You are encouraged to start up a Python environment (like Spyder or Jupyter Notebook). If you do so, you can try out the examples while listening. If you prefer to listen only, that's fine as well.

Big Data and Automated Content Analysis

Week 2 – Monday

»Getting started with Python«

Anne Kroon

a.c.kroon@uva.nl
@annekroon

Afdeling Communicatiewetenschap
Universiteit van Amsterdam

April 8, 2019

Today

1. The very, very, basics of programming with Python
   - Datatypes
   - Functions and methods
   - Modifying lists and dictionaries
   - Indention: The Python way of structuring your program
2. Exercise
3. Next meetings

The very, very, basics of programming

See also Chapter 4.

Python lingo

Basic datatypes (variables)

int	`32`
float	`1.75`
bool	`True`, `False`
string	`"Jessica"`

Python lingo

Basic datatypes (variables)

int	`32`
float	`1.75`
bool	`True`, `False`
string	`"Jessica"`
(variable name	`firstname`)

"firstname" and firstname is not the same.

Python lingo

Basic datatypes (variables)

int	`32`
float	`1.75`
bool	`True`, `False`
string	`"Jessica"`
(variable name	`firstname`)

"firstname" and firstname is not the same.
"5" and 5 is not the same.
But you can transform it: `int("5")` will return 5.
You cannot calculate `3 * "5"` (In fact, you can. It's `"555"`).
But you can calculate `3 * int("5")`

Python lingo

More advanced datatypes

Python lingo

More advanced datatypes

```
list firstnames = ['Damian','Lori','Bjoern']
     lastnames =
     ['Trilling','Meester','Burscher']
```

Note that the elements of a list, the keys of a dict, and the values of a dict can have any datatype! (Better to be consistent, though!)

Python lingo

More advanced datatypes

```
list firstnames = ['Damian','Lori','Bjoern']
     lastnames =
     ['Trilling','Meester','Burscher']
list ages = [18,22,45,23]
```

Note that the elements of a list, the keys of a dict, and the values of a dict can have any datatype! (Better to be consistent, though!)

Python lingo

More advanced datatypes

list `firstnames = ['Damian','Lori','Bjoern']`
`lastnames = ['Trilling','Meester','Burscher']`

list `ages = [18,22,45,23]`

dict `familynames= {'Bjoern': 'Burscher', 'Damian': 'Trilling', 'Lori': 'Meester'}`

dict `{'Bjoern': 26, 'Damian': 31, 'Lori': 25}`

Note that the elements of a list, the keys of a dict, and the values of a dict can have any datatype! (Better to be consistent, though!)

Python lingo

Functions

Python lingo

Functions

functions Take an input and return something else `int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

Python lingo

Functions

functions Take an input and return something else `int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

methods are similar to functions, but directly associated with an object. `"SCREAM".lower()` returns the string "scream"

Python lingo

Functions

functions Take an input and return something else `int(32.43)` returns the integer 32. `len("Hello")` returns the integer 5.

methods are similar to functions, but directly associated with an object. `"SCREAM".lower()` returns the string "scream"

Both functions and methods end with `()`. Between the `()`, *arguments* can (sometimes have to) be supplied.

Writing own functions

You can write an own function:

```
1  def addone(x):
2      y = x + 1
3      return y
```

Functions take some input ("argument") (in this example, we called it `x`) and *return* some result.
Thus, running

```
1  addone(5)
```

returns `6`.

Modifying lists and dictionaries

Modifying lists

Appending to a list

```
1 mijnlijst = ["element 1", "element 2"]
2 anotherone = "element 3" # note that this is a string, not a list!
3 mijnlijst.append(anotherone)
4 print(mijnlijst)
```

gives you:

```
1 ["element 1", "element 2", "element 3"]
```

Modifying lists

Merging two lists (= extending)

```
1 mijnlijst = ["element 1", "element 2"]
2 anotherone = ["element 3", "element 4"]
3 mijnlist.extend(anotherone)
4 print(mijnlijst)
```

gives you:

```
1 ["element 1", "element 2", "element 3", "element 4]
```

Modifying dicts

Adding a key to a dict (or changing the value of an existing key)

```
1 mydict = {"whatever": 42, "something": 11}
2 mydict["somethingelse"] = 76
3 print(mydict)
```

gives you:

```
1 {'whatever': 42, 'somethingelse': 76, 'something': 11}
```

If a key already exists, its value is simply replaced.

Indention: The Python way of structuring your program

Indention

Structure

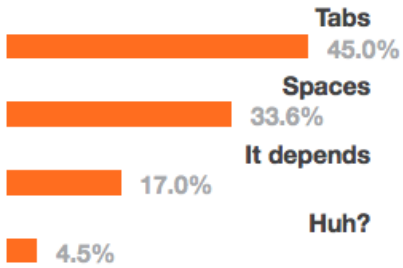
The program is structured by TABs or SPACEs

stackoverflow

Looking for a job?

about 10% of developers still don't use it.

VII. TABS VS. SPACES



25,807 responses

After millennia of heated debate, mercifully, at long last, we have an answer. **Most developers prefer tabs to spaces.**

Upon closer examination of the data, a trend emerges: Developers increasingly prefer spaces as they gain experience. Stack Overflow reputation correlates with a preference for spaces, too: users who have 10,000 rep or more prefer spaces to tabs at a ratio of 3 to 1.

Indention

Structure

The program is structured by TABs or SPACEs

```
1  firstnames=['Anne','Lori','Bjoern']
2  age={'Bjoern': 27, 'Anne': 33, 'Lori': 26}
3  print ("The names and ages of these people:")
4  for naam in firstnames:
5      print (naam,age[naam])
```

Indention

Structure

The program is structured by TABs or SPACEs

```
1 firstnames=['Anne','Lori','Bjoern']
2 age={'Bjoern': 27, 'Anne': 33, 'Lori': 26}
3 print ("The names and ages of these people:")
4 for naam in firstnames:
5     print (naam,age[naam])
```

Don't mix up TABs and spaces! Both are valid, but you have to be consequent!!! Best: always use 4 spaces!

Indention

Structure

The program is structured by TABs or SPACEs

```
1  print ("The names and ages of all these people:")
2  for naam in firstnames:
3      print (naam,age[naam])
4      if naam=="Anne":
5          print ("She teaches this course")
6      elif naam=="Lori":
7          print ("She is a former assistant")
8      elif naam=="Bjoern":
9          print ("He helped teaching this course in the past")
10     else:
11         print ("No idea who this is")
```

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a :

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a `:`

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (`if`, `elif`, and `else` statements)

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a `:`

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (`if`, `elif`, and `else` statements)
- an alternative block should be executed if an error occurs (`try` and `except` statements)

Indention

The line *before* an indented block starts with a *statement* indicating what should be done with the block and ends with a `:`

Indention of the block indicates that

- it is to be executed repeatedly (`for` statement) – e.g., for each element from a list
- it is only to be executed under specific conditions (`if`, `elif`, and `else` statements)
- an alternative block should be executed if an error occurs (`try` and `except` statements)
- a file is opened, but should be closed again after the block has been executed (`with` statement)

We'll now together do some simple exercises . . .

Exercises

1. Warming up

- Create a list, loop over the list, and do something with each value (you're free to choose).

2. Did you pass?

- Think of a way to determine for a list of grades whether they are a pass (>5.5) or fail.
- Can you make that program robust enough to handle invalid input (e.g., a grade as 'ewghjieh')?
- How does your program deal with impossible grades (e.g., 12 or -3)?
- ...

Next meetings

Thursday

We will work together on "Describing an existing structured dataset" (Appendix A).
Preparation: Make sure you understood all of today's concepts!