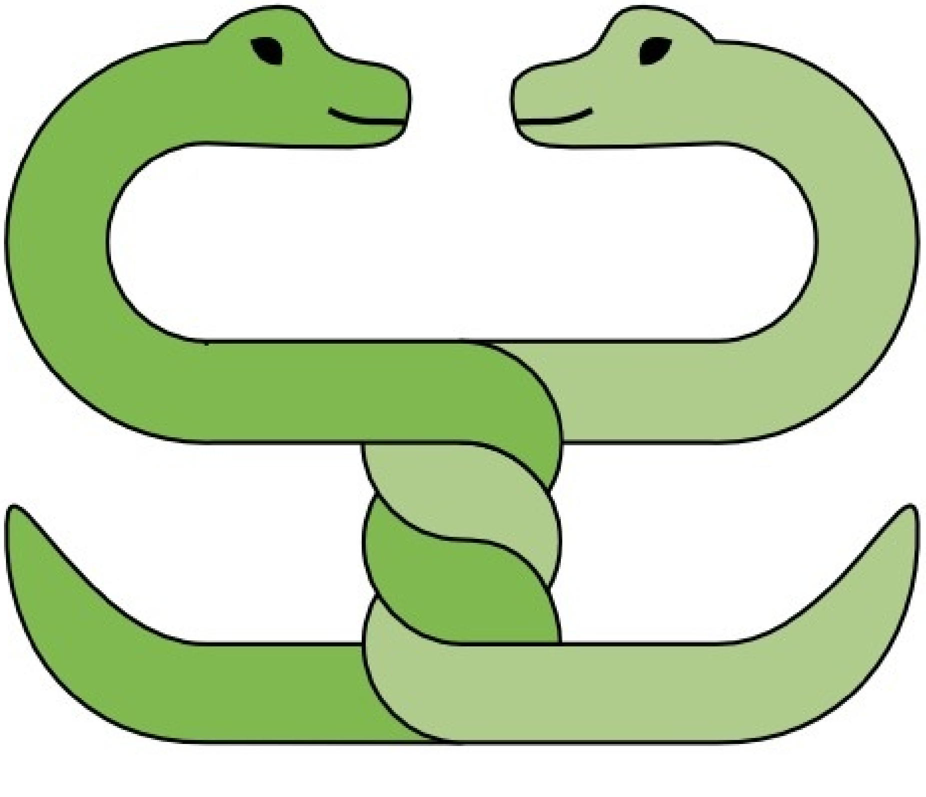
ACADEMIS

Python 3 Basics



tutorial

Table of Contents

Python 3 Basics Tutorial

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

with contributions by Allegra Via, Kaja Milanowska and Anna Philips

Sources of this document can be found on https://github.com/krother/Python3_Basics_Tutorial

Introduction

Who is this tutorial for?

This is a tutorial for novice programmers. You are the learner I had in mind when writing this tutorial if:

- you have worked a little with a different programming language like R, MATLAB or C.
- you have no programming experience at all
- you know Python well and would like to teach others

This tutorial works best if you follow the chapters and exercises step by step.

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.

If you are an experienced programmer

If you are fluent in any programming language, this tutorial might be very easy for you. Of course, you can work through the exercises to get the Python syntax into your fingers. However, this tutorial contains very little material on the higher abstraction levels in Python, like classes, namespaces or even functions.

For a tutorial for non-beginners, I recommend the following free online books:

- 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**

Installing Python

The first step into programming is to get Python installed on your computer. You will need two things

- Python itself
- a text editor

Which to install, depends on your operating system.

On Ubuntu Linux

By default, Python is already installed. In this tutorial however, we will use Python3 whenever possible. You can install it from a terminal window with:

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

To check whether everything worked, type:

```
ipython3
```

As a text editor, it is fine to start with **gedit**. Please make sure to change tabs to spaces via *Edit -> Preferences -> Editor -> tick 'Insert spaces instead of tabs'*.

On Windows

A convenient way to install Python, an editor and many additional packages in one go is **Anaconda**, a Python distribution with many pre-installed packages for scientific applications.

After installing, you will need to launch the **Spyder** editor from the Start menu.

Other Python environments

- **Python 3** - the standard Python installation
- **Canopy** - another Python instllation for scientific purposes.
- **Idle** - a standard editor installed with Python by default.
- **Sublime Text** - a very powerful text editor for all operating systems
- **Notepad++** - a powerful text editor for Windows. *Please do not use the standard Notepad. It won't get you anywhere.*
- **PyCharm** - a professional Python development environment capable of handling large projects. You won't need most of the functionality for a long time, but it is a well-written editor.
- **vim** - a console-based text editor for Unix systems. The tool of choice for many system administrators.

Questions

Question 1

Which text editors are installed on your system?

Question 2

Which Python version are you running?

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

Exercise 1:

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).

Exercise 2:

Which operations result in 8?

- `0 + 8`
- `4 4`
- `8 /`
- `65 // 8`
- `17 % 9`
- `2 * * 4`
- `64 ** 0.5`

Exercise 3:

Collect the Python operators you already know in a table (`+` , `-` , `*` etc.).

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*.

Exercise 1:

Let's define some variables. Fill in the gaps:

```
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]: ______
```

Exercise 2:

Let's change the content. Insert the correct values and variable names into the blanks.

```
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
```

If you like maths, you may sigh loud at the notion of redefining a variable every few lines.

Exercise 3:

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

```
Sarah
ASHLEY
madison alexis
sam90
2000jessy
liz_lauren
alyssa.kay
```

Exercise 4:

Which are correct variable assignments?

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

Storing text

Exercise 1:

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

```
In [1]: first = 'Emily'
In [2]: last = "Smith"
In [3]: name = first + " " + last
In [4]: name
```

Exercise 2:

What do the following statements do?

```
In [5]: name[0]
In [6]: name[3]
In [7]: name[-1]
```

Exercise 3:

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

```
Ä á @ ? & * \ / " '
```

Exercise 4:

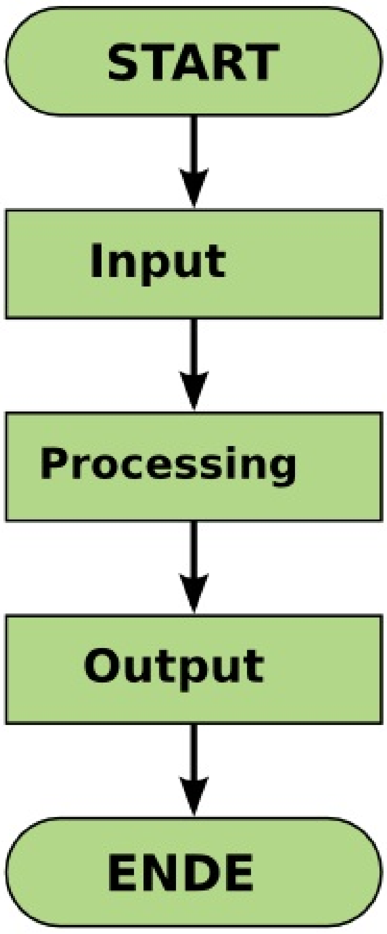
What does `name` contain after the following statements? Explain the code.

```
In [8]: name = ""
In [9]: anna = "Anna"
In [10]: name = anna[0] + name
In [11]: name = anna[1] + name
In [12]: name = anna[2] + name
In [13]: name = anna[3] + name
In [14]: name
```

Writing Python programs

A program consists of multiple lines that are executed in one go.

Usually a program contains the following sections:



Of course, programs can grow a lot more complicated than that.

In this chapter, we will learn to know the `print()` function for producing output, loops and branching statements.

Turing Completeness

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.

My first program

Using the interactive IPython shell 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 this section we will write our first Python program. It is going to simply write a few names of babies to the screen.

Exercise 1

Open a text editor (e.g. Spyder) and create a new file. Write into it:

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

Save the file with the name `output.py` ab.

Exercise 2

Now let's execute our program.

- In **Anaconda Spyder** you can use the *"Play"* button or press `F5`.
- In **Unix** open a terminal, change to the directory with the Python file and type:

  python3 output.py

Exercise 3

Explain the following program:

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

Exercise 4

Write the following program:

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

What happens when you execute the code?

Repeating instructions

In our early programs, each Python instruction was executed only once. That makes programming a bit pointless, because our programs are limited by our typing speed.

In this section we will take a closer look at the `for` statement that repeats one or more instructions several times.

Exercise 1

What does the following program do?

```
for number in range(1, 43):
    print(number)
```

Exercise 2

What advantages does the `for` loop have over the following one?

```
print(0)
print(1)
print(2)
print(3)
print(4)
..
```

Exercise 3

Write a for loop that creates the following output

```
1
4
9
16
25
36
49
```

Exercise 4

Explain the difference between the following two programs:

```
total = 0
for number in range(10):
    total = total + number
    print(total)
```

and

```
total = 0
for number in range(10):
    total = total + number
print(total)
```

Exercise 5

What does the following program do?

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

Exercise 6

Write a program that calculates the number of characters in `Stefani Joanne Angelina Germanotta`. **Spaces count as well!**

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.

Exercise 1

Execute the following program and explain its output.

```
number = 123

if number > 1000:
    print("The number is larger than 1000.")
elif number == 1000:
    print("The numer is exactly 1000.")
else:
    print("The number is smaller than 1000.")
```

Exercise 2

Set `name` to such a value that one, two or all three conditions apply.

```
name = ____

if "m" in name:
    print("There is a 'm' in the name.")
    if name != "Mimi":
        print("The name is not Mimi.")
        if name[0] == "M" and name[-1] == "m":
            print("The name starts and ends with m.")
```

Exercise 3

The following program writes the positions of all letters *"n"* in the name to the screen. Unfortunately, it contains **three errors**. Make the program execute correctly:

```
name = "Anna"
position = 1

for char in name
    if char = "n":
        print(position)
position = position + 1
```

Exercise 4

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):`

Overview of Data types in Python

Data types

Match the data samples with their types.

`1023`	`boolean`
`None`	`int`
`[2, 4, 8, 16]`	`tuple`
`True`	`list`
`17.54`	`str`
`('Roger', 1952)`	`dict`
`'my fat cat'`	`NoneType`
`{'name':'Jack', 'birth':1952}`	`float`

Definitions

Immutable and mutable data types

In Python there are basic and composite data types. The values of basic data types cannot be changed, they are **immutable**. Most of the composite data types are **mutable**.

The immutable data types in Python are:

- Boolean (`True` / `False`)
- Integer (`0` , `1` , `-3`)
- Float (`1.0` , `-0.3` , `1.2345`)
- Strings (`'apple'` , `"banana"`) - both single and double quotes are valid
- None (aka an empty variable)
- Tuples (multiple values in parentheses, e.g. `('Jack', 'Smith', 1990)`)

The mutable data types are

- List [1, 2, 2, 3]

- Dictionary {'name': 'John Smith', 'year': 1990}
- Set ()

Type conversions

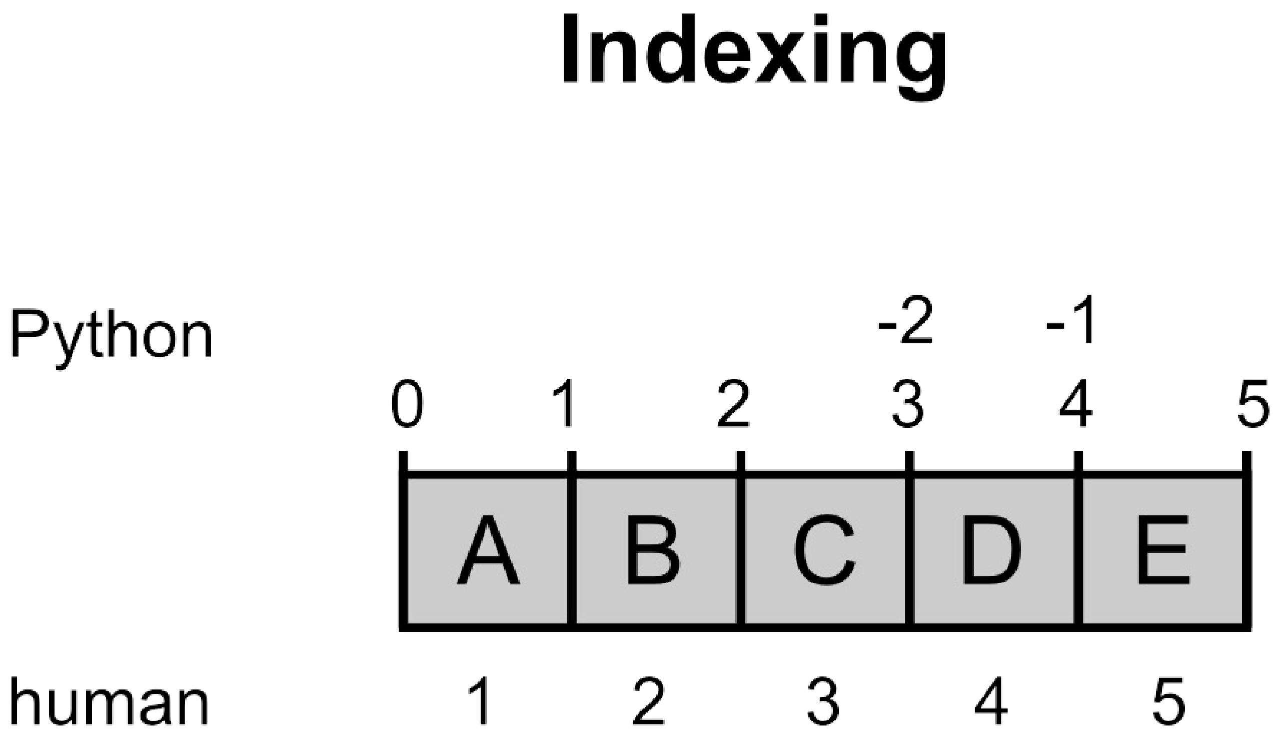
Values can be converted into each other using *conversion functions*. Try the following:

```
int('5.5')
float(5)
str(5.5)
list("ABC")
tuple([1,2,3])
dict([('A',1),('B',2)])
set([1,2,2,3])
```

Lists

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

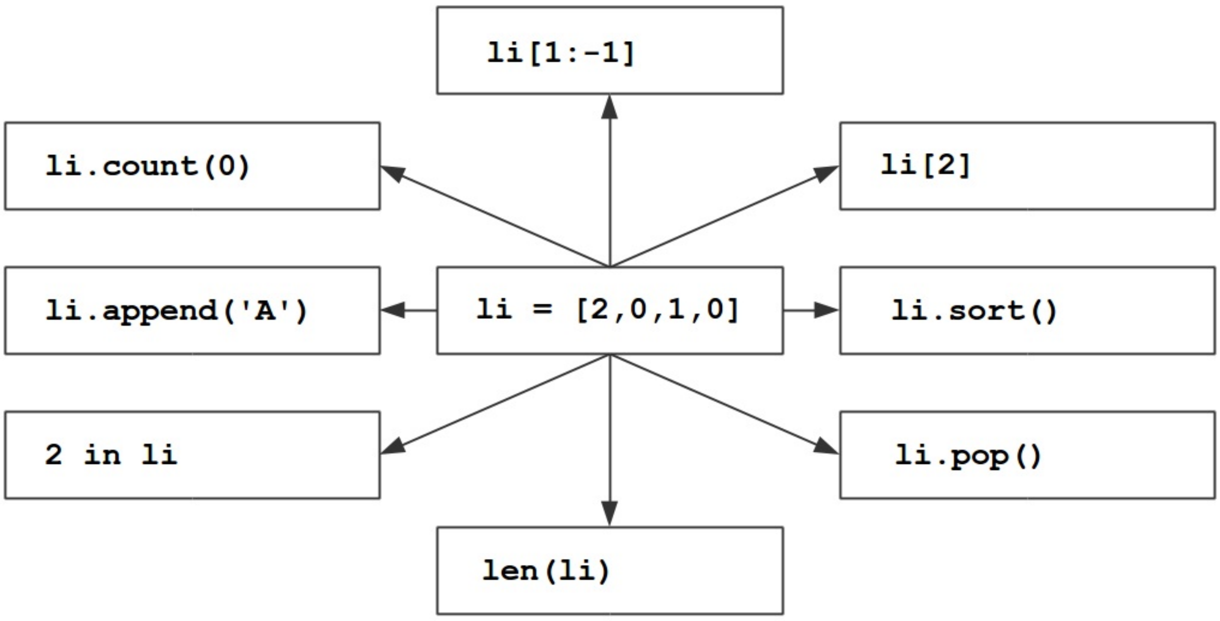
However, Python counts a bit different than humans do:



Creating lists

Exercise 1

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



Exercise 2

What does the following program do?

```
top8 = [34465, 32025, 28569, 27531, \
        24928, 23632, 22818, 22307]

for value in top8:
    print(value)
```

Exercise 3

How many babies are there in total? Write a program that calculates that number.

Exercise 4

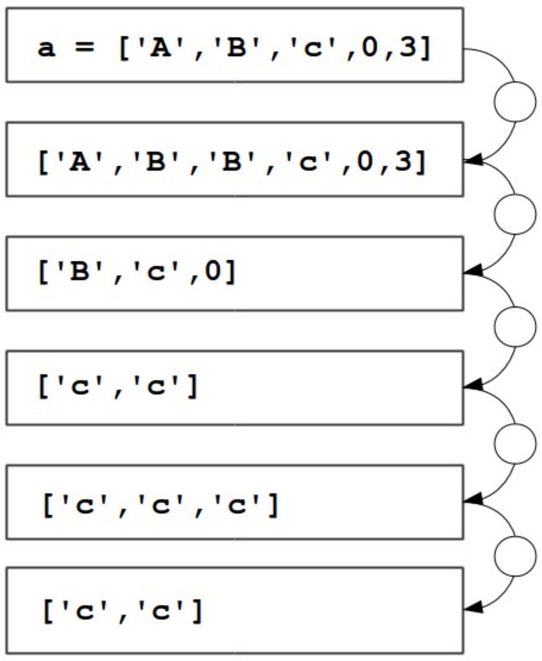
You have a list of the 20 most popular girls names from the year 2000:

```
['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 starting with `'A'` or `'M'`.

Exercise 5

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:]`

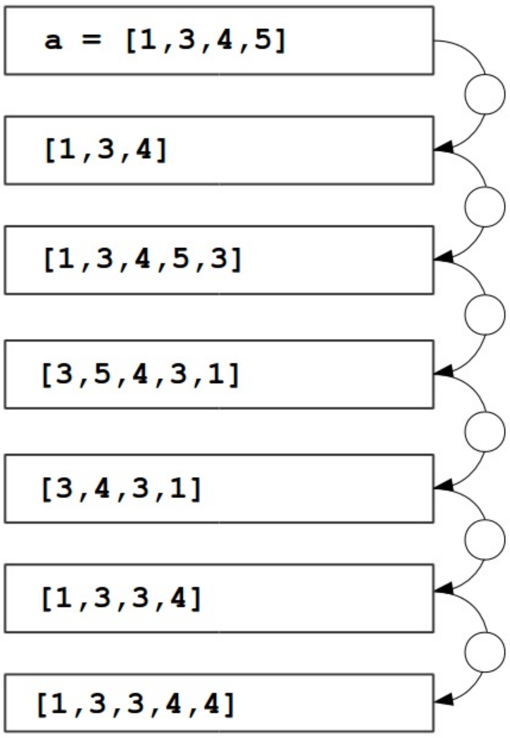
Exercise 6

Create a new list containing the sum of *California* and *New York* for each name.

```
names = ["Emily", "Amy", "Penny", "Bernadette"]
california = [2269, 542, 54, 21]
new_york = [881, 179, 12, 11]
```

Exercise 7

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



1. `a.reverse()`
2. `a.sort()`
3. `a.pop()`
4. `a.append(4)`
5. `a = a + [5,3]`
6. `a.remove(5)`

Shortcuts

Exercise 1

Simplify the following code using the function `sum()` :

```
counts = [356, 412, 127, 8, 32]

total = 0
for number in data:
    total = total + number
print(total)
```

Exercise 2

Simplify the following code using the function `range()` :

```
i = 0
while i < 10:
    print(i * '*')
    i += 1
```

Exercise 3

Simplify the following code using the function `zip()` :

```
names = ['Lilly', 'Lily', 'Leila', 'Lilja', 'Lillie']
counts = [356, 412, 127, 8, 32]

table = []
i = 0
while i < len(names):
    row = (names[i], counts[i])
    table.append(row)
    i += 1
print(table)
```

Exercise 4

Simplify the following code using the function `enumerate()` :

```
names = ['Lilly', 'Lily', 'Leila', 'Lilja', 'Lillie']

i = 0
for name in names:
    print(i, name)
    i += 1
```

Exercise 5

Use `list(range())` to create the following lists:

- `[4, 7, 9, 12]`
- `[10, 20, 30, 40, 50]`
- `[33, 32, 31, 30]`

Exercise 6

On which data types does the `len()` function work?

- lists
- dictionaries
- strings
- floats
- sets

Tables

Data frequently occurs in the form of tables. To process tables in Python, it helps to know that we can put lists in other lists. These are also called **nested lists**.

A simple table in Python looks like this:

```
# columns: Name, #California, #New York
[
  ["Emily", 2269, 881],
  ["Amy", 542, 179],
  ["Penny", 54, 12],
  ["Bernadette", 21, 11]
]
```

In this chapter we will deal with creating and processing nested lists.

Exercise 1

Write all rows of the above table to the screen with a `for` loop.

Exercise 2

Write all *cells* of the table to the screen with a double `for` loop.

Exercise 3

Create an empty table of 10 x 10 cells and fill them with numbers from 1 to 100.

Exercise 4

Sort the above table by the second column. Use the following code sniplet:

```
from operator import itemgetter
tabelle.sort(key=itemgetter(col_index))
```

Reading data from files

In this section, you will learn to extract information from simple text files.

For the next exercises, you will need the complete archive of baby names (the shorter one not grouped by states). You can download the files from http://www.ssa.gov/oact/babynames/limits.html.

Reading simple text files

Exercise 1

Create a text file `bigbang.txt` in a text editor, containing the following data:

```
Emily,F,12562
Amy,F,2178
Penny,F,342
Bernadette,F,129
Leonard,M,384
Howard,M,208
Sheldon,M,164
Stuart,M,82
Raj,M,41
```

Exercise 1:

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

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

Hint:

Depending on your editor, you may need to insert the complete path to the file. If the program does not work, a wrong file name or location are the most probable reasons.

Exercise 3

How many different *girls names* were there in 2015?

Sort the following code lines and **indent them correctly**:

```
girls = 0

if "B" in line:

print(girls)

if ",F," in line:

for line in open('names/yob2015.txt'):

girls += 1
```

Exercise 4

Extend the program from the previous exercise such that boys and girls names are counted separately.

Exercise 5

Which of the following commands are correct?

- `for char in "ABCD":`

- `for i in range(10):`
- `for number in [4, 6, 8]:`
- `for k in 3+7:`
- `for (i=0; i<10; i++):`
- `for var in open('bigbang.txt'):`

Exercise 6

Write a program that reads lines from the file `yob2015.txt`. Identify all lines containing your name and print them to the screen.

Parsing data from text

Most text files contain both text and numbers. Extracting these data fields from a file and storing them in reasonably *structured* variables is called **parsing**. To parse files, we need methods of strings, lists and **type conversions**.

Exercise 1

Insert the following pieces into the code, so that all commands are executed correctly: `age` , `int(age)` , `name` , `str(born)` , `2000`

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

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

Exercise 2

The following program collects names in a list that occur at least 10000 times. Unfortunately, the program contains **four errors**. Find and fix these.

```
frequent = []

for line in open('names/yob2015.txt'):
    columns = line.strip().split(',')
    name = colums[1]
    anzahl = int(columns[3])
    if anzahl >= 10000
        frequent.append(name)

print(frequent)
```

Exercise 3

Write a program that calculates the total number of babys for the year 2015 and writes it to the screen. Compare that number with the year 1915.

Exercise 4

Write a program that finds the *three most frequent* names for boys and girls in a given year and writes them to the screen.

Hint: The three most frequent names are on top of the list.

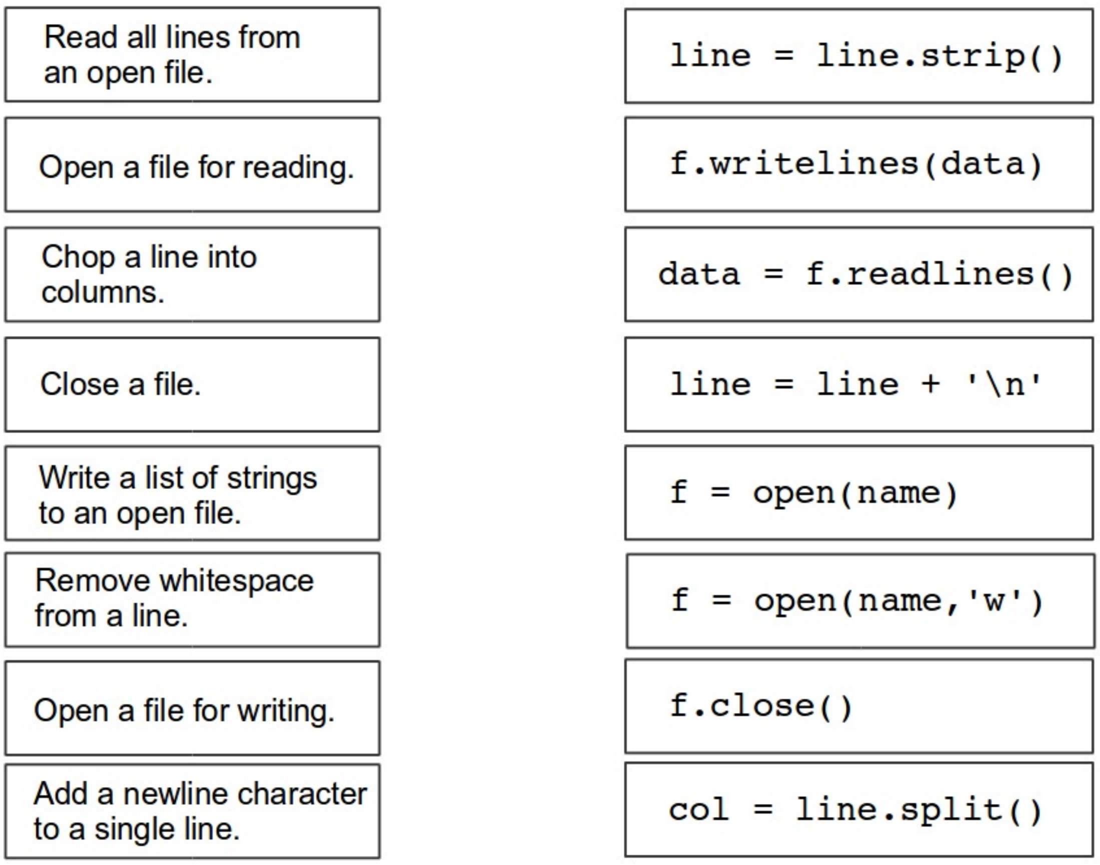
Exercise 5

Write a program that calculates the percentage of the 10 most frequent names for the year 2015 and writes it to the screen.

Writing Files

Exercise 1

Form pairs of Python commands and their meanings.



Read all lines from an open file.	`line = line.strip()`
Open a file for reading.	`f.writelines(data)`
Chop a line into columns.	`data = f.readlines()`
Close a file.	`line = line + '\n'`
Write a list of strings to an open file.	`f = open(name)`
Remove whitespace from a line.	`f = open(name,'w')`
Open a file for writing.	`f.close()`
Add a newline character to a single line.	`col = line.split()`

Exercise 2

Execute the following program. Explain what happens.

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

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

Exercise 3

Remove the `+ '\n'` from the program and execute it again. What happens?

Exercise 4

Complete the following statements by `int()` or `str()` so that all of them work.

```
In [1]: 9 + 9

In [2]: 9 + '9'

In [3]: '9' + '9'

In [4]: 9 * '9'
```

Exercise 5

Write a program that writes the following data into a two-column text file.

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

Processing multiple files

Exercise 1

The following program calculates the total number of births during the past 130 years.

The code contains a *subtle semantic bug*. Execute the program. Inspect the output. Find and fix the bug.

```
births = []
result = 0

print("\nyear    births per year")
print("-" * 25)
for year in range(1890, 2015, 1):
    filename = 'names/yob{}.txt'.format(year)
    for line in open(filename):
        spalten = line.strip().split(',')
        births.append(int(spalten[2]))
    print(year, sum(births))
    result += sum(births)
print("\nResult: {} births total".format(result))
```

Exercise 2

Write a program that finds lines containing your name in the years 1880 to 2014.

Exercise 3

Extend the program in such a way that the gender is being checked as well. Print only lines with matching `'M'` or `'F'`, respectively.

Exercise 4

Collect all matches in a list.

Exercise 5

If no matches were found in a given year, add a `0` to the result.

Screen output

In this chapter we will explor the `print()` function to write text to the screen.

Exercise 1

Which `print` statements are correct?

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

Exercise 2

Explain the following statement:

```
print("Emily\tSmith\n2000")
```

Exercise 3

Some babies have names similar to celebrities. Write a program producing the following output:

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

Exercise 4

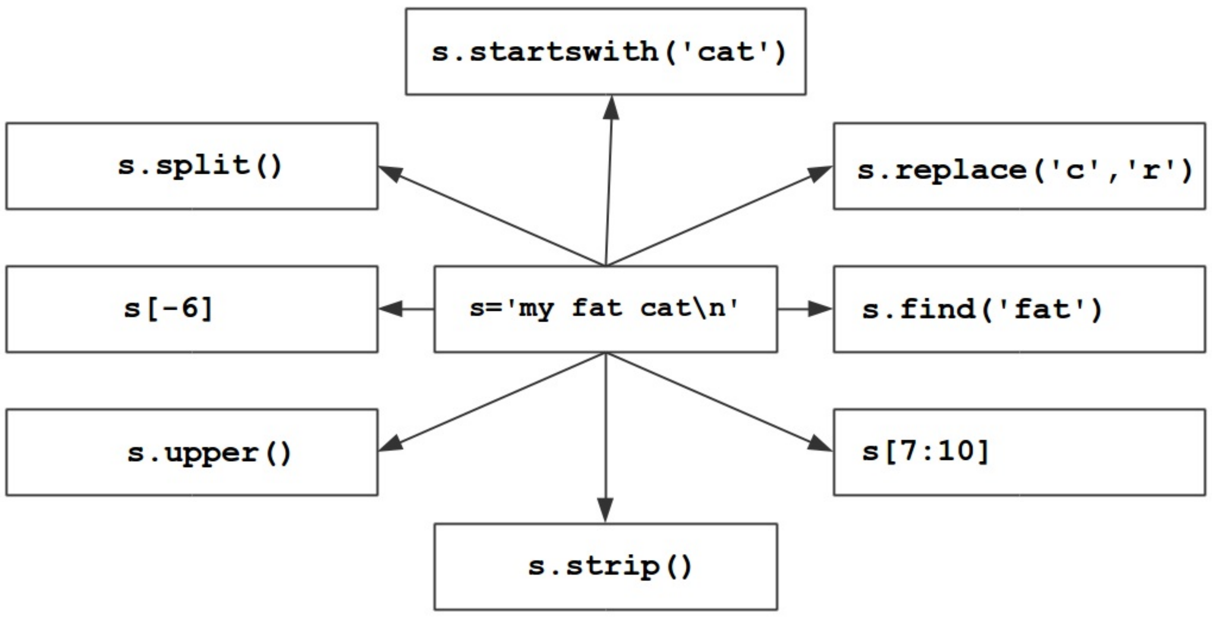
Expand the previous exercise so that:

- only a single `print` statement is used.
- the names are stored in a variable before building the string.

String methods

Exercise 1

Determine what the expressions do to the string in the center.



Exercise 2

The following program identifies names used for both girls and boys and writes them to a file.

Complete the code to dissect lines to columns, so that the variables `name` and `gender` are defined.

```
girls = []
duplicates = []

for line in open('names/yob2015.txt'):

    # insert your code here

    if gender == 'F':
       girls.append(name)
    elif gender == 'M':
       if name in girls:
           duplicates.append(name)

output = open('duplicate_names.txt', 'w')
for name in duplicates:
    text = "{:>15s}\n".format(name)
    output.write(text)
output.close()
```

Format Strings

Exercise 1

Try the following expressions in a Python shell:

```
"{}".format("Hello")
"{:10}".format("Hello")
"{:>10}".format("Hello")
"{1} {0}".format("first", "second")
"{:5d}".format(42)
"{:4.1f}".format(3.14159)
"{:6.3f}".format(3.14159)
```

Exercise 2

Write a `for` loop producing the following string:

```
000111222333444555666777888999
```

Exercise 3

You have the following two lists:

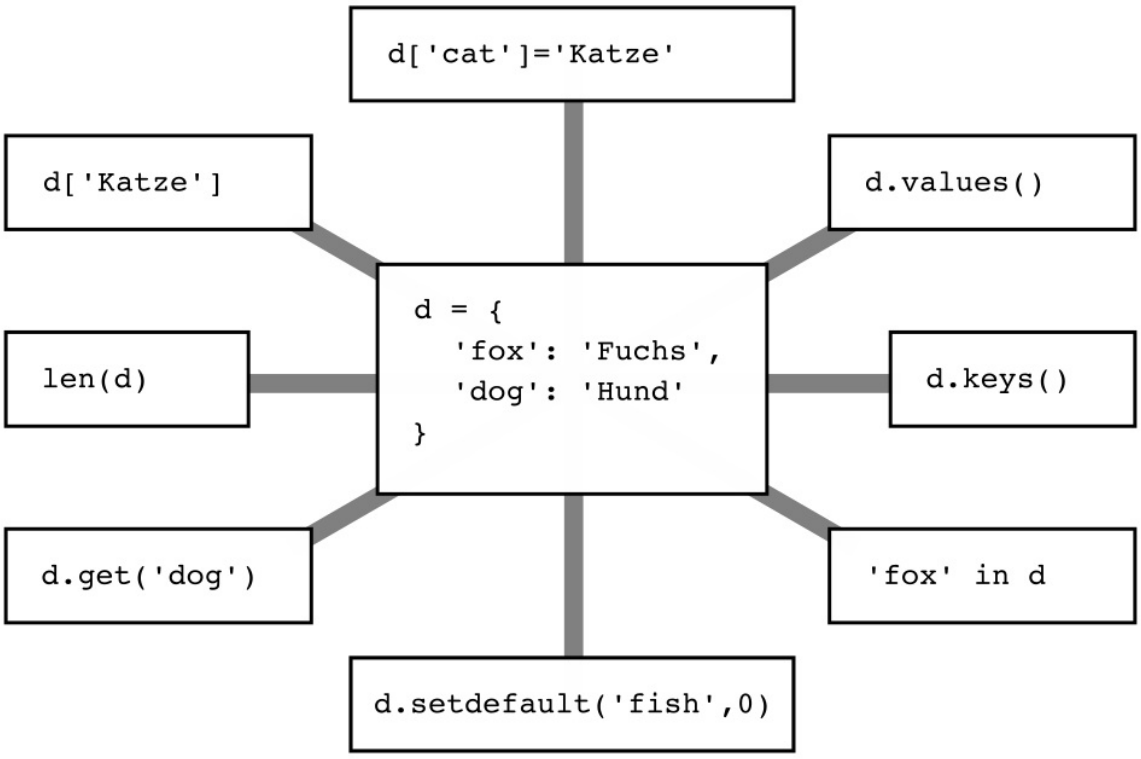
```
top_ten_names = ['Jacob', 'Michael', 'Matthew', 'Joshua', \
    'Christopher', 'Nicholas', 'Andrew', 'Joseph', \
    'Daniel', 'Tyler']

top_ten_numbers = [34465, 32025, 28569, 27531, \
    24928, 23632, 22818, 22307, 21500]
```

Write a program that creates a table with two vertically aligned columns.

Dictionaries

Exercise 1

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



Exercise 2

What do the following commands produce?

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

- `False`
- `"B"`
- `True`
- `1`

Exercise 3

What do these commands produce?

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

- `1`

- `True`
- `"B"`
- `False`

Exercise 4

What do these commands produce?

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

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

Exercise 5

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 6

What do these commands produce?

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

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

Exercise 7

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`

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.

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.

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

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.

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.

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:

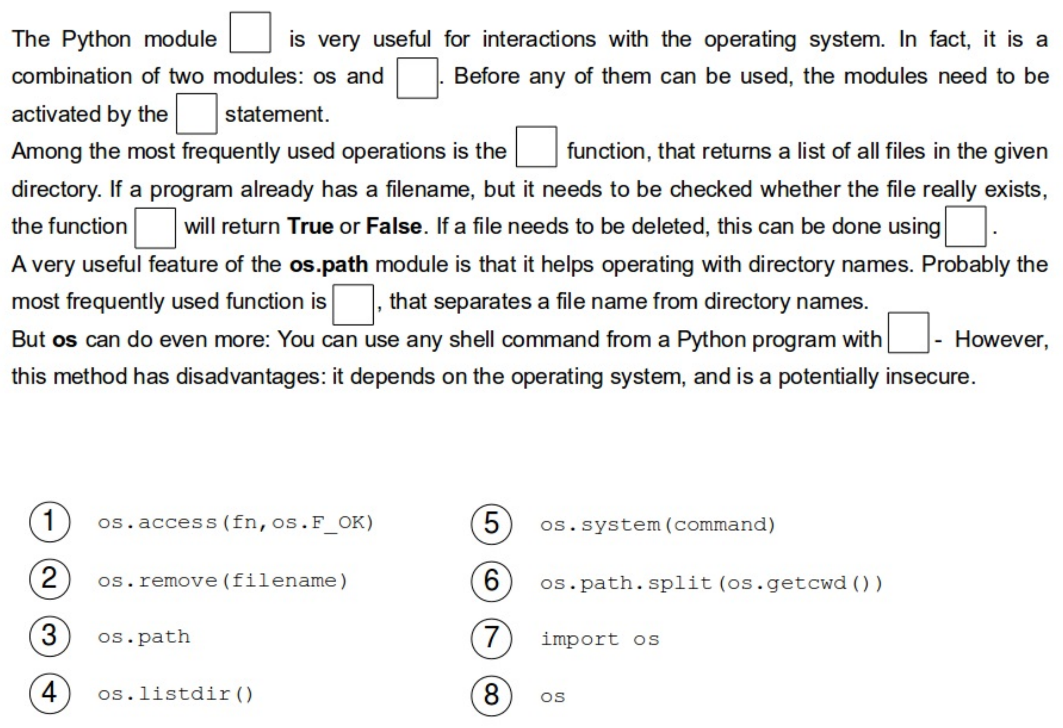
- let the user choose the gender of the babies.
- let the user enter how many babies they want to have.
- load baby names from a file.

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)`	5	`os.system(command)`
2	`os.remove(filename)`	6	`os.path.split(os.getcwd())`
3	`os.path`	7	`import os`
4	`os.listdir()`	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.

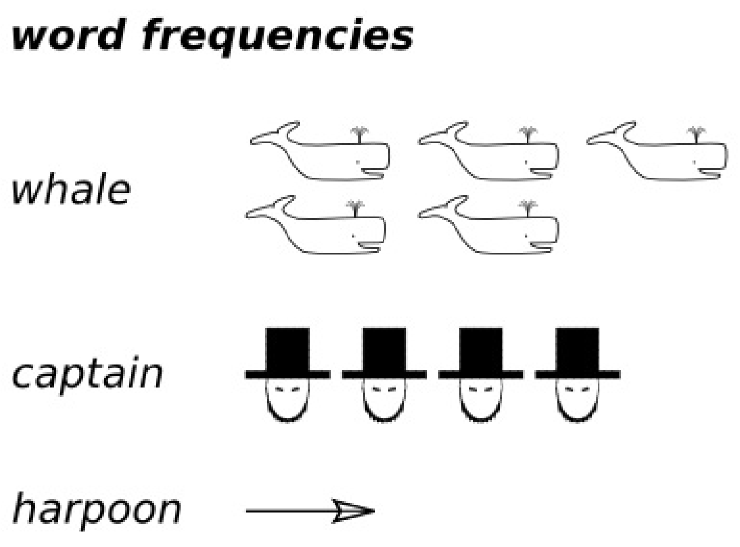
The Challenge

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.

Coding Challenge: Count Words in Moby Dick

The Challenge

The book *"Moby Dick"* by Herman Melville describes an epic battle of a gloomy captain against his personal nemesis, the white whale. Who of them is mentioned in the book more often?



Your Task

Write a program that counts how often each word occurs in the book. Determine how often the words `captain` and `whale` occur. You will need different data structures for counting words and for sorting them.

When you know whether `whale` or `captain` occurs more often, you have mastered this challenge.

Use the 8 hint cards as necessary.

Background

You can find the full text for Herman Melville's "Moby Dick" in the text file `mobydick.txt` and on www.gutenberg.org.

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