

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

```
1 import pandas as pd
2
3 df = pd.read_csv('myfile.csv', sep=';', decimal = ',')
4 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

- Flow control

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

¹qz.com/1126615/

Python

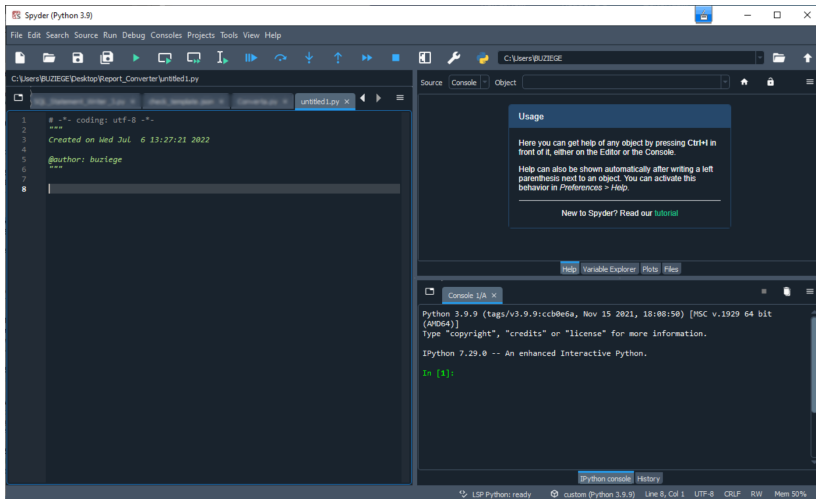
Datatypes

Functions

Flow control

Spyder

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



Python as a Calculator

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

```
1 In [1]: 4*5.4
2 Out[1]: 21.6
3
4 In [2]: 4/12
5 Out[2]: 0.3333333333333333
6
7 In [3]: _*3
8 Out[3]: 1.0
9
10 'hello'
11 Out[4]: 'hello'
```

Python as a Calculator

```
1
2 In [1]: 4%2
3 Out[1]: 0
4
5 In [2]: 5%2 # Modulo
6 Out[2]: 1
7
8 In [3]: 3**3
9 Out[3]: 27
10
11 In [4]: 5//2
12 Out[4]: 2
```

Priority of Operators

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

TODO4U:

⇒ Solve exercise sheet 1!

Basic Input & Output

```
1 print('Hello World')
2
3 yourName = input('Tell me your name: ')
4
5 print('Hello ' + yourName + ', welcome to this class')
6
7 print('Hello ', yourName, ', welcome to this class', sep='')
8
9 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 reserved 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 #

```
1 a = "I am a string"
2
3 b = 'me too'
4
5 c = """I am
6 as string as well """
7
8 # I am a comment
```

Booleans

- ▶ Named after George Boole
- ▶ 1854: „An investigation into the Laws of Thought“
- ▶ Essence of modern computing
- ▶ Boolean operators or (\cup) , and (\cap) , not

```
1 a = True
2 b = False
3
4 a == b #False
5 a or b # True
6 a and b # False
7 a and not b # True
8 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>)`

```
1 >>> a + str(b)
2 'abc123'
3 >>> a+str(c)
4 'abc3.141'
5 >>> a*str(b)
6 Traceback (most recent call last):
7   File "<stdin>", line 1, in <module>
8   TypeError: can't multiply sequence by non-int of type 'str'
9
10 >>> 'a' * 3
11 'aaa'
12
13 >>> 3 * 'a'
14 '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	delattr	hash	memoryview	set
all	dict	help	min	setattr
any	dir	hex	next	slice
ascii	divmod	id	object	sorted
bin	enumerate	input	oct	staticmethod
bool	eval	int	open	str
breakpoint	exec	isinstance	ord	sum
bytearray	filter	issubclass	pow	super
bytes	float	iter	print	tuple
callable	format	len	property	type
chr	frozenset	list	range	vars
classmethod	getattr	locals	repr	zip
compile	globals	map	reversed	__import__
complex	hasattr	max	round	

Functions in C and Python

```
1  #include<stdio.h>
2  long add(long a, long b){
3      long result;
4      result = a + b;
5      return result;}
6
7  int main(){
8      int a, b, c;
9      printf("Enter two numbers\n");
10     scanf("%d%d", &a, &b);
11     long myresult = add(a,b);
12     printf("Sum of numbers = %d\n", myresult);
13     return 0;}
```

Listing 1: addTwoNumbers.c Code

```
1  def add(a, b):
2      return int(a) + int(b)
3
4  a, b = input('Enter two numbers!').split()
5  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

```
1 def unfinished_function():  
2     pass  
3  
4  
5 unfinished_function()
```

Listing 3: funktion-12.py **Code**

Simple Functions

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

```
1 def print_hello():  
2     print('Hello')  
3  
4  
5 print_hello()
```

Listing 4: funktion-01.py **Code**

```
1 Hello
```

Functions with Arguments

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

```
1 def print_text(text):  
2     print(text)  
3  
4  
5 print_text('Hello World!')
```

Listing 5: funktion-02.py **Code**

```
1 Hello World!
```

Functions with Arguments

- ▶ Two arguments, text and count

```
1 def print_text_multiple(text, anzahl):  
2     print(text*count)  
3  
4  
5 print_text_multiple('Hello!', 0)  
6  
7 print_text_multiple('Hello!', 1)  
8  
9 print_text_multiple('Hello!', 3)  
10  
11 print_text_multiple('Hello!', -1)
```

Listing 6: funktion-03.py Code

```
1  
2 Hello!  
3 Hello!Hello!Hello!
```

Functions with Arguments

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

```
1 def print_text_multiple(text='Hello CGN', count=2):  
2     print(text*count)  
3  
4  
5 print_text_multiple()  
6  
7 print_text_multiple('Hello!')  
8  
9 print_text_multiple('Hello!',3)
```

Listing 7: funktion-04.py Code

```
1 Hello CGNHello CGN  
2 Hello!Hello!  
3 Hello!Hello!Hello!
```

Functiona with return values

- Functions can return values for further processing

```
1 def clone_text(text, count=2):  
2     return text*count  
3  
4  
5 a = clone_text('Huhu')  
6  
7 print(a)
```

Listing 8: funktion-05.py Code

```
1 HuhuHuhu
```

Functions with multiple return values

- ▶ Functions can have more than one return value
- ▶ Function then return tuples, an unmutable list of values²
- ▶ Dealing with the tuple is called „Unpacking“
- ▶ Remark: Parameter sep in the example is a parameter of the `print()` function

```
1 def clone_text(text, count=2):  
2     return count, text*count  
3  
4  
5 i, a = clone_text('Huhu')  
6  
7 print(i, a, sep='>')
```

Listing 9: funktion-06.py **Code**

```
1 2>HuhuHuhu
```

²more on this later

Functions with variable count of arguments

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

```
1 def addthem(*args):  
2     result = 0  
3     for number in args:  
4         result += number  
5     return result  
6  
7  
8 print(addthem(1, 2, 3, 4, 5))
```

Listing 10: funktion-08.py Code

```
1 def give(**args):  
2     for key, value in args.items():  
3         print(key, value, sep=': ')  
4  
5  
6 print(give(Vorname='Uwe', Nachname='Ziegenhagen'))
```

Listing 11: funktion-09.py Code

Flow control

- ▶ Input & Output
 - ▶ input
 - ▶ print
- ▶ Branching
 - ▶ if else elif
- ▶ Loops
 - ▶ for
 - ▶ while

Branching

► `if <condition>:`

```
1 def check_length(text):  
2     if len(text)>8:  
3         print('Longer than 8 characters!')  
4  
5 check_length('Köln')  
6 check_length('Düsseldorf')
```

Listing 12: if-01.py **Code**

```
1 Longer than 8 characters!
```

Branching

- ▶ There is no `switch()` in Python
- ▶ `if` statements can be used multiple times
- ▶ maybe not the most pythonic approach

```
1 def check_length(text):
2     if len(text)>8:
3         print(text, 'Longer than 8 characters!', sep=': ')
4     if len(text)<=8:
5         print(text, 'Shorter or equal to 8 characters!', sep=': '
6             )
7 check_length('Köln')
8 check_length('Düsseldorf')
```

Listing 13: if-02.py **Code**

```
1 Köln: Shorter or equal to 8 characters!
2 Düsseldorf: Longer than 8 characters!
```

Branching

► more „pythonic“: `else:`

```
1 def check_length(text):
2     if len(text)>8:
3         print(text, 'Longer than 8 characters!', sep=': ')
4     else:
5         print(text, 'Shorter or equal to 8 characters!', sep=': ')
6
7 check_length('Köln')
8 check_length('Düsseldorf')
```

Listing 14: if-03.py Code

```
1 Köln: Shorter or equal to 8 characters!
2 Düsseldorf: Longer than 8 characters!
```

Branching

- Nesting von `if <condition>`: `else`: leads to confusing code

```
1 def check_length(text):
2     if len(text)>8:
3         print(text, 'Longer than 8 chars!', sep=': ')
4     else:
5         if len(text)<=5:
6             print(text, 'Shorter or equal 5 chars!', sep=': ')
7         else:
8             print(text, 'Longer than 5, shorter than 8', sep=': ')
9
10 check_length('Köln')
11 check_length('Berlin')
12 check_length('Düsseldorf')
```

Listing 15: if-04.py Code

```
1 Köln: Shorter or equal 5 chars!
2 Berlin: Longer than 5, shorter than 8
3 Düsseldorf: Longer than 8 chars!
```

Branching

► `if <condition>: else:` can be shortened to `elif`:

```
1 def check_length(text):
2     if len(text)>8:
3         print(text, 'Longer than 8 chars!', sep=': ')
4     elif len(text)<=5:
5         print(text, 'Shorter or equal 5 chars!', sep=': ')
6     else:
7         print(text, 'Longer than 5, shorter than 8', sep=': ')
8
9 check_length('Köln')
10 check_length('Berlin')
11 check_length('Düsseldorf')
```

Listing 16: if-05.py Code

```
1 Köln: Shorter or equal 5 chars!
2 Berlin: Longer than 5, shorter than 8
3 Düsseldorf: Longer than 8 chars!
```

Loops

for

- ▶ `for` loops iterate over a sequence
- ▶ sequence can be a string, a list, etc.
- ▶ `range(start, end, stepsize=1)` creates numerical sequence from start until below (!) end with step size stepsize

```
1 for char in 'Hallo Welt':  
2     print(char)  
3  
4 for j in range(1, 10):  
5     print(j) # 1 bis 9  
6  
7 for j in range(1, 10, 3):  
8     print(j) # 1, 4 und 7
```

Listing 17: for-01.py Code

Loops

while

- ▶ `while` loop runs, until condition is not fulfilled anymore

```
1 s = 'Hallo Welt'
2 l = len(s)
3
4 while (l>0):
5     print(s[l-1])
6     l-=1
7
8 i = 1
9 while (i<100):
10     i += 1
11
12 print(i)
```

Listing 18: while-01.py **Code**

Loops

break and continue

- ▶ `break` and `continue` influence loops
- ▶ `break` can e. g. be used to exit a loop

```
1 s = 'Hallo Welt'
2 l = len(s)
3
4 while (l>0):
5     temp = s[l-1]
6     if temp == 'W':
7         break
8     print(temp)
9     l-=1
```

Listing 19: while-02.py **Code**

pandas

Links

Links

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