Python & pandas

A one day course

Uwe Ziegenhagen

github.com/UweZiegenhagen/OneDayPythonPandasCourse

Cologne, 8. Juli 2022

Introduction

Why Python/pandas?

- You *have* a CSV-file with semicolon as column separator and comma as decimal separator
- You *need* a CSV-file with comma as column separator and dot as decimal separator

```python
import pandas as pd

df = pd.read_csv('myfile.csv', sep=';', decimal = ',')
df = pd.to_csv('myfile.csv', sep=';', decimal = ',')
```

Limits of this Course

- Basis for this course is a semester-long course I held at the FOM („Fachhochschule für Ökonomie und Management“) in Cologne
- It is not a *full* course, we would need a whole week for this.
- We will skip many interesting things (that you do not necessarily need for your job)
- Goal: Teach you enough Python to a) read and b) understand Python-Code and c) write smaller programs relevant for your job

Introduction

Python

- Datatypes
- Functions
- Kontrollfluss

pandas

Links

Python

- Invented by Guido van Rossum at the „Centrum Wiskunde & Informatica“ in Amsterdam as successor for the teaching language ABC
- Current version is 3.11
- For a long time, Python 3.x and Python 2.7 existed together
- Python 2.7 support expired in 2019:
- How to spot 2.7 code: → `print 'hello'` instead of `print('hello')`

Python versus Java & C

Python code is often much slower than C or Java but. . .

- the implementation time for Python is way faster
- speeds only matters sometimes, not always
- many computing-intensive Python modules use C/C++ modules „under the hood“
- bad C-Code is slower than good Python-code

pandas

- A Python library for data wrangling and management
- Invented by Wes McKinney during his time at AQR Capital Management
- In his own words: „I tell them that it enables people to analyze and work with data who are not expert computer scientists," he says. „You still have to write code, but it's making the code intuitive and accessible. It helps people move beyond just using Excel for data analysis."[1]

[1] qz.com/1126615/the-story-of-the-most-important-tool-in-data-science/

Python

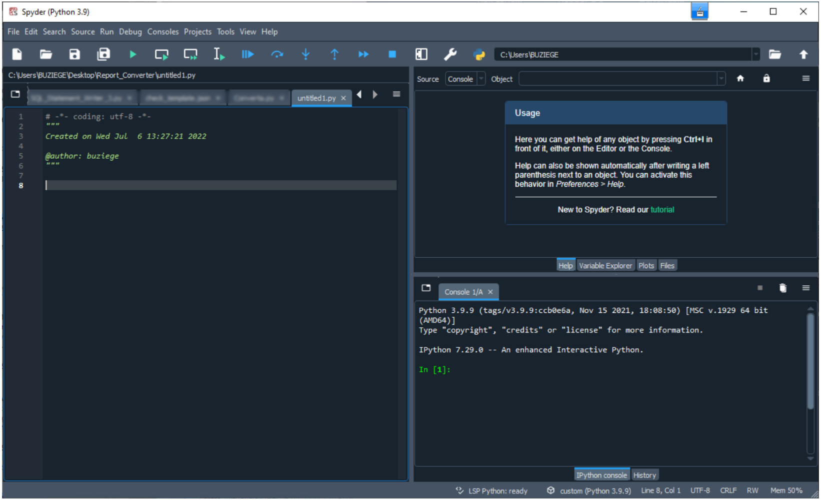
Datatypes

Functions

Kontrollfluss

Spyder

- We will use the Spyder5 IDE with Python 3.9.9, make sure it is installed



```
# -*- coding: utf-8 -*-
"""
Created on Wed Jul  6 13:27:21 2022

@author: buziege
"""

```

Usage

Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
Python 3.9.9 (tags/v3.9.9:ccb0e6a, Nov 15 2021, 18:08:50) [MSC v.1929 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.29.0 -- An enhanced Interactive Python.

In [1]:
```

Python as a Calculator

- Spyder5 runs an IPython kernel, this runs our programs
- We can also use it as a calculator

```
In [1]: 4*5.4
Out[1]: 21.6

In [2]: 4/12
Out[2]: 0.3333333333333333

In [3]: _*3
Out[3]: 1.0

'hello'
Out[4]: 'hello'
```

Python as a Calculator

```

In [1]: 4%2
Out[1]: 0

In [2]: 5%2 # Modulo
Out[2]: 1

In [3]: 3**3
Out[3]: 27

In [4]: 5//2
Out[4]: 2
```

Priority of Operators

- Round brackets have highest priority
- followed by Power
- followed by multiplication and division
- followed by addition and substraction

TODO4U:

$\Rightarrow$ Solve exercise sheet 1!

Basic Input & Output

```python
print('Hello World')

yourName = input('Tell me your name: ')

print('Hello ' + yourName + ', welcome to this class')

print('Hello ', yourName,', welcome to this class', sep='')

print(f'Hello {yourName}, welcome to this class')
```

- `input()` only reads strings
- If you need a number, you need to convert it
- there are better ways than `print()` for logging, but it works. . .
- f-Strings (last row) are recommended for mixed output!

Rules for Variables

- must start with a letter or _
- Case-sensitivity: 'A' is not 'a'
- Recommendation: small letters
- Let them speak for themselves: 'diameter' is good, 'd' is bad

Reserved Keywords

The following keywords are reserviert and must not be used for variables' names.

and	as	assert	break	class
continue	def	del	elif	else
except	False	finally	for	from
global	if	import	in	is
lambda	None	nonlocal	not	or
pass	raise	return	True	try
while	with	yield		

Datentypen

- Integer (integer numbers)
- Float (Floating point number)
- Strings
- Booleans
- Complex numbers

Integer

- unlimited length
- must not start with 0 if they shall represent decimal numbers
- Leading 0 by representation in hex-, binary- or Octal system:

`0b/0B`	binary
`0x/0X`	hex
`0o/0O`	octal

- Functions `hex()`, `bin()`, `oct()` for conversions into string, are internally represented as decimal numbers

Float

- floating point numbers
- `3.1415927`
- `3.1e8`
- Hint: Not every floating point number can be represented *exactly* („Floating-Point Arithmetic“)
- `docs.python.org/3/tutorial/floatingpoint.html`

Strings

- Single or double quotes
- For multiline strings:
  - Triple double or single quotes
  - Alternatively backslash at the end of the line
- Python has numerous functions for strings, more on this later
- Comments start with a hash #

```python
a = "I am a string"

b = 'me too'

c = """I am
as string as well """

# I am a comment
```

Booleans

- Named after George Boole
- 1854: „An investigation into the Laws of Thought“
- Essence of modern computing
- Boolean operators or (∪), and (∩), `not`

```python
a = True
b = False

a == b #False
a or b # True
a and b # False
a and not b # True
not a and b # False
```

Type Conversions

- Mixing strings and floats/integers requires explicit type conversion using the `str()` function
- Hint: pandas provides `.astype(<Datatype>)`

```
>>> a + str(b)
'abc123'
>>> a+str(c)
'abc3.141'
>>> a*str(b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'

>>> 'a' * 3
'aaa'

>>>3 * 'a'
'aaa'
```

Functions

- Functions: named sequence of commands
- Purpose: encapsule code for multiple calls
- Can take arguments as input, can have return values
- Define a new function as
    - Determine name
    - Determine arguments
    - Define function code

Built-in Functions

abs
all
any
ascii
bin
bool
breakpoint
bytearray
bytes
callable
chr
classmethod
compile
complex
delattr
dict
dir
divmod
enumerate
eval
exec
filter
float
format
frozenset
getattr
globals
hasattr
hash
help
hex
id
input
int
isinstance
issubclass
iter
len
list
locals
map
max
memoryview
min
next
object
oct
open
ord
pow
print
property
range
repr
reversed
round
set
setattr
slice
sorted
staticmethod
str
sum
super
tuple
type
vars
zip
__import__

Functions in C and Python

```c
    #include<stdio.h>
    long add(long a, long b){
       long result;
       result = a + b;
       return result;}

    int main(){
      int a, b, c;
      printf("Enter two numbers\n");
      scanf("%d%d", &a, &b);
      long myresult = add(a,b);
      printf("Sum of numbers = %d\n", myresult);
      return 0;}
```

Listing 1: addTwoNumbers.c Code

```python
def add(a, b):
    return int(a) + int(b)

a, b = input('Enter two numbers!').split()
print('The sum is: {}'.format(add(a,b)))
```

Listing 2: addTwoNumbers.py Code

Simple Functions

Empty Functions

- pass is often used in the development process
- Useful, when parts of the code are not ready, yet
- without pass one gets `IndentationError: expected an indented block` error

```python
def unfinished_function():
    pass


unfinished_function()
```

Listing 3: funktion-12.py Code

Simple Functions

- No parameter
- No return value (`void`)

```python
def print_hello():
    print('Hello')


print_hello()
```

Listing 4: funktion-01.py Code

```
Hello
```

Functions with Arguments

- Function with one argument `text`
- Error message, when argument is missing

```python
def print_text(text):
    print(text)


print_text('Hello World!')
```

Listing 5: funktion-02.py Code

```
Hello World!
```

Functions with Arguments

- Two arguments, `text` and `count`

```python
def print_text_multiple(text, anzahl):
    print(text*count)


print_text_multiple('Hello!', 0)

print_text_multiple('Hello!', 1)

print_text_multiple('Hello!', 3)

print_text_multiple('Hello!', -1)
```

Listing 6: funktion-03.py Code

```

Hello!
Hello!Hello!Hello!
```

Functions with Arguments

- Setting standard values for parameters
- Allows calling the function without parameters

```python
def print_text_multiple(text='Hello CGN', count=2):
    print(text*count)


print_text_multiple()

print_text_multiple('Hello!')

print_text_multiple('Hello!',3)
```

Listing 7: funktion-04.py Code

```
Hello CGNHello CGN
Hello!Hello!
Hello!Hello!Hello!
```

Functiona with return values

- Functions can return values for further processing

```python
def clone_text(text, count=2):
    return text*count


a = clone_text('Huhu')

print(a)
```

Listing 8: funktion-05.py Code

```
HuhuHuhu
```

Functions with multiple return values

- Functions can have more than one return value
- Function then return tuples, an unmutuable list of values[2]
- Dealing with the tuple is called „Unpacking“
- Remark: Parameter `sep` in the example is a parameter of the `print()` function

```python
def clone_text(text, count=2):
    return count, text*count


i, a = clone_text('Huhu')

print(i, a, sep='>')
```

Listing 9: funktion-06.py Code

```
2>HuhuHuhu
```

[2] more on this later

Functions with variable count of arguments

- Parameter with *: variable count of arguments
- Parameter with **: variable count of key-value arguments

```python
def addthem(*args):
    result = 0
    for number in args:
        result += number
    return result


print(addthem(1, 2, 3, 4, 5))
```

Listing 10: funktion-08.py Code

```python
def give(**args):
    for key, value in args.items():
        print(key, value, sep=': ')


print(give(Vorname='Uwe', Nachname='Ziegenhagen'))
```

Listing 11: funktion-09.py Code

Kontrollfluss

englisch: „flow control“

- Eingabe & Ausgabe
  - input
  - print
- Verzweigungen
  - if else elif
- Schleifen
  - for
  - while

Verzweigungen

- `if <condition>:`

```python
def check_laenge(text):
    if len(text)>8:
        print('Länger als 8 Zeichen!')

check_laenge('Köln')
check_laenge('Düsseldorf')
```

Listing 12: if-01.py Code

```
Länger als 8 Zeichen!
```

Verzweigungen

- Es gibt kein `switch()` Konstrukt in Python
- `if` Statements können mehrfach verwandt werden
- ist aber nicht „pythonic“

```python
def check_laenge(text):
    if len(text)>8:
        print(text, 'Länger als 8 Zeichen!', sep=': ')
    if len(text)<=8:
        print(text, 'Kürzer oder gleich als 8 Zeichen!', sep=': ')

check_laenge('Köln')
check_laenge('Düsseldorf')
```

Listing 13: if-02.py Code

```
Köln: Kürzer oder gleich als 8 Zeichen!
Düsseldorf: Länger als 8 Zeichen!
```

Verzweigungen

- more „pythonic“: `else`:

```python
def check_laenge(text):
    if len(text)>8:
        print(text, 'Länger als 8 Zeichen!', sep=': ')
    else:
        print(text, 'Kürzer oder gleich als 8 Zeichen!', sep=': ')

check_laenge('Köln')
check_laenge('Düsseldorf')
```

Listing 14: if-03.py Code

```
Köln: Kürzer oder gleich als 8 Zeichen!
Düsseldorf: Länger als 8 Zeichen!
```

Verzweigungen

- Verschachteln von `if <condition>: else:` führt zu unübersichtlichem Code

```python
def check_laenge(text):
    if len(text)>8:
        print(text, 'Länger als 8 Zeichen!', sep=': ')
    else:
        if len(text)<=5:
            print(text, 'Kürzer oder gleich als 5 Zeichen!', sep=': ')
        else:
            print(text, 'Länger als 5, kürzer als 8', sep=': ')

check_laenge('Köln')
check_laenge('Berlin')
check_laenge('Düsseldorf')
```

Listing 15: if-04.py Code

```
Köln: Kürzer oder gleich als 5 Zeichen!
Berlin: Länger als 5, kürzer als 8
Düsseldorf: Länger als 8 Zeichen!
```

Verzweigungen

- `if <condition>: else:` kann zu `elif:` abgekürzt werden

```python
def check_laenge(text):
    if len(text)>8:
        print(text, 'Länger als 8 Zeichen!', sep=': ')
    elif len(text)<=5:
        print(text, 'Kürzer oder gleich als 5 Zeichen!', sep=': ')
    else:
        print(text, 'Länger als 5, kürzer als 8', sep=': ')

check_laenge('Köln')
check_laenge('Berlin')
check_laenge('Düsseldorf')
```

Listing 16: if-05.py Code

```
Köln: Kürzer oder gleich als 5 Zeichen!
Berlin: Länger als 5, kürzer als 8
Düsseldorf: Länger als 8 Zeichen!
```

Schleifen

for

- for Schleifen iterieren über eine Sequenz
- Sequenz kann String, Liste, etc. sein
- range(start, ende, stepsize=1) erzeugt numerische Sequenz von start bis unter (!) ende mit Schrittweite stepsize

```
for char in 'Hallo Welt':
    print(char)

for j in range(1, 10):
    print(j) # 1 bis 9

for j in range(1, 10, 3):
    print(j) # 1, 4 und 7
```

Listing 17: for-01.py Code

Schleifen

while

- while Loop läuft, bis Bedingung nicht mehr erfüllt ist

```python
s = 'Hallo Welt'
l = len(s)

while (l>0):
    print(s[l-1])
    l-=1

i = 1
while (i<100):
    i += 1

print(i)
```

Listing 18: while-01.py Code

Schleifen

break und continue

- break und continue beeinflussen Schleifen
- break kann z. B. dazu genutzt werden, eine Schleife vorzeitig zu verlassen

```python
s = 'Hallo Welt'
l = len(s)

while (l>0):
    temp = s[l-1]
    if temp == 'W':
        break
    print(temp)
    l-=1
```

Listing 19: while-02.py Code

Schleifen

`break` und `continue`

- `continue` überspringt restliche Sequenz, setzt Schleife fort

`for` Schleifen mit `else` Zweig

- Python unterstützt `else` Konstrukte in `for` Schleifen
- Werden dann aufgerufen, wenn kein Abbruch durch `break` erfolgt ist
- oftmals zusammen mit `if` benutzt
- Grundlegende Syntax:

```python
for item in container:
    if search_something(item):
        # Found it!
        process(item)
        break
else:
        # Didn't find anything..
        not_found_in_container()
```

`while` Schleifen mit `else` Zweig

- Python unterstützt auch `else` Konstrukte in `while` Schleifen
- der `else` Zweig wird aufgerufen, wenn die `while` Bedingung auf `False` geht
- wird nicht aufgerufen bei `break`, `return` oder Exception
- Grundlegende Syntax:

```python
while condition:
    handle_true()
else:
    # condition is false now, handle and go on with the rest of the program
    handle_false()
```

pandas

Links

Links

- https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html
- https://pandas.pydata.org
-
-
-
-