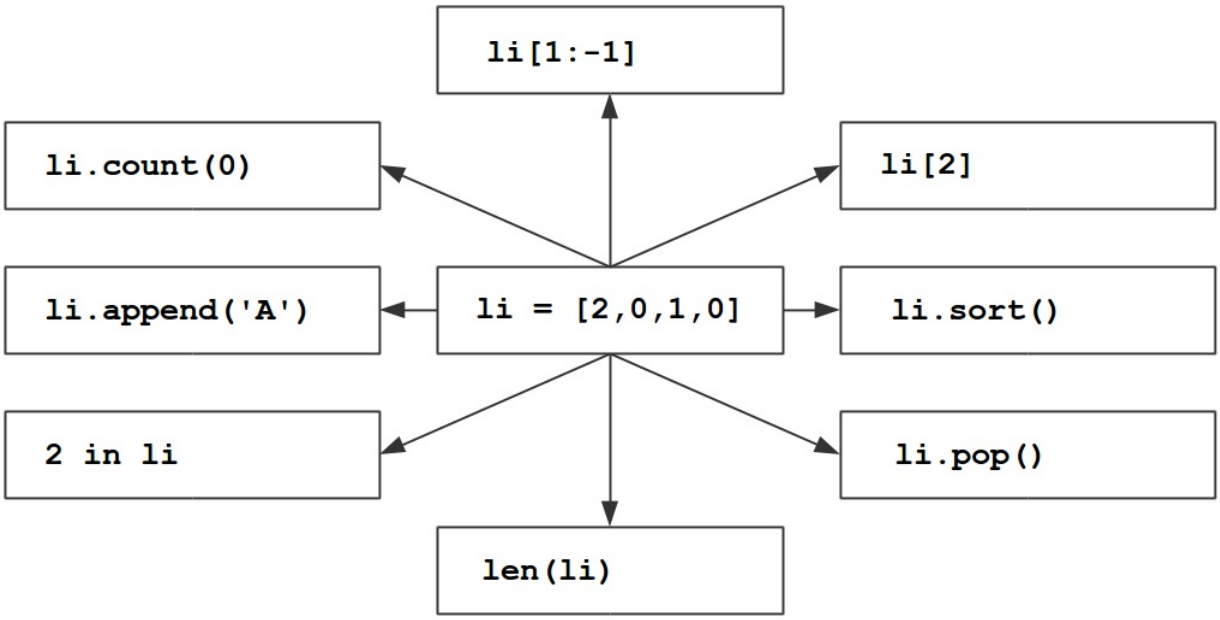
- Duplicate each character, so that `Emily` becomes `EEmmiillyy`.

Storing lists of items

To handle larger amounts of data, we cannot invent a new variable for every new data item. Somehow we need to put more data into one variable. This is where Python lists come in.

Warming up

Find out what each of the expressions does to the list in the center.



Exercises

Exercise 1

What does the list b contain?

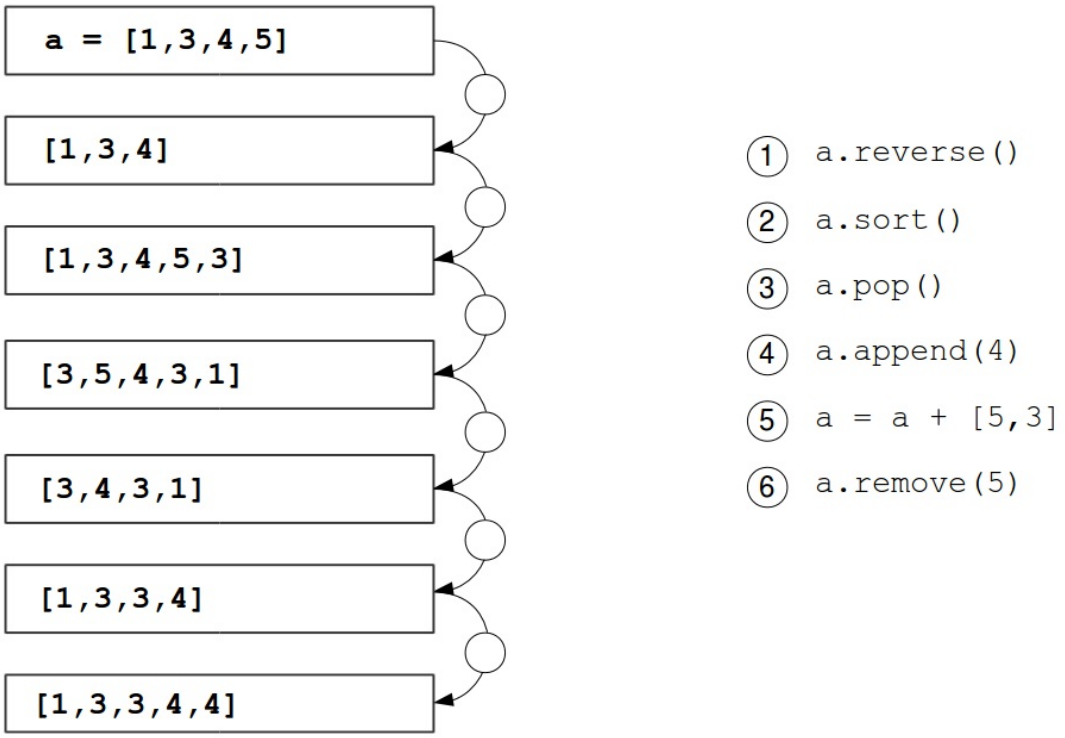
```
a = [8, 7, 6, 5, 4]
b = a[2:4]
```

- [ ] `[7, 6, 5]`
- [ ] `[7, 6]`

- [ ] `[6, 5]`
- [ ] `[6, 5, 4]`

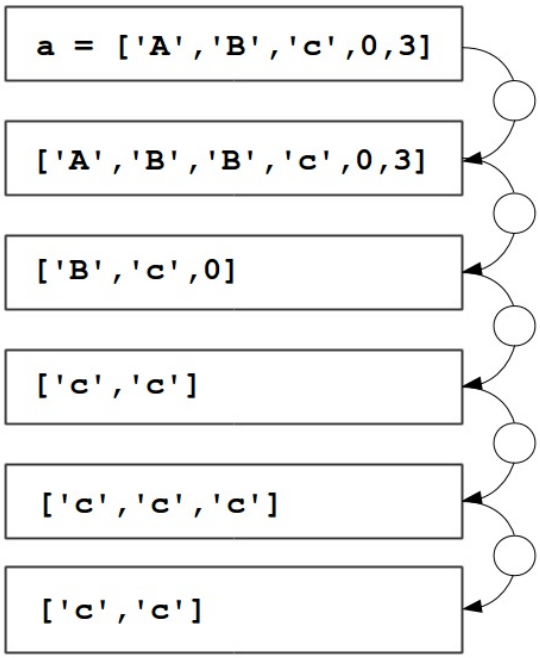
Exercise 2

Use the expressions to modify the list as indicated. Use each expression once.



Exercise 3

Use the expressions to modify the list as indicated. Use each expression once.



1. `a = a[2:5]`
2. `a = [a[-2]] + [a[1]]`
3. `a = a[:2]`
4. `a = [a[-1]]*3`
5. `a = a[:2] + a[1:]`

The Challenge: Percentage of top 10 names

In the year 2000, a total of 1962406 boys were registered born in the U.S. What percentage of the total names do the top 10 names consist?

Write a program to calculate that number.

name	number
Jacob	34465
Michael	32025
Matthew	28569
Joshua	27531
Christopher	24928
Nicholas	24650
Andrew	23632
Joseph	22818
Daniel	22307
Tyler	21500

Hint

If you are using Python2, you need to specify whether you want numbers with decimal places when dividing. That means

```
3 / 4
```

gives a different result than

```
3.0 / 4
```

However, Python 3 automatically creates the decimal places if needed.

Making decisions

The last missing piece in our basic set of commands is the ability to make decisions in a program. This is done in Python using the `if` command.

Warming up

Add your favourite movie to the following program and execute it:

```
movie = input('Please enter your favourite movie: ')
if movie == 'Star Trek':
    print('Live long and prosper')
elif movie == 'Star Wars':
    print('These are not the droids you are looking for')
elif movie == 'Dirty Dancing':
    print('I fetched the melons')
else:
    print("Sorry, I don't know the movie %s." % (movie) )
```

Exercises

Exercise 1

Which of these `if` statements are syntactically correct?

- [ ] `if a and b:`
- [ ] `if len(s) == 23:`
- [ ] `if a but not b < 3:`
- [ ] `if a ** 2 >= 49:`
- [ ] `if a != 3`
- [ ] `if (a and b) or (c and d):`

Exercise 2

Write a program that lets the user enter a number on the keyboard. Find the number in the list that is closest to the number entered and write it to the screen.

```
data = [1, 2, 5, 10, 20, 100, 200, 500, 1000]
query = int(input())
```

The Challenge: Filter a list

You have a list of the top 20 girls names from 2000:

```
['Emily', 'Hannah', 'Madison', 'Ashley', 'Sarah',
'Alexis', 'Samantha', 'Jessica', 'Elizabeth', 'Taylor',
'Lauren', 'Alyssa', 'Kayla', 'Abigail', 'Brianna',
'Olivia', 'Emma', 'Megan', 'Grace', 'Victoria']
```

Write a program that prints all names that start with an `A`.

Extra challenge

- count the number of names starting with `A` and print that number as well.

Reading and writing files

In this section, you will learn to read and write files and to extract information from them.

Goals

- read text files
- write text files
- extract information from a file
- list all files in a directory

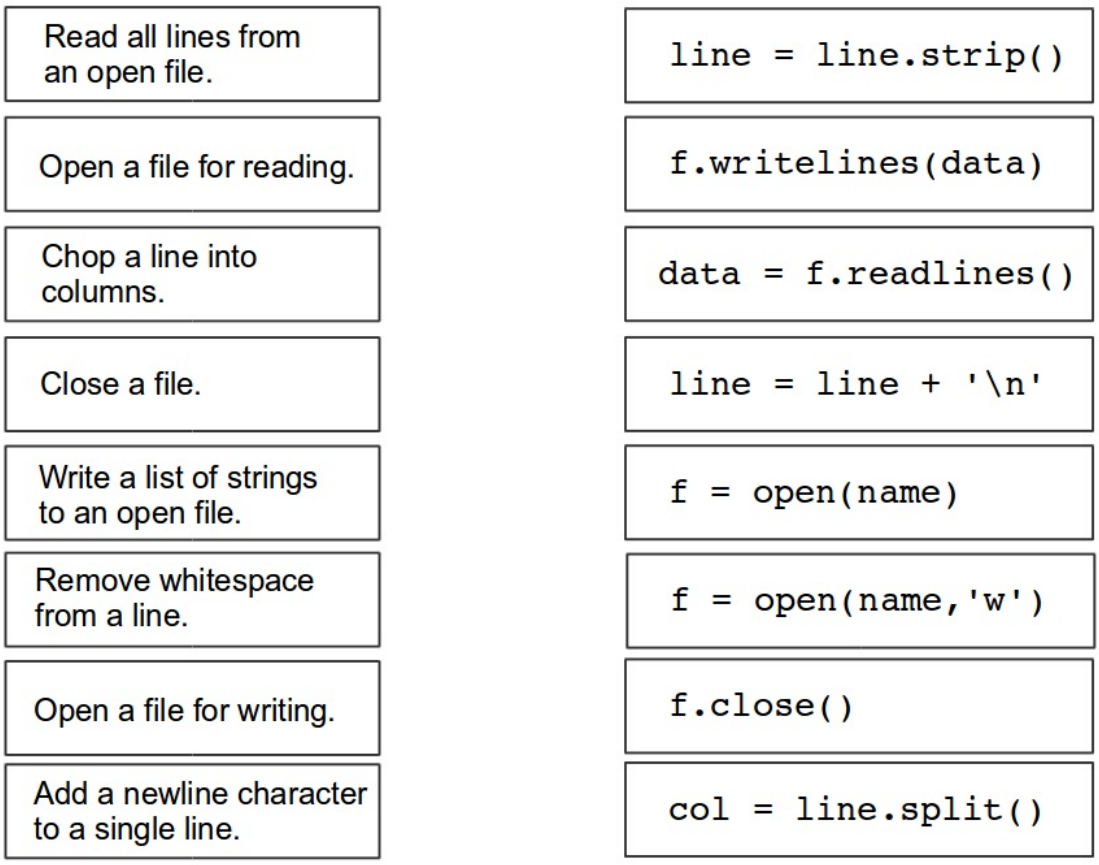
The Dataset of Baby Names

For the next exercises, you will need the complete archive of baby names. You can download the files from http://www.ssa.gov/oact/babynames/limits.html.

Reading text files

Warming up

Match the descriptions with the Python commands.



Exercise 1:

Make the program work by inserting `close` , `line` , `yob2014.txt` , `print` into the gaps.

```
f = open(___)
for ____ in f:
    ____(line)
f.____()
```

Exercise 2

Write a program that counts the number of names in the file `yob2014.txt` from the dataset of baby names.

Exercise 3

Execute the following program. What does it calculate?

```
boys = 0
for line in open('yob2014.txt'):
    if ",M," in line:
        boys = boys + 1
```

Exercise 4

How many different names starting with an `M` were there in 2014?

Exercise 5

How many different *girls* names starting with an `M` were there in 2014?

Writing text files

Warming up

Execute the following program. Explain what happens.

```
names = ['Adam', 'Bob', 'Charlie']

f = open('boy_names.txt', 'w')
for name in names:
    f.write(name + '\n')
f.close()
```

Remove the `+ '\n'` from the code and run the program again. What happens?

Exercise 1

Which are correct commands to work with files?

- [ ] `for line in open(filename):`
- [ ] `f = open(filename, 'w')`
- [ ] `open(filename).writelines(out)`
- [ ] `f.close()`

The Challenge: Write a table

Given is the following data:

```
names = ["Emma", "Olivia", "Sophia", "Isabella",
         "Ava", "Mia", "Emily"]
numbers = [20799, 19674, 18490, 16950,
           15586, 13442, 12562]
```

Write a program that writes all names into a single text file.

Extra Challenges

- Write each name into a separate line.
- Write the numbers into the same file.
- Write each number into the same line as the corresponding name.

Extracting information from a file

Rarely you can use the information from a file directly. Most of the time you will need to *parse* it, that is extracting relevant information.

Parsing means for instance:

- dividing tables into columns
- converting numbers to integers or floats.
- discarding unnecessary lines.

Warming up

Create a text file in a text editor. Write the following line there:

```
Alice Smith;23;Macroeconomics
```

Save the file with the name `alice.txt`.

Then run the following program:

```
f = open('alice.txt')
print(f)
text = f.read()
print(text)
columns = text.strip().split(';')
print(columns)
name = columns[0]
age = int(columns[1])
studies = columns[2]
print(name)
print(age)
print(studies)
```

What happens?

Exercises

Exercise 1

What does the following line produce?

```
"Take That".split('a')
```

Exercise 2

Create a text file with the contents:

```
Alice Smith;23;Macroeconomics
Bob Smith;22;Chemistry
Charlie Parker;77;Jazz
```

Write a program that reads all names and puts them into a list. Print the list.

Exercise 3

Collect the ages into a separate list.

Exercise 4

Collect the occupations into a separate list.

Exercise 5

Leave the `strip()` command away from the above program. What happens?

The Challenge: Find Your name

Write a program that finds out how often your name occurs in the list of 2014 baby names and print that number.

Extra Challenge:

- Calculate the total number of babies registered in 2014.

Working with directories

To process bigger amounts of data, you will need to work on more than one file. Sometimes you don't know all the files in advance.

Warming up

Fill in the gaps

The Python module [ ] is very useful for interactions with the operating system. In fact, it is a combination of two modules: os and [ ]. Before any of them can be used, the modules need to be activated by the [ ] statement.
Among the most frequently used operations is the [ ] function, that returns a list of all files in the given directory. If a program already has a filename, but it needs to be checked whether the file really exists, the function [ ] will return **True** or **False**. If a file needs to be deleted, this can be done using [ ].
A very useful feature of the **os.path** module is that it helps operating with directory names. Probably the most frequently used function is [ ], that separates a file name from directory names.
But **os** can do even more: You can use any shell command from a Python program with [ ]. However, this method has disadvantages: it depends on the operating system, and is a potentially insecure.

1. `os.access(fn,os.F_OK)`
2. `os.remove(filename)`
3. `os.path`
4. `os.listdir()`
5. `os.system(command)`
6. `os.path.split(os.getcwd())`
7. `import os`
8. `os`

Exercise 1

Explain the following code:

```
import os
for dirname in os.listdir('.'):
    print(dirname)
```

Exercise 1

Write a program that counts the number of files in the unzipped set of baby names. Have the program print that number.

Verify that the number is correct.

Exercise 2

How many entries (lines) does the entire name dataset have?

Hint: Generate a message that tells you which file the program is reading.

Exercise 3

Write a program that finds the *most frequently occuring name* in each year and prints it.

The Challenge

Find and print your name and the according number in each of the files, so that you can see how the number changes over time.

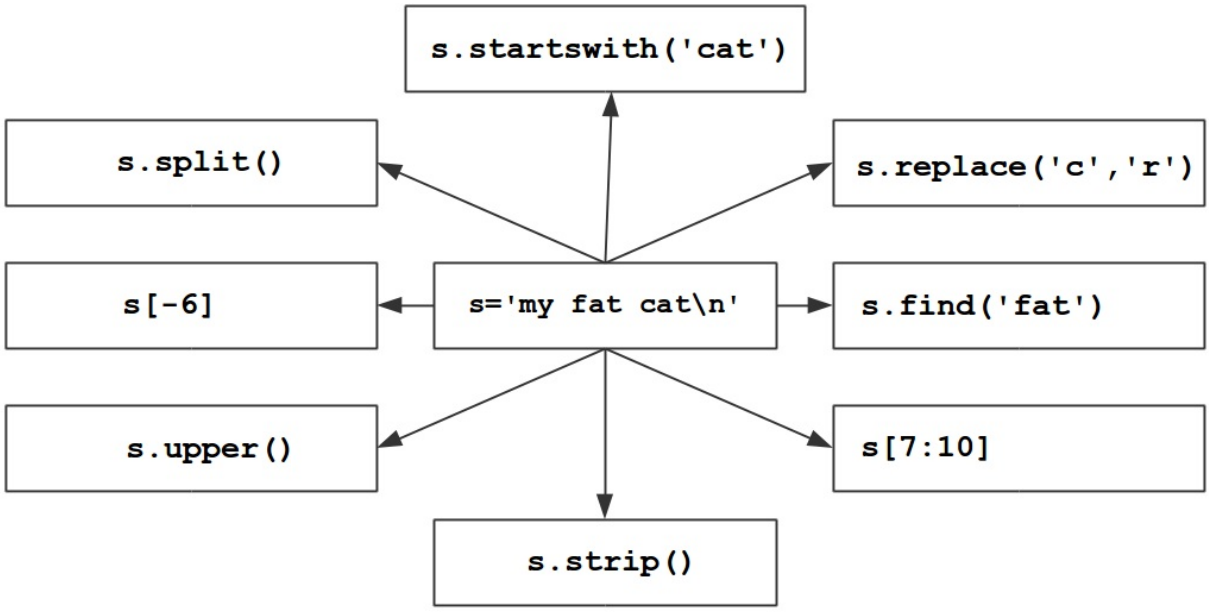
Functions

Python 3.5 has 72 builtin functions. To start writing useful programs, knowing about 25 of them is sufficient. Many of these functions are useful shortcuts that make your programs shorter.

String methods

Exercise

Find out what each of the expressions does to the string in the center.



Definitions

String methods

Every string in Python brings a list of functions to work with it. As the functions are contained within the string they are also called **methods**. They are used by adding the `.` to the string variable followed by the method name.

Below you find a few of the available methods:

Changing case:

```
s = 'Manipulating Strings '
s.upper()
s.lower()
```

Removing whitespace at both ends:

```
s.strip()
```

Cutting a string into columns:

```
s.split(' ')
```

Searching for substrings:

```
s.find('ing')
```

The method returns the start index of the match. The result -1 means that no match has been found.

Replacing substrings:

```
s.replace('Strings','text')
```

Removing whitespace at both ends:

```
s.startswith('Man')
s.endswith('ings')
```

Introspection

Warming up

Try the following on the interactive shell:

```
import random
dir(random)
help(random.choice)
name = random.choice(['Hannah', 'Emily', 'Sarah'])
type(name)
```

What does the code do?

The Challenge: a Baby name generator

Write a program that will generate a random baby name from a list of possible names.

Use the Python module `random`

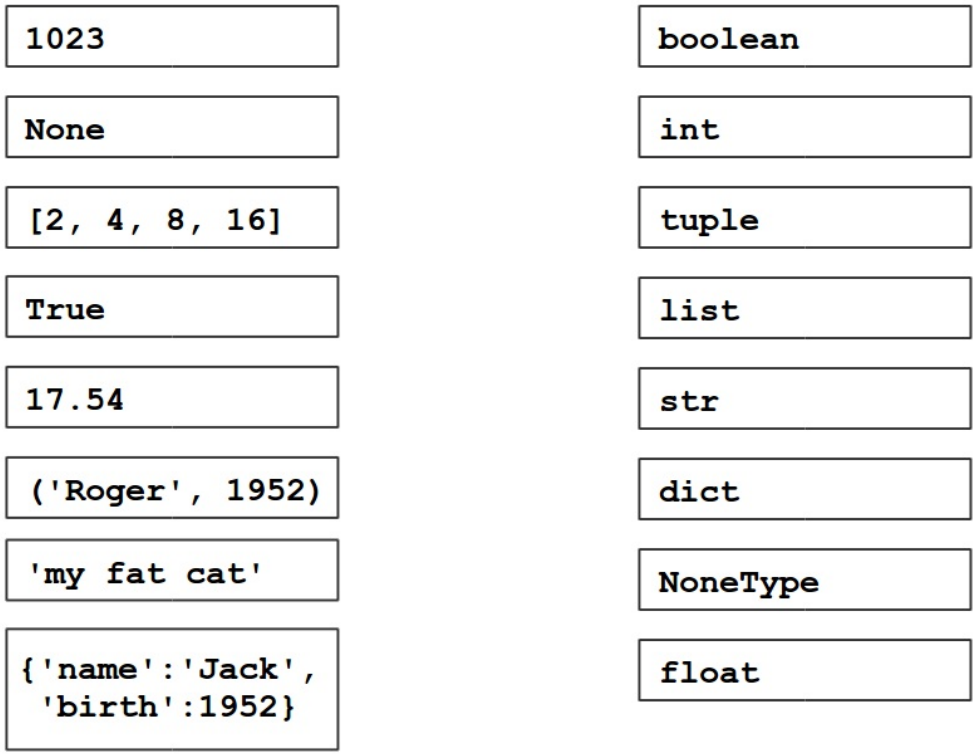
Extra challenges:

- let the user choose the gender of the babies.
- let the user enter how many babies they want to have.
- load baby names from a file.

Overview of Data types in Python

Data types

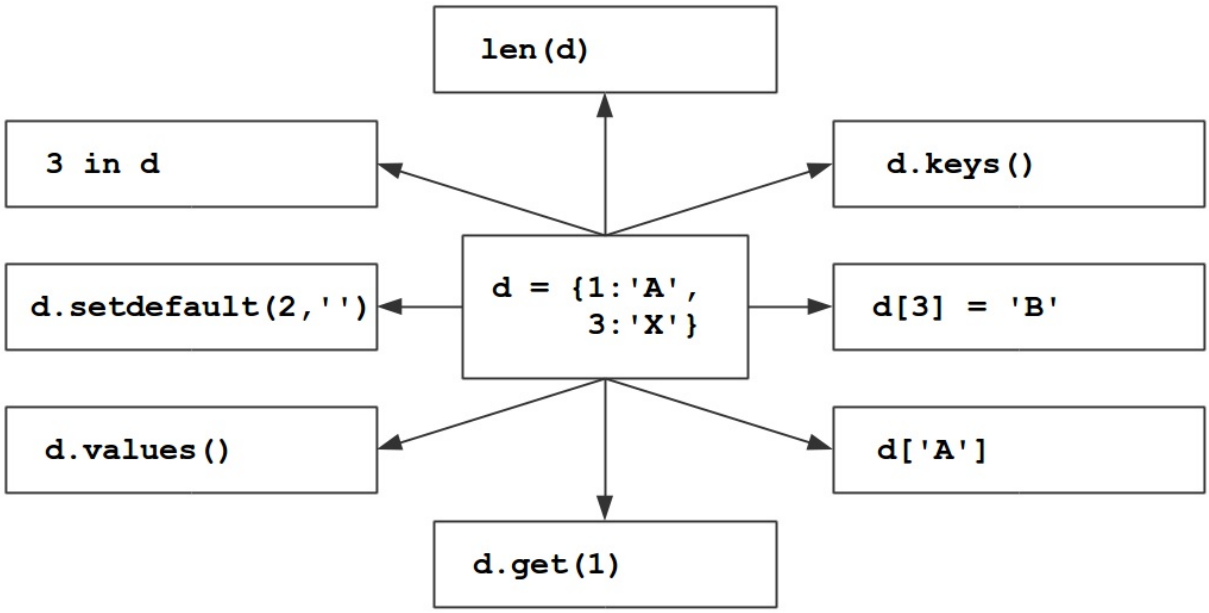
Match the data samples with their types.



Dictionaries

Using a dictionary

Find out what each of the expressions does to the dictionary in the center.



Definitions

Dictionaries

Dictionaries are an unordered, associative array. They have a set of key/value pairs. They are very versatile data structures, but slower than lists. Dictionaries can be used easily as a hashtable.

Creating dictionaries

```
prices = {
    'banana':0.75,
    'apple':0.55,
    'orange':0.80
    }
```

Accessing elements in dictionaries

By applying square brackets with a key inside, the values of a dictionary can be requested. Valid types for keys are strings, integers, floats, and tuples.

```
print prices['banana']  # 0.75
print prices['kiwi']    # KeyError!
```

Looping over a dictionary:

You can access the keys of a dictionary in a `for` loop. However, their order is not guaranteed, then.

```
for fruit in prices:
    print fruit
```

Methods of dictionaries

There is a number of functions that can be used on every dictionary:

Checking whether a key exists:

```
prices.has_key('apple')
```

Retrieving values in a fail-safe way:

```
prices.get('banana')
prices.get('kiwi')
```

Setting values only if they dont exist yet:

```
prices.setdefault('kiwi', 0.99)
prices.setdefault('banana', 0.99)
# for 'banana', nothing happens
```

Getting all keys / values:

```
print prices.keys()
print prices.values()
print prices.items()
```

Exercises

Exercise 1.

What do the following commands produce?

```
d = {1:'A', 'B':1, 'A':True}
print(d['A'])
```

- [ ] `False`
- [ ] `"B"`
- [ ] `True`
- [ ] `1`

Exercise 2.

What do these commands produce?

```
d = {1:'A', 'B':1, 'A':True}
print(d.has_key('B'))
```

- [ ] `1`
- [ ] `True`
- [ ] `"B"`
- [ ] `False`

Exercise 3.

What do these commands produce?

```
d = {1:'A', 'B':1, 'A':True}
print(d.values())
```

- [ ] `True`
- [ ] `['A', 1, True]`
- [ ] `3`
- [ ] `[1, 'B', 'A']`

Exercise 4.

What do these commands produce?

```
d = {1:'A', 'B':1, 'A':True}
print(d.keys())
```

- [ ] `[1, 'B', 'A']`
- [ ] `['A', 'B', 1]`
- [ ] `[1, 'A', 'B']`
- [ ] `The order may vary`

Exercise 5.

What do these commands produce?

```
d = {1:'A', 'B':1, 'A':True}
print(d['C'])
```

- [ ] `None`
- [ ] `'C'`
- [ ] `an Error`
- [ ] `False`

Exercise 6.

What do these commands produce?

```
d = {1:'A', 'B':1, 'A':True}
d.setdefault('C', 3)
print(d['C'])
```

- [ ] `3`

- [] `'C'`
- [] `None`
- [] `an Error`

Tuples

A tuple is a sequence of elements that cannot be modified. They are useful to group elements of different type.

```
t = ('bananas','200g',0.55)
```

In contrast to lists, tuples can also be used as keys in dictionaries.

Exercises

Exercise 1

Which are correct tuples?

- [ ] `(1, 2, 3)`
- [ ] `("Jack", "Knife")`
- [ ] `('blue', [0, 0, 255])`
- [ ] `[1, "word"]`

Exercise 2

What can you do with tuples?

- [ ] `group data of different kind`
- [ ] `change the values in them`
- [ ] `run a for loop over them`
- [ ] `sort them`

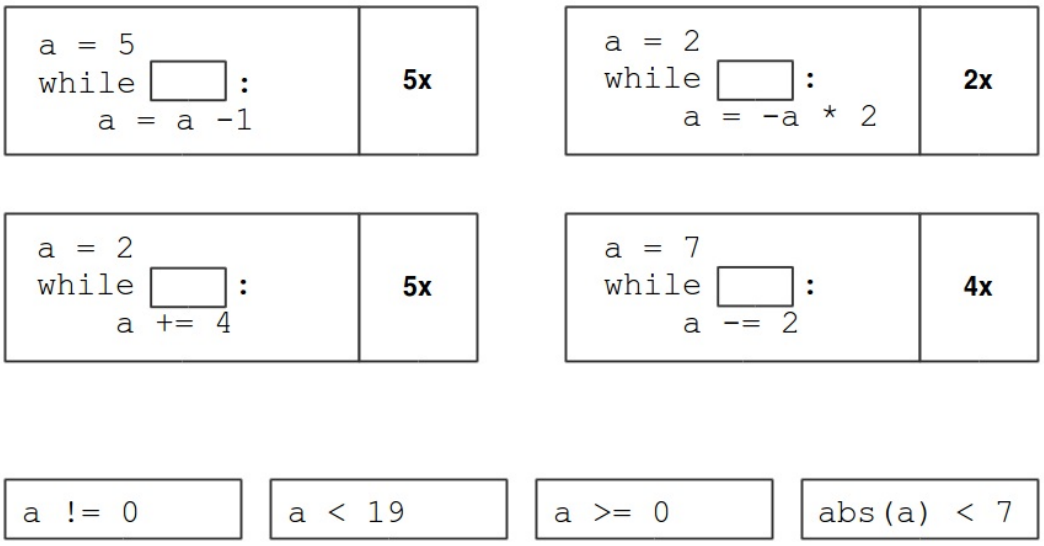
Control flow statements

Control flow means controlling, which instruction is handled next. In this tutorial, we cover three of Pythons control flow statements: `for`, `if` and `while`.

Conditional loops with while

Exercise

Match the expressions so that the `while` loops run the designated number of times.



While loops combine `for` and `if`. They require a conditional expression at the beginning. The conditional expressions work in exactly the same way as in `if.. elif` statements.

```
i = 0
while i < 5:
    print (i)
    i = i + 1)
```

When to use while?

- When there is an exit condition.

- When it may happen that nothing is done at all.
- When the number of repeats depends on user input.
- Searching for a particular element in a list.

Exercises

Exercise 1

Which of these `while` commands are correct?

- [ ] `while a = 1:`
- [ ] `while b == 1`
- [ ] `while a + 7:`
- [ ] `while len(c) > 10:`
- [ ] `while a and (b-2 == c):`
- [ ] `while s.find('c') >= 0:`

Exercise 2

Which statements are correct?

- [ ] `while` is also called a conditional loop
- [ ] The expression after `while` may contain function calls
- [ ] It is possible to write endless `While` loops
- [ ] The colon after `while` may be omitted
- [ ] The code block after `While` is executed at least once

Exercise 3

The following `for` loop searches for `33` in the data. Change the code so that it uses a `while` loop instead.

```
data = [5, 7, 33, 12, 4, 3, 18]

found = False
for n in data:
    if n == 33:
        found = True

print("The value 33 has been found: {}".format(found))
```

Exercise 4

The following `while` loop counts numbers higher than 10. Change the code so that it uses a `for` loop instead.

```
data = [4, 7, 11, 1, 3,  15]

i, j = 0,0
while i < len(data):
    if data[i] > 10:
        j += 1
    i += 1

print("Values higher than 10: {}".format(j))
```

Exercise 5

Will the following `while` loop finish?

```
count = 0
while count > 0:
    print count
    count += 1
```

Exercise 6

Will the following `while` loop finish?

```
text = "a"
while "z" not in text:
    text += "a"
```

Exercise 7

Will the following `while` loop finish? than a = 7 b = 135 while a != b: a += (a - b) / 10.0 b -= (a - b) / 10.0

Exercise 8

Will the following `while` loop finish?

```
n = 0
while n * 5 != n ** 2:
    n += 2
```

Exercise 9

Will the following `while` loop finish?

```
data = [1,2,7,8]
while data[-1] > 2:
    data.pop()
```

Exercise 10

Will the following `while` loop finish?

```
data = [2,3,15]
while data[0] < 100:
    data = data[1:]
```

Tables

Exercise 1:

Create an empty table of 10 x 10 cells.

Exercise 2:

Fill the table with the numbers from 1 to 100.

Exercise 3:

Save the table to a file.

Exercise 4:

Calculate the average number from the count column for a file with baby names for the year 2000 and print it.

Exercise 5:

Calculate the standard deviation as well.

Exercise 6:

Calculate how many girls' names and boys' names are there in total in 1900 and in 2000.

Exercise 7: Merge the files

Read all name files. Add an extra column for the year, so that you end up with a single big table with four columns. Save that table to a text file.

Structuring programs

In Python, you can structure programs on four different levels: with functions, classes, modules and packages. Of these, classes are the most complicated to use. Therefore they are skipped in this tutorial.

Goals

- Learn to write functions
- Learn to write modules
- Learn to write packages
- Know some standard library modules
- Know some installable modules

Modules

What is a module?

Any Python file (ending .py) can be imported from another Python script. A single Python file is also called a module.

Importing modules

To import from a module, its name (without .py) needs to be given in the import statement. Import statements can look like this:

```
import fruit
import fruit as f
from fruit import fruit_prices
from my_package.fruit import fruit_prices
```

It is strongly recommended to list the imported variables and functions explicitly instead of using the *import ** syntax. This makes debugging a lot easier.

When importing, Python generates intermediate code files (in the ***pycache*** directory) that help to execute programs faster. They are managed automatically, and dont need to be updated.

Modules and Packages

Exercises

Exercise 1

Join the right halves of sentences.

An optional parameter	tells about properties of anything in Python.
An import statement	cay return several values as a tuple.
A package	may run endlessly
The dir() function	is a directory with Python modules.
Every package	is best written on top of a file.
A function that calls itself	must contain a file __init__.py
A function in Python	must be written after obligatory parameters.
A function	may modify its own parameters.

Exercise 2

Which `import` statements are correct?

- [ ] `import re`
- [ ] `import re.sub`
- [ ] `from re import sub`
- [ ] `from re import *`
- [ ] `from re.sub import *`

Exercise 3

Where does Python look for modules to import?

- [ ] in the variable `sys.path`
- [ ] in the current working directory
- [ ] in the directory where the current module is
- [ ] in the `site-packages` folder
- [ ] in directories in the `PYTHONPATH` variable

Exercise 4

Which statements about packages are true?

- [ ] a package is a directory with modules
- [ ] a package must contain a file `__init__.py`
- [ ] a package may contain no code
- [ ] a module may contain many packages

Exercise 5

Which packages are installed by default?

- [ ] `os` - manipulating files and directories
- [ ] `time` - accessing date and time
- [ ] `csv` - read and write tables
- [ ] `numpy` - number crunching
- [ ] `pandas` - clever handling of tabular data

Additional Exercises

Exercise 1

Calculate the total number of births for the years 1900 and 2000.

Exercise 2

Read the Baby names from 1900 to 2000. How many different names per year are there? Create a bar plot with distinct bars for girls/boys.

Exercise 3

How many baby names are there in 1900 and 2000 beginning with `A` or `Z`, respectively?

Exercise 4

Create a plot showing how the last letters of names change over time.

Exercise 5

Schreibe ein Programm, das den Mittelwert der Daten in der Datei *datei_lesen/zahlen.txt* berechnet.

Exercise 6

Schreibe den Mittelwert in eine Datei.

Exercise 7

Berechne ausserdem die Standardabweichung. Verwende:

```
import math
wurzel = math.sqrt(wert)
```

Review Questions

TODO: sort basic/advanced questions apart

- is it equivalent to initialize a variable at the beginning of the program or inside a loop? (see average.py for an example)
- Name a situation in which you would prefer a programming language other than Python?
- How to write a program that prints all file names in a folder?
- What is an object and what is a class? Describe the difference?

How can you swap the values of two variables? How can you create a dictionary with some values inside? How do the methods get(), has_key() and keys() of a dictionary differ? How can you retrieve values from a dictionary when you do not know whether their keys exist? How can you set values in a dictionary? What is a list comprehension? Is it possible to create a for loop over a dictionary? How to sort values from a dictionary? What is the dir(object) built-in function good for? What does the getattr(object,name) built-in function return? How can i execute a string containing Python code created by a Python program? What can you do to make your programs run faster? What can i use to calculate with numbers? Write down at least 5 python commands. What can i do to manipulate strings? Write down at least 5 python commands. What can i do with lists? Write down at least 5 python commands. What can i do with dictionaries? Write down at least 5 python commands.

How can you: Swap the values of two variables (Python Cookbook 1.2) Create a dictionary (Python Cookbook 1.3) Retrieve values from a dictionary (Python Cookbook 1.4) Set values in a dictionary (Python Cookbook 1.5) Loop through a list I (Python Cookbook 1.14) Loop through a list II (Python Cookbook 1.15) Loop through a text file (Python Cookbook 4.2). Sort values from a dictionary (Python Cookbook 2.2) Sorting list of objects by an attribute (Python Cookbook 2.8) String handling (Python Cookbook 3.2-3.11) Test if a string can be converted to an integer (Python Cookbook 3.13) Write a text file (Python Cookbook 4.3) Replace text in a file (Python Cookbook 4.4)

You have an integer variable a, and four different functions that should be called depending on the value of a. How would you implement this? What does the assert statement do? What kinds of operations should be wrapped in a try.. except clause. How to create an exception on purpose? Write two different ways to calculate the numbers from 1 to 10. Write two different ways to call string.upper for each element of a list. What different kinds of arguments can a function definition contain? Which data types as function arguments are mutable, which are immutable? For how long does a local variable defined in a function

exist? What is a function variable good for? What are rules for good style of functions? When a module is importet by three other modules in one program, how often is its initialization code run? Name two different kinds of import statements. What is a package? Where does Python look for module files available for import?

Objects An object is a container for data plus code. Name three advantages of writing programs with objects. What is a class? What is an instance? How does a constructor method look like? What are class attributes, what instance attributes? Explain how inheritance works using the classes 'Plant', 'Vegetable', 'Carrot' and 'Cactus' and the methods 'grow()', 'hurt()' or 'eat()'? Data types and structures What kinds of values can the basic data types (boolean, integer, float and string) take? What is a type cast and how is it written in Python? What is the difference between a tuple and a list? Which values of these data types are equivalent to None? Which values of these data types are equal to a boolean True? Enumerate five useful things that you can do with lists. How to make a copy of an entire list? How to loop through all elements of a dictionary? How to check whether a dictionary has a certain key? What data types work as keys of a dictionary? What is the difference between a stack and a queue? How many leafs has a binary tree with 7 nodes? Give an example of a graph. Operations What is a floor division? How is it used in Python? Where to look for the sine, cosine and square root functions? What do programmers need to take care of when doing divisions? What needs to be done before using the arithmetic operators (+, -, *, /) on Python objects? Give an expression that checks whether a number is odd or even. Does the arithmetic (+, -, *, /) or the AND operator have the higher priority? What does the expression x<7 AND „small“ OR „big“ return? Is an expression always evaluated completely? How can you check whether a number variable is between two values? How to cut off the last two characters of a string? How to replace the letter A with B in a string? How to make a list of string variables from a tab-separated string? How to find out if a string starts with a decimal number? Enumerate some elements of the regular expression syntax. Are string functions or regular expressions faster? Give a line of code that writes a list of strings to a file. What parameters does the format string „%i str: %s %4.3f“ expect? How to read text from the keyboard? What do the escape characters \t and \n stand for? Propose three different ways to write data to a file.

Where in your programs should you put triple-quoted documentation strings? Which method is called if you convert an object to a string? What do you see when reading the **doc** string of a function, a module or an object?

What programs can you use to edit Python programs comfortably?

Name alternative Python interpreter(s)

What is a profiler?

What does EpyDoc do?

Name one thing you can do using a debugger, like pdb.

What is an object and what is a class? Describe the difference?

Background information on Python 3

What is Python?

- Python is an interpreted language.
- Python uses dynamic typing.
- Python 3 is not compatible to Python 2.x
- The Python interpreter generates intermediate code (in the **pycache** directory).

Strengths

- Quick to write, no compilation
- Fully object-oriented
- Many reliable libraries
- All-round language
- 100% free software

Weaknesses

- Writing very fast programs is not straightforward
- No strict encapsulation

What has changed from Python 2 to Python 3?

- print is now a function
- all Strings are stored in Unicode (better handling of umlauts and special characters in all languages)
- many functions like *zip(), map() and filter() return iterators*
- the standard library has been cleaned up completely

Recommended books and websites

Free Online Books

Books for beginners and people with programming experience:

- Learn Python the Hard Way - a bootcamp-style tutorial by **Zed Shaw**
- How to think like a Computer Scientist - a very systematic, scientific tutorial by **Allen B. Downey**
- Dive into Python 3 - explains one sophisticated program per chapter - **by Mark Pilgrim**

Paper books

- Managing your Biological Data with Python - **Allegra Via, Kristian Rother and Anna Tramontano**
- Data Science from Scratch - **Joel Grus**

Websites

- Main documentation and tutorial: http://www.python.org/doc
- Tutorial for experienced programmers: http://www.diveintopython.org
- Tutorial for beginners: http://greenteapress.com/thinkpython/thinkCSpy/html/
- Comprehensive list of Python tutorials, websites and books: http://www.whoishostingthis.com/resources/python/
- Python Library Reference covering the language basics: https://docs.python.org/3/library/index.html
- Global Module Index – description of standard modules: https://docs.python.org/3/py-modindex.html

Authors

© 2013 Kristian Rother (krother@academis.eu)

This document contains contributions by Allegra Via, Kaja Milanowska and Anna Philips.

License

Acknowledgements

I would like to thank the following people for inspiring exchange on training and Python that this tutorial has benefited from: Pedro Fernandes, Tomasz Puton, Edward Jenkins, Bernard Szlachta, Robert Lehmann and Magdalena Rother