Python for Researchers

Based on the course 27. Feb to 1. Mar 2018 at ZEW by Peter Buchmann.

Change history

30.5.2019 – added tuples
5.6.2019 – setup instructions for Python 3.7.3

Table of contents

Basics

Variables and Types

- Variables, values and assignments
- Print function
- Types
- Conversions
- `if` Clause
- Formatted output

Variables, values and assignments

```
someVariable =  1.5
```

A `variable` can be imagined as a shoe box containing a `value`.
Value assignments happen by the equal sign =.
The `type` of the value is "number".

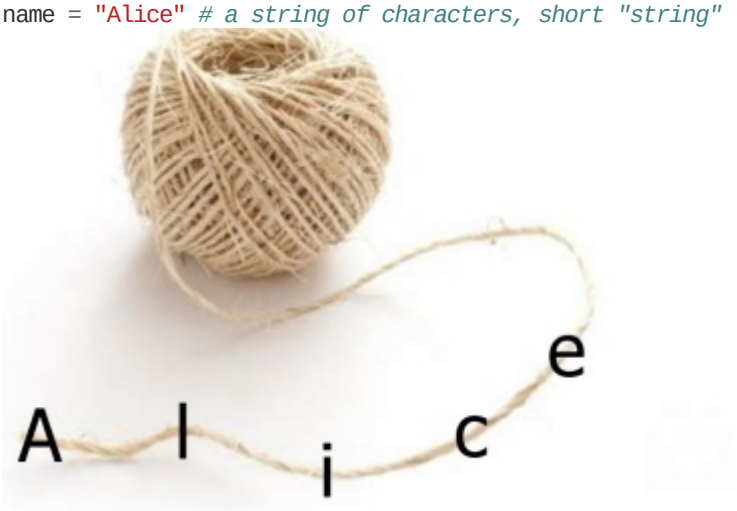
```
anotherVariable = "string-of-characters"
```

Another value assignment.
Quotation marks indicate a string value.
Without the quotation marks, `string-of-characters` would be considered a variable name.

Variables continued

In [10]:

```
age = 34       # integer
height = 1.7   # float
status = True  # bool
name = "Alice" # a string of characters, short "string"
```



Printing variables

```
# The print function outputs the content of a variable.
# Similar to display in stata
```

```
print(name)

print("values are", age, height, "!") # multiple variables, separated by comma
```

Types

Recognizing the aspect of types in programming languages is essential.

- Each variable has at least one `type`.
- The `type` governs what I can do with the variable.
  - floats have `division`; strings have not.
  - strings have `upper case` ('Alice' => 'ALICE') , floats have not.
- Above, we have seen only `primitive` `scalar` types.
- We will encounter all sorts of more complex `composite` types later on.

Print the type

```
# We can check the type of each variable using the type function:

# type(variable)

typeOfName = type(name)

print(typeOfName)


print( type(0) )

print( type(0.0) )
```

Conversion of types

```
# We can convert most types to string

print(str(1.0))

print(str(True))

# The other way around is not always easy:

print(int("0010")) # fine

print(int(height)) # fine, but truncated to 1

print(int("0.9") ) # blows up

converted = float("Alice") # also blows up
```

We will later write see how to solve this.

`if` clauses

`if` clauses enable us to have a *conditional* program execution.

- `if` clause starts with an `if`
- followed by an `expression` yielding `True` or `False`
- `if` clause ends with a **colon**
- `if` body must be indented
  - First line indent governs all following indents
  - Empty line or outdent ends if body

```
if 1 > 0:
    print("always greater")



# The simplest expression, that is always true
if True:
    print("always be true!")



# What is wrong with this?
# Remember assignments?
if var = 1:
    pass



# Testing for equality
if 2 == 1+1:
    print("notice the == as oposed to assignment '='")



# Unequal operator - body with multiple lines
if name != 'Bob':
    print("name is")
    print("not Bob")
```

```
print("unconditional code")
# else
# pass does nothing - but can be used to fullfill the requirement of at least one line in the if-body
if name == 'Bob':
    pass
else:
    print("not Bob")
```

`if` conditions for type checking

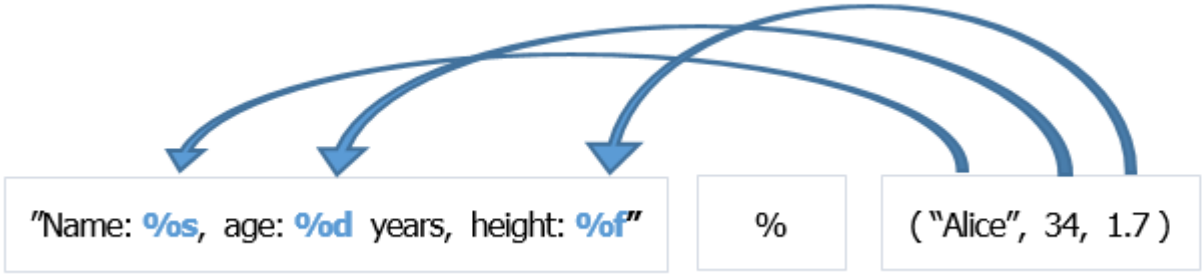
- We can compare types using `is` and branch our program accordingly.
- Notice the special operator `is`, instead of '=='.
- Notice the builtin 'type variables' str and int.
- Notice their green color indicating 'builtin' variables

```
if type("Alice") is str:
    print(
        "'Alice' is a string")


if type(0) is int:
    print("0 is an integer")
```

Formatted printing

- Format strings are combinations of decoration and placeholders for variable values.
- They are useful to create precise structured output.
- %s, %d, %f and %r are placeholders for values.
- %s demands a string, %d - digit, %f - float, %r - bool.
- We combine the formatting string with values using '%' (var1, var2, ...).




```
formatString = "Name: %s %d years old. %f metres tall. Married: %r"

# Note the '%' operator to combine format string and list of variables in brackets
out = formatString % (name, age, height, status)
print(out)
# Numbers between % and [s,d,f,r] make a padding - a filling of the value with space characters.
# Float variables are padded with trailing zeroes.
formatString = "Name: %12s %4d years old. %8f metres tall. Married: %r"
out = formatString % (name, age, height, status)
print(out)
# A leading 0 leads to padding with zeroes instead of spaces
print("%020d" % 111)

# Notice that the distinct types have slightly distinct formatting options.
# For floats, a dot and a second number give the number of digits after decimal separator
print("%8.2f" % 1.7)
# A minus sign makes left alignment; default is right
print("%-8.2f" % 1.7)
```

Exercise: Make a format string for vertical alignment

```
# Now combine these to make a good format string
formatString = "Name %s, age %d, height %f, married %r"


# We print two records - and hope they are vertically aligned:
```

```
name = "Alice"
age = 34
height = 1.7
status = True
out = formatString % (name, age, height, status)
print(out)



name = "Bob"
age = 7
height = 0.8
status = False
out = formatString % (name, age, height, status)
print(out)
```

Solution

```
formatString = "Name: %-12s %4d years old. %8.2f metres tall. Married: %r"
```

Special values and the `is` operator

During your work with data for stata,
or with data from excel or the internet, you will encounter a few "special values"
that you need handle by conditions.

```
# === None ===
# If variables of a composite type, such as employee,
# contain no data, then there is a special value for this condition.
newEmployee = None


# We test for None with special comparison operator "is":
if newEmployee is None:
    print("%s is     None" % newEmployee)



# === NaN ===
# When data files contain invalid numbers, i.e. 4z1,
```

```
# then Python often expresses this as "Not a Number" - NaN.
val = float('NaN')


# How do we test, whether a variable contains NaN?
#
# docs.python.org/3/library/math.html
# NaN is not considered close to any other value, including NaN.
if val == float('NaN'):
    print("val is not a number - 1") # does not work


# We actually have to use a special function, to test for NaN:
import math
if math.isnan(val):
    print("val is not a number - 2")

if math.isnan( float('nan') ):
    print("val is not a number - 3") # lower case does not matter


# === inf ===
# Finally, there is a special value for cases, when the
# variable contains numbers beyond the maximum scope of the variable.
#
if float('-inf') < 1.0 < float('inf'):
    print("float('-inf') < float(1) < float('inf')")


# inf and -inf are only considered close to themselves.
# Thus we need another specific test, to compare inf's
someDirtyVar = float('inf')
if math.isinf(someDirtyVar):
    print("someDirtyVar is infinite")
```

Comparison: multiple conditions, strings versus numbers, edge cases, conversions

```
name = "Alice"
age = 34
height = 1.7
status = True



# How to check, if Alice can vote (18), but cannot drink alcohol in Texas (21)
if age >= 18 and age < 21:
    print("Alice can vote, but she cannot drink alcohol in Texas")



# What happens, if we compare strings to each other?
if name > "A":
    print("'Alice' comes after 'A' in dictionary order")

if name < "Am":
    print("'Alice' comes before 'Am' in dictionary order")



# What happens, if we compare a float to an int?
if age > height:
    print("Most people are older than their height in metres")
    print("We also have an implicit conversion of int to float here.")

# What happens, if we compare an int to a string?
if age != "34":
    print("An integer is never equal to a string.")
    print("And there is no implicit conversion.")
```

Incompatible types

Some comparison between distinct types would only be wrong or misleading.

Therefore Python blows up in the next line.

```
# print(name > 10)
# if name > 10:  # blows up
#    pass
```

Conversions

We need conversions,

- for cleansing dirty data from the internet or Excel
- for cleansing ugly data in Stata

```
# Suppose we get this:
dirtyAge = " 34 "

# We have to convert this into an integer or into a float
#dirtyAge = " 34.0 "
#dirtyAge = " 34 a "


print(int(dirtyAge))
print(float(dirtyAge))

# How would we test, whether a conversion is necessary?
if type(dirtyAge) is not int:
    print("type of dirtyAge is not int but %s" % str(type(dirtyAge)))
    # quit()


# Try to find an erroneous conversion to string
print(str(True))
print(str(1/3))

# => We can always convert to string.
# Converting from string to numerical type or date or object is conditional.
```

```
# We will learn later, how to prevent blowups
```

Loops and date functions

- We only use simple loop constructs
- Loops start with `for`
- One or two 'loop variables' follow
- The keyword `in`
- A function returning something 'loopable'/'iterable' or a list or a dictionary (more later)
- A **colon**
- Loop body must be indented
  - First line indent governs all following indents
  - Outdented line ends loop

Later, we want to construct a *combined* example of loops and conditions,
where we use the years and weekdays as base for our loops and conditions.

```
for i in range(2):

    print("i is %d." % (i))
```

- Loop variable is `i`
- `i` can be used in loop body
- `range(2)` is a function that yields a stream of elements, that can be looped
- Notice range starts with 0 - implicitly, without us saying so.
- Notice that 2 is not reached.

Combine loop with format string

```
# Lets say the year is 2017 and Alice is 34 years old.

age = 34

curYear = 2017


# We can use the `range` function again.

# This time with an explicit start value of -2

for i in range(-2,4):

    print("In %d, Alice has her %dth birthday." % (curYear+i,34+i))


# Why is 2021 not printed?
```

Dates - making the current year 'dynamic'

- So far, our current year never changes. It is *static*.
  but our program should adapt to the passing of time.

- We now learn how to fetch the current year into a variable.
  It will change in time; it will be *dynamic*.

- We need a `module`

- Modules contain additional functions, that are not built into the core of Python.

- The module 'datetime' contains functions for dates and time.
  It is always installed with basic Python.
  Other modules like 'numpy' would have to be installed with pip before we could use them.

```
# instead of just assigning a constant year

curYear = 2017


import datetime as dt

# Notice the alias dt for the datetime module.

# It means, all functions from datetime must be prefixed with dt.

# Confusingly module datetime has a submodule which is also called datetime.

# In our case dt.datetime.now()

currentDateTime = dt.datetime.now()


curYear = currentDateTime.year # extract the year part

print("current year is %d" % curYear)



# Remember the type aspect of our variables:

print("type of currentDateTime is %s type of curYear is %s" % (type(currentDateTime), type(curYear)) )
```

Introducing modules

We just used a Python `module`.

Python has thousands of extensions; called `modules`.
Modules extend the functionality of Python.
Modules contain sets of additional functions.

We give a few examples, to demonstrate the scope of modules.

- Internet data gathering => requests, selenium

- Arrays and matrices => numpy
- Stata functions => statsmodels, scikit, matplotlib
- Website creation => django
- Online experiments => otree
- Machine learning => scikit, tensorflow
- White board software => jupyter-notebook (used to display this page)
- Thousands of modules on Github.com:
  Most active projects (http://github.com/trending/python)

Some modules are already installed with Python.
Others can be installed using `pip install [module-name]`.
The appendix (http://semalloc.zew.de:8000/notebooks/00/090-python-setup-command-line-access.ipynb) contains details on how to install Python and modules on your own.

Conda

Keeps modules updated for you.

Conda also offers older python version. https://conda.io/docs/index.html (https://conda.io/docs/index.html)

```
yearOfBirth = 1983


for i in range(-2,4):
    print("In %d, Alice has her %dth birthday." % (curYear+i,curYear+i-yearOfBirth))
```

Introducing the modulo operator

The output above is too 'verbose'.
Lets control the frequency of our output.

The % yields the remainder or modulus of an integer division.

For example 8%3 = 2

We can now restrain our printing to even years.

```
for i in range(-2,4):
    if (curYear+i)%2 == 0:
        print("In %d, Alice has her %dth birthday." % (curYear+i,curYear+i-yearOfBirth))
```

Excursion: When do I need brackets? - operator precedence

Notice the bracket around `curYear+i`

Python has the following *operator precedence* rules:

```
lowest                          ...                                highest

and, or, in, not in, is, is not, <, <=, >, >=, !=, ==, +, -, *, /, %, **
```

Thus, are the brackets necessary in
`(curYear+i)%2` ?

Is this probably correct? Or incorrect?
`employees+guests/(chairs+sofas)`

And this?
`yearlyRainfallInLitres/(cubeSize**3)`

Controlling loops

- If you import data for stata, you often encounter invalid observations.
- Python gives you two useful tools for controlling your import loops.
  skip *one* loop with `continue`
  call off the *entire* remaining iterations with `break`

```
# Question:

# Which maximum value of i will be printed

for i in range(0,5):

    print("i is %d" % i)

    if i>2:

        continue

    if i>2:

        break

print("end of loop")

# With break and continue exchanged:

# Which maximum value of i will be printed

for i in range(0,5):

    print("i is %d" % i)

    if i>2:

        break

    if i>2:

        continue

print("end of loop")
```

More on dates

- Construct any date

- Find the weekday - as number and as string
- Find the number of the week of the year

```
import datetime as dt


# We already know the function datetime.now() to obtain the current date.
# With datetime(year,month,day,hour,minute) we can construct any date
birthDayDate = dt.datetime(2017, 2, 25, 0, 0)



# Such variables of type datetime have various methods attached to themselves.
# A method is a function, that receives its arguments by a preceding object-variable,
# such as employee.entryDate()

# Examples
# The method strftime converts a datetime to a string in a particular format.
# The format codes %d, %m etc. are different from format string codes above.
print("Short format is '%s'" %  birthDayDate.strftime("%d.%m.%Y"))

print("Weekday name is '%s'" % ( birthDayDate.strftime("%A") ))

print("  That is the %dth day in the week, starting from monday." % ( birthDayDate.weekday()+1 ))

print("It is the %dth week of the year." % ( birthDayDate.isocalendar()[1] ))

hundredDaysLater = birthDayDate + dt.timedelta(days=100)
print("Hundred days later: '%s'" %  hundredDaysLater.date())
```

Exercise: Influence the loop index

Can you influence the looped index variable in the loop body?

- Write a loop
- Output i
- Change i to a previous value
- Output i
- Try changing i to a value larger then the range.

What value has i after the loop?

```
i = 99


for i in range(0,3):
    print(i)
    i=-2


print(i)


for i in range(0,3):
    print(i)
    i=11
```

Comprehensive exercise 1 - loops and conditions - years and age

- Loop from 10 years previous to 10 years into the future
- Enter *your* name and birth year
- Output your age at every year
- Print correct past tense, present and future tense.
  [was, is, will be]

- Write your output only every three years
- Always report "round" birthday years,
  (those divisible by ten)
- Your output should look similar to this:

```
... ... ... In 2014, Bob was 31 years old. In 2017, Bob is 34 years old. In
2020, Bob will be 37 years old. In 2023, Bob will be 40 years old. 2023 is a
round birthday. In 2026, Bob will be 43 years old. ... ...
```

```
# Extend this code

import datetime as dt


name = "Bob"

birthYear = 1987

cur = dt.datetime.now()
```

```
curYear = cur.year


for i in range(-6, 10):
    print("In %d, %s is %d years old." % (i, name, i))
```

Comprehensive exercise 2 - loops and conditions - weekdays

- Take the loop from previous exercise
- Print the *weekday* of your birthday.
- Report, if the your birthday falls on a sunday.
- Output should look like this:

```
... ... ... In 2017, I have my 34th birthday. Weekday Number is 5. Weekday Name
is Saturday In 2018, I have my 35th birthday. Weekday Number is 6. Weekday Name
is Sunday Birthday falls on Sunday. Invite aunt Erna! In 2019, I have my 36th
birthday. Weekday Number is 0. Weekday Name is Monday ... ...
```

```
name = "Alice"

yearOfBirth = 1983


import datetime as dt

yr = dt.datetime.now().year




for i in range(-12, 12):
    birthday = dt.datetime(yr+i, 2, 25, 0, 0) # construct my birthday
    #     if (i%5)==0 :
    print("In %d, I have my %dth birthday. Weekday Number is %d. " % (yr+i, yr-yearOfBirth+i, birthday.w
eekday() ), end=' ')
    print("\tWeekday Name is %s" % ( birthday.strftime("%A") ))
    if birthday.weekday() == 6:
        print("\tBirthday falls on Sunday. Invite aunt Erna!")
```

Exercise - forever loops

Run this on your **local** machine

- How could a simple forever loop look like?

- How could you exhaust the your computer memory?

Observe CPU and memory during runtime.
Use for instance the Windows task manager.

```
import sys


# sys.argv is a list containing the command line arguments


for i, arg in enumerate(sys.argv[1:]):    # chop off first arg, for it is the program file name
    print("arg %d is: %s" % (i+1, arg))


mode = ""



if len(sys.argv) > 1:
    mode = sys.argv[1]


l = [1,2,3] # a list



if mode=="cpu":
    # CPU load
    i = 2
    y=0
    #while i > 1:                   # not obvious
    for j in range(1000*1000*1000):  # obvious
        y=y+1



if mode=="memory":
    # Memory load
    i = 2
    #while i > 1:                   # not obvious
    for j in range(1000*1000*1000):  # obvious
        l.extend([i])
```

```
        l.extend(l[:500])
        print(len(l))


print("end of program")
#  The solution to forever-loops is to put in a safety counter

safetyCounter = 0

for i in range(1000*1000*1000):
    for j in range(1000*1000*1000):
        safetyCounter += 1  # count it up in the innermost loop
        #
        # do stuff
        #
        if safetyCounter > 200:
            raise Exception("too much")
            quit() # caught by the Jupyter IDE
```

Data structures: `list` and `dictionary`

When working with data you will need two essential data structures.

`list`

```
# List
myList = ['a', 'b', 'c']
myList.append('d')         # adding one element
myList.extend(['e','f'])   # adding multiple elements - a second list
myList[5]='ff' # changing an item; the sixth item
del myList[3] # remove an item: delete the forth item
print(myList)
# What is terrible about ?
# list = ['a', 'b', 'c']
print(len(myList)) # indize 0...4
```

```
myList[5]='g' # changing the sixth item fails, since it does not exist
# create an emtpy list; and check type
if type([]) is list:
    print("[] looks unusual - but is just an empty list")
# Slicing lists
firstTwo = myList[:2]
print(firstTwo)

lastThree = myList[-3:]
print(lastThree)
# Looping lists
# Most simple
for val in firstTwo:
    print("val is %s" % (val))
print()

# With index; "canonical", "idiomatic"
for index, val in enumerate(firstTwo):
    print("idx %-3d val %s" % (index, val))

# Strings can be treated like lists of characters
# What is the first element of "Hello"[0]
print("Hello"[0])



for letter in "Hello":
    print(letter, end="-")
```

`list` characteristics

- indices are of type integer
- values can be any type
- implicitly sorted by index

- iteration and random access by index are cheap

- lists have a defined length that must be explicitly changed
- with `append` or `extend` or `del`.
  Changes of size can be expensive.

`dict`

```
# Dictionary
myDict = {'use':'benutzen', 'put':'legen', 'get':'holen'}
myDict["run"] = "rennen"  # add one element
del myDict["put"]  # delete one element

# We dont use the syntax of "for key, value in d.items()"
# stackoverflow.com/questions/17793364 - second answer
for index, key in enumerate(myDict):
    print("idx %-3d key %-8s => val %s" % (index, key, myDict[key]))

# Test whether an item (key) exists
if "use" in myDict:
    print("'use' exists and points to '%s'" % myDict["use"])
# Non existing keys blow up
print(myDict["missing"])
```

`dict` characteristics

- keys can be of type integer, string or date...

- values can be any type - same as list.

- the keys are `unique` -
  there can not be two elements myDict['EU']
  the values can repeat.

- iteration and random access by key are cheap
  but iteration is **unsorted**

- the length of a dict changes with every
  `myDict['newKey']=val` or
  `del myDict['oldKey']`.
  Such changes to the size are cheap.

Dictionary revisited

Once again, `dict` juxtaposed against `list`

```
myList = [] # new list

myDict = {} # new dict


myDict = {"US": 19,  "EU": 17}  # new dict with some keys and values; Gross Net Product in trillions


myDict["EU"] = 17     # no change over previous dictionary

myDict["China"] = 12

myDict["Japan"] = 5

myDict["Germany"] = 3.6


del myDict["Germany"] # deletion similar to list


if "Vanuatu" in myDict:

    print("Vanuatu is not in the dictionary data set")


print("GNP of US is %.0f Trillion." % myDict["US"])

print("Dictionary has %d entries." % len(myDict))


# Iterate - only accidentally sorted

for index, key in enumerate(myDict):

    print(index, key, myDict[key])
```

Testing for `list` and `dict`

After `string`, `int` and `float`, we now know two more types.
Keep the type aspect in mind - and remember how to test for the type. Avoid `list` and `dict` as variable names,
because these are predefined variables containing the respective type value.

```
whatIsIt1 = []     # minimal, empty list

whatIsIt2 = {}     # minimal, empty dict


if type(whatIsIt1) is list:

    print("is a list")
```

```
if type(whatIsIt2) is dict:
    print("is a dictionary")
```

Nesting of data structures - printing composite variables - variable names pitfall

```
# Any wild combination of dicts and lists are possible:
nested = [{"name": "Alice", "age":34, "height": 1.7, "salaries": {1998: 12000, 2007: 14000}}, {"name": "
Bob", "age":7, "height": 0.8, "salaries": {2007: 20, 2009: 30}}]
print(nested)
```

Many web services deliver data as nested combinations of lists and dicts.
For instance google search data.

We would unpack such data
with nested loops.

Exercise: Traverse nested lists and dicts

- Traverse (drill down) into to following composite of lists and dicts.
- Use multiple nested loops.
- Use print to output the loop index, the key (if applicable) and the value.
- Your output should look approx. like this:

```
person record  0
    name       Alice
    age        34
    height     1.7
             0   salary 1998: 12000.00
             1   salary 2007: 14000.00

person record  1
    name       Bob
    age        7
    height     0.8
```

```
0    salary 2007:    20.00

1    salary 2009:    30.00
```

Solution

```
# Notice the naming system.

# We have on each level of nesting: index, key, value.

# We name the loop variables i,k,v - combined with nesting depth.



for i1,v1 in enumerate(nested):

    print("\nperson record %2d" % i1)

    for i2, v2 in enumerate(v1):

        if v2 != "salaries":

            print("\t%-10s %s" % (v2, v1[v2]))

    for i3,v3 in enumerate(v1["salaries"]):  # enumerate the salararies data points

        print("\t\t\t%2d   salary %s: %8.2f $" %(i3,v3,v1["salaries"][v3]))
```

For merely *printing* such nested structures,
the `json` module provides a comfort function.

```
import json  # for printing nested lists-dicts


myList = ['a', 'b', 'c', 'd', 'e', 'f']

jd = json.dumps(myList, indent=4)

print(jd)

jd = json.dumps(nested,sort_keys=True, indent=4)

print(jd)
```

Pitfall: accidentally overwriting Python elements

```
# What is wrong with this?

list = 4



import datetime as dt

dt = 4
```

```
True = 4
```

```
def = 4
```

Partial answer (https://www.programiz.com/python-programming/keyword-list)

But not only keywords should not be overwritten.

Imported modules and builtin types and constants.

tuples

tuples are lists which cannot be changed after first creation.

```
myTuple = (1, 2, 3)
```

```
confusingSyntaxButItsATuple = (1,)
```

Functions, errors and `try-except` blocks

Functions

- A function starts with **def**
- Then the function's name
- Argument(s) in Brackets
- Don't forget the colon
- Function body must be indented
  - First line indent governs all following indents
  - Outdent line ends the function
- `return` is a Python keyword to designate the value of the function

```
def multiply(factor1, factor2):
    return factor1 * factor2


print(multiply(6,7))
print(multiply(0.01,100))
# Mostly we have 'positional' arguments
print(multiply(1,2))


# But we can have named arguments too.
# Named arguments give optional instructions to functions.
```

```
# The print() function has an optional end parameter:
print("Alice", "Bob", end=";")
print("Cesar", end=";")

# The print() function also has an optional file parameter:
print("Dora", file=open("./out.txt", 'wt'))
```

Errors

What kind of errors occur during program execution?

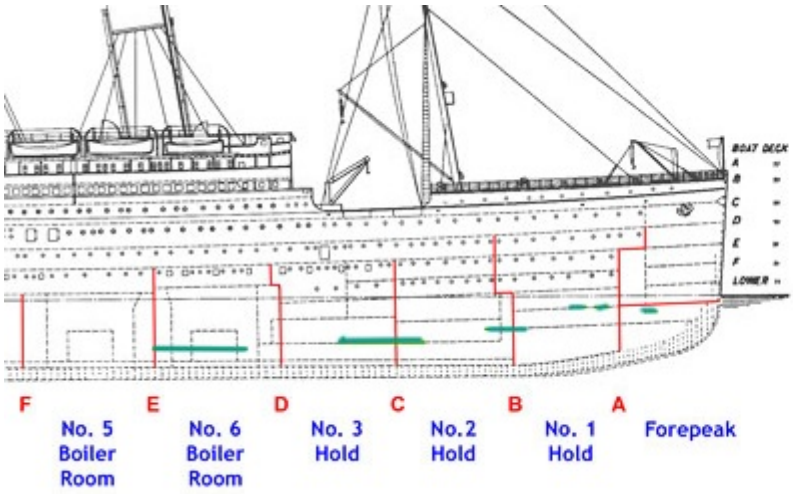
- Conversion
- Division by zero
- File not found, file locked
- Out of memory
- Network connection timeout
- Operation / function not applicable to type
  `"Alice" * 2`
- Key not found in dictionary

What happens, if an error occurs?

The entire program ends

Error handling in Python

You will need to create compartments to contain errors in your program.



Analogy (imperfect): Barriers containing leaks and explosions in the Titanic. Compartments. Firewalls

try ... except ... block

- `try` wraps a block of code
- *All* errors, that could occur in the enclosed block, are caught.
- The program does *not* terminate.
- Instead, the program switches to the except block.
- `try` and `except` require each other. You cannot have one without the other.

```
# Example one - remedy conversion errors
try:
    convertite = float("0.22") # error prone code.
    undefined = 1/0
except Exception as e:
    print("program caught an error: ",str(e))
    # remedial action
    convertite = 0.0
    undefined = float('inf')
print("program continues...")
```

- Since an exception may occur in a large block of code, the offending line is not easily attributable.
- We want the row in which it was caused.

```
import traceback
```

```
try:
    convertite = float("0.22") # error prone code.
    undefined = 1/0
except Exception as e:
    print(traceback.format_exc())
    # remedial action
    convertite = 0.0
    undefined = float('inf')
print("program continues...")
# Example two - keeping a loop going
for i  in range(5):
    try:
        print(1/(i-2))
    except Exception as e:
        print(str(e))
        print(traceback.format_exc())

# Example three - connecting to email server - to send an email
import smtplib
try:
    s = smtplib.SMTP('hermes.zew-private.de', timeout=4)
    s.sendmail("me@zew.de", "you@zew.de", "Hello")
except Exception as e:
    print("Error connecting to email server: ", str(e))
    print(traceback.format_exc())

# Example four - missing keys in dictionaries
try:
    myDict = {}
    print(myDict["missing"])
    print(myDict["unknown"])
except Exception as exc:
    print("key not found:", exc)
```

```
    print(traceback.format_exc())


# Example five - give verbose assistance if a module must be installed
# Remember the math module for math.isnan() previously?
try:
    import math
    print("module 'math' imported")
except Exception as e:
    print(str(e))
    print("missing python module 'math' - open command window and type 'pip install math'")
    print(traceback.format_exc())
    quit()

print("square root of 2 is %f" % math.sqrt(2))
# Break out of multiple - nested loops
try:
    for i in range(5):
        for j in range(2):
            for k in range(2):
                print(i,j,k)
                if i>1:
                    #break # only leaves the inner most loop
                    raise Exception("Leave all the loops!")
except Exception as e:
    print(traceback.format_exc())

print("All the loops were broken off.")
```

Some usage for exceptions are

- Remedy conversion errors => set fallback value
- Long running loops: Wrapping the loop body => just skip
- Missing dict keys => skip or fallback value

- Module import: Module not installed => explain before terminate
- Leaving several nested loops
- File locked by somebody else => change filename, retry
- File already exists => change filename, retry
- Network file: Access timeout => backoff in increasing intervals
- Crawling: Element not found => skip, continue

Function - converting the single argument to float

```
# We can now solve the problem of converting dirty data for Stata
# to float numbers, without crashing our import program.
# We wrap the conversion into a function, which contains any conversion errors.
def toFloat(value):
  try:
    float(value) # this might trigger an exception.
    return float(value)  # previous line is redundant
  except Exception as e:
    #print(str(e))
    #print(traceback.format_exc())
    return 0.0

print(toFloat("  0.2"))
print(toFloat("  0.2 kg tomatoes"))


# print("My name is %s. My name expressed as floating point number is %f" % (name, toFloat(name)))
```

Pitfall: list and dict inside functions

- What goes on inside a function should not affect the calling code.

$$y = x^2$$

- x should not be changed.

- In Python, changes made to "vectors" or "matrices" in functions persist outside the function.

```
# Simple scalar variable are copied to the function.
# "call by value"
def translateWord(argStr):
    argStr='Maria'

name = 'Mary'
print(name)
translateWord(name) # take name and change it
print(name)    # does the local change spill over into the outer code?
# A list is directly given to the function.
# "call by reference"
def translateList(argL):
    argL[2]='Maria'



l = ['Peter','Paul','Mary'] # a list
print(l)
translateList(l)
print(l) # does the change to the name spill over into the outer list?
import datetime as dt

def changeADate(argDate):
    argDate=dt.datetime(1492, 6, 15, 0, 0)

someDate = dt.datetime(2017, 2, 25, 0, 0)
print(someDate)
changeADate(someDate)
print(someDate)    # does the local change spill over into the outer code?
```

- Primitive, scalar variables (`int`, `string`)
  dont break out of inner contexts (loops, functions, if-bodies)

- But changes to *non-scalar* variables
  - like `list` and `dict`

    - composite variables like `datetime`
      do 'filter upwards'

- Prevent unwanted changes by creating `deep copies` of `lists`, `dicts` etc.

```
import copy


l2 = copy.deepcopy(l)

l2[2] = 'Mary'


print(l)

print(l2)
```

String manipulation

Double quotes - single quotes - control characters / escaping

```
'''

The 'u' prefix indicates the utf-8 convention.

We may omit it, since it is the default in our modern Python version 3.x .


'''

s = u"All digits and letters are allowed.  \t \" ' - German Umlauts öüä - but also Czech letter měkké - even Chinese characters 原"

print(s)


# The double quote must be 'escaped'.

# Backslash must be escaped.

# \t - tabulator and \n - new line are the most useful 'control characters'



# Single quotes work as well.

# Now internal double quotes are allowed - but internal single quotes must be escaped.

s = u'All digits and letters are allowed.  \t \' " - German Umlauts öüä - but also Czech letter měkké - even Chinese characters 原'

print(s)
```

```
# The 'r' prefix prevents all control characters.
s = r"All digits and letters are allowed.  \t \" ' - German Umlauts öüä - but also Czech letter měkké - even Chinese characters 原"
print(s)
```

replace

```
# Column names from Orbis Bank Focus data source:
s = "Anteile an verbundenen Unternehmen, Ausleihungen an Unternehmen, Wertpapiere des Anlagevermögens, "



# We want to make these columns acceptable to stata
# => No spaces, no Umlaute, shorter

s = s.replace("ö","oe")
s = s.replace("ß","ss")

print(s)
# We might put multiple replacements into a dictionary
replacements = {"ä":"ae", "Ä":"Ae", "ö":"oe", "Ö":"Oe", "ü":"ue", "Ü":"ue" , "ß":"ss"}
replacements[" "] = "_"
replacements["Anteile"] = "Ant"
replacements["Unternehmen"] = "Unt"
replacements["Anlagevermoegen"] = "AV"
print(replacements)
# Now loop over them
for i, k in enumerate(replacements):
    print("%3d replacing %2s with %s" % (i, k, replacements[k]))
    s = s.replace(k, replacements[k])

print("Result: ",s)
```

Excursion: Sorted iteration of a dictionary

- Dictionaries are iterated in *random* order.
- We *cannot rely* on the creation order.
- The function `sorted()` extracts a sorted list of keys
  `listOfKeys = sorted(myDict)`
- Now we may iterate over the sorted keys

Advanced question: Why is the dictionary not sorted in the first place?

```
sortedKeys = sorted(replacements)
for index, key in enumerate(sortedKeys):
    print("%3d replacing %2s with %s" % (index, key, replacements[key]))
    s = s.replace(key, replacements[key])

print("Result: ",s)
# That is alphabetical :(
# Is there no way to preserve the creation order?
# lists preserve the creation order, but have only one value, right?
# But that value can be a dictionary.
# Therefore we might write:
replacements = [{"ä":"ae"},{"Ä":"Ae"},{"ö":"oe"},{"Ö":"Oe"},{"ü":"ue"},{"Ü":"ue"},{"ß":"ss"}]
replacements.extend([{" ":"_"},{"Anteile":"Ant"},{"Unternehmen":"Unt"},{"Anlagevermoegen":"AV"}])

for i, v in enumerate(replacements):
    for i2, k2 in enumerate(v):
        print("%3d replacing %2s with %s" % (i, k2, v[k2]))
        s = s.replace(k2, v[k2])
print("Result: ",s)
```

`split` , `join` and `strip`

- Transform strings into lists
- Transform lists into strings

```
l = s.split(",")
```

```
print(l)
s = s.strip("_")
s = s.strip(",")
s = s.strip()

l = s.split(",")
print(l)
print("--".join(l))
# Lets process our list further
# We cannot and should not change the list which we are iterating.
# Thus we re-assemble it into a new list `l1

l1 = []

for index, elem in enumerate(l):
    elem = elem.strip()
    elem = elem.strip("_")
    if elem.startswith("Ausleihungen_"):
        elem = elem.replace("Ausleihungen_", "Auslh_")
    if elem.startswith("Wertpapiere_"):
        elem = elem.replace("Wertpapiere_", "WertP_")
    if "_an_" in elem:
        elem = elem.replace("_an_", "_")
    if "_verbundenen_" in elem:
        elem = elem.replace("_verbundenen_", "_verb_")
    l1.append(elem)

print(l1)
```

Test for substring - position of a substring

```
# Simple test, whether substring exists:
if 'vermoegen' in 'Anlagevermoegen':
    print('AV contains Verm.')
```

```
# Scanning for a substring position: string.find(s, sub[, start[, end]])
for index, elem in enumerate(l1):
    pos = elem.find("vermoegen")
    if pos > 1:
        elem = elem[:pos+len("vermoegen")]  # truncate everything after 'vermoegen'
```

Upper case, lower case

```
# We could standardize everything on lower case
l2 = []

for index, elem in enumerate(l1):
    changed = elem.lower()
    l2.append(changed)

print(l2)

# Or drop the underscores
```

Using strings as lists of characters

```
"python"[0:2] == "py"
"python"[ :2] == "py"
"python"[2: ] == "thon"
"python"[-2:] == "on"
"python"[-2:-1] == "o"
# We could abbreviate with an inner ellipsis
l3 = []

for index, elem in enumerate(l1):
    if len(elem) > 2:
        changed = elem[:2] + "..." + elem[-2:]
```

```
    l3.append(changed)

print(l3)
```

Advanced alternative: `regular expressions`

Example: Cleanup all trouble making characters

```
import re

# cleanse everything except a-z or 0-9 or '_':
pattern1 = re.compile(r"[^a-z0-9_]", re.IGNORECASE)  # r => raw string \n is not interpreted as new line

s = "Anteile an verbundenen Unternehmen, Ausleihungen an Unternehmen, Wertpapiere des Anlagevermögens, "
# sub(repl, string)
# Returns the string obtained by replacing occurrences of pattern in string with repl.
s = pattern1.sub("_", s)
print(s)


pattern2 = re.compile(r"[__]+")
s = pattern2.sub("_", s)
print(s)
```

Reading and writing files

`Files` used to be stored and read from magnetic tapes.
`Files` still behave like tapes.
Either you read them piece by piece with no going back.
Or you write something - after the last position.

For *random* reads and writes, use `list` or `dict`.

Thus typically, we read from a file into a list or dict.
Then we modify the data.
Finally we save the modified data to a file again.

CSV Files

"Comma Separated Value" Files are the lowest common denominator for data files.

They are human readable with `vscode` or `notepad` or `notepad++`.

They can be used with Excel, with Stata, with Databases and with Python.

We will store our crawled data in them.

```
# Reading a CSV file. Line by line. Splitting each line into its columns.
inpFile = open('inp.csv', 'rt', encoding='utf8')  # a prepared input file - read it as text
for line in inpFile:
    line = line.strip()          # clean up leading and trailing white space
    elements = line.split(";")
    print(elements)
inpFile.close()                  # release the file for other programs
```

- `Opening` a file locks it from other programs writing to it.
- Files are either *read from* or *written to*.
  Thus we open our file as *r* or *w*.
- We deal with text files (thus the *t*).
- Reading and writing is done in chunks - in our case in `lines`.
- To read the lines, we loop over the file variable.
- Lines contain data as a long string, separated by `;` or by `\t` (tabulator).
- Files must be closed (freed),
  otherwise they might appear `locked by another user`
  and the operating systems capability to provide access to files is exhausted after a while.

```
# Imagine you have some data, and you want to write
# it to a file.
outFile   = open("out.csv", 'wt', encoding='utf8')
for i in range(0,3):
    outRow = []
    for j in range(0,5):
        outRow.append("c%d%d" % (i,j))
    # print(outRow)
    outFile.write(";".join(outRow))
    outFile.write("\n")
outFile.close()
```

- Notice the *w* for write in the file open function.
- Opening for write erases the previous content
- `outFile.write(...)` is only possible, because we opened the file with *w*.
- Writes are appended to the end of the file.
- There is no 'going back'

In [23]:

```
# All kinds of exceptions can occur
# when reading from or writing to files.
# Thus we enclose it into a try-except barrier.
import traceback
```

```
def cleanUp():
    inpFile.close()
    outFile.close()



try:
    inpFile = open('inp.csv', 'rt', encoding='utf8')
    pass
    pass
    outFile = open("out.csv", 'wt', encoding='utf8')
    pass
    pass
except Exception as e:
    print(traceback.format_exc())



cleanUp()  # make sure, files are always closed
# Reminder of split and join

print("1;2;3".split(";") == ["1", "2", "3"])

print(";".join(["1", "2", "3"]) == "1;2;3")
```

Fully functional example

- We read a csv file line by line

  ```
  Growth;GDP
  0.06;6.2
  0.06;6.572
  0.05;6.9006
  ```

- Each line is split into elements
- Elements are converted to float,
  floats are multiplied by two
- Multiplied elements are reformatted as string,
  strings are joined into lines again.
- Lines are written into out-twofold.csv

Solution

```
import traceback


def toFloat(value):
    try:
        return float(value)
    except Exception as e:
        return 0.0


def cleanUp():
    inpFile.close()
    outFile.close()



try:
    # reading inp.csv into a list of lists called inpList
    inpFile = open('inp.csv', 'rt', encoding='utf8')
    inpList = []
    for i,line in enumerate(inpFile):              # loop over infile
        if i == 0:
            continue                               # skip header line
        elements = line.split(";")                 # split line by semicolon
        inpList.append(elements)                   # store result in inpList
```

```
    print(inpList)



    # multiply every element and putting the result into outList
    outList = []
    for i,elements in enumerate(inpList):          # loop over inpList
        outRow = []
        for i2, cell in enumerate(elements):       # loop over each cell
            cellAsFloat = toFloat(cell) * 2        # convert to float, multiply
            outRow.append( "%5.2f" % cellAsFloat)  # convertr back to string
        outList.append(outRow)                     # store in outlist
    print(outList)



    # put the contents of outList into a file
    outFile = open("out-twofold.csv", 'wt', encoding='utf8')
    for i3, v in enumerate(outList):               # loop over outlist
        outline = ";".join(v) + "\n"               # write each row to outFile
        print(outline)
        outFile.write(outline)



except Exception as e:
    print(traceback.format_exc())



cleanUp()
```

Sending email if long running programs fail or finish

Writing a function to send an email.

Packaging this function into our own `module`

Importing and calling the function.

```
'''
```

```
    Usage:
    from send_email import sendEmail
    ...
    sendEmail("your python script finished","no exception occured","yourname@zew.de")
    ...

   This is a primitive module, containing one function
   to send an email from inside the ZEW Ltd organization.


   The module is educational, demonstrating
   reusing some functionality across many Python scripts.
'''

import traceback

import smtplib  # Module for sending an email; smtp is a www standard.
from email.mime.text import MIMEText   # encoding the email body

def sendEmail(subj, msgBody, address):
    try:
        sender     = address
        recipients = address  # could be list
        body = '%s\n(end of message body)\n' % (msgBody)
        msg = MIMEText(body)  # string to email body - ascii only
        msg['Subject'] = '%s (subject line)' % (subj[:40])
        msg['From'] = sender
        msg['To']   = recipients

        # Send the message via our own SMTP server.
        # But don't include the envelope header.
        # c = smtplib.SMTP('localhost')
        c = smtplib.SMTP('hermes.zew-private.de',timeout=4)
        c.sendmail(sender, [recipients], msg.as_string())
```

```
        c.quit()
    except Exception as e:
        print("Exception while sending Email")
        print(traceback.format_exc(2))
```

- Save above code as file named `send_email.py`
- Then use the file as `module`

```
from send_email import sendEmail sendEmail("your python script finished","no exception occured","yourname@zew.de")
```

```
from send_email import sendEmail
sendEmail("your python script finished","no exception occured","yourname@zew.de")
```

Webserver

```
from http.server import BaseHTTPRequestHandler, HTTPServer
from os import curdir, sep
import time
import traceback


class MyHandler(BaseHTTPRequestHandler):

    def do_GET(self):   # GET

        try:

            print("serving %s" % self.path)

            self.send_response(200)  # Send response status code

            self.send_header('Content-type', 'text/html;charset=utf-8')  # Send headers
            self.end_headers()


            # Root responsse
```

```
    if self.path == "/" or self.path == "":
        message = "Hello world!"
        self.wfile.write(bytes(message, "utf8"))  # String to bytes with utf-8 encoding
        return


    # Search for pertinent local file and show its contens
    # INSECURE
    p = self.path
    if p[:1] == "/":
        p = p[1:]
    p = curdir + sep + p

    try:
        f = open(p)
    except Exception as e:
        s = traceback.format_exc()
        self.wfile.write(bytes(s, "utf-8"))
        return

    cnt = f.read()
    bCnt = bytes(cnt, "utf8")
    self.wfile.write(bCnt)
    f.close()
    return


except Exception as e:
    print(traceback.format_exc())
    s = traceback.format_exc()
    self.wfile.write(bytes(s, "utf-8"))
```

```
try:
    print('about to start http server...')
    ipPort = ('127.0.0.1', 8086) # the 127.0.0.1 constrains access to our webserver to our own computers
    httpd = HTTPServer(ipPort, MyHandler)
    print('               http server started at %s...' % str(ipPort))
    httpd.serve_forever()  # break this with CTRL+Pause


except Exception as e:
    print(traceback.format_exc())
```

Crawling

Please observe the `fair use` custom.

Use the websites only like a fast human operator would.
One page every three seconds should be a good orientation.

Don't send 20 requests per second to a website!

ZEW Ltd. offers internal cloud computers (http://intranet.zew.de/pages/viewpage.action?pageId=69831709) for researchers.
You will have admin rights there and can crawl 24x7.

Similar cloud computers can be rented by the hour from Amazon's data centres (AWS) (https://aws.amazon.com/ec2/pricing/on-demand/).

Some services, like google search, defend themselves against more than casual usage.
In such case, use a VPN (https://en.wikipedia.org/wiki/Virtual_private_network) like Cyberghost.
Otherwise you might impair the service access of all colleagues at ZEW (or Uni Mannheim).

HTML Structure

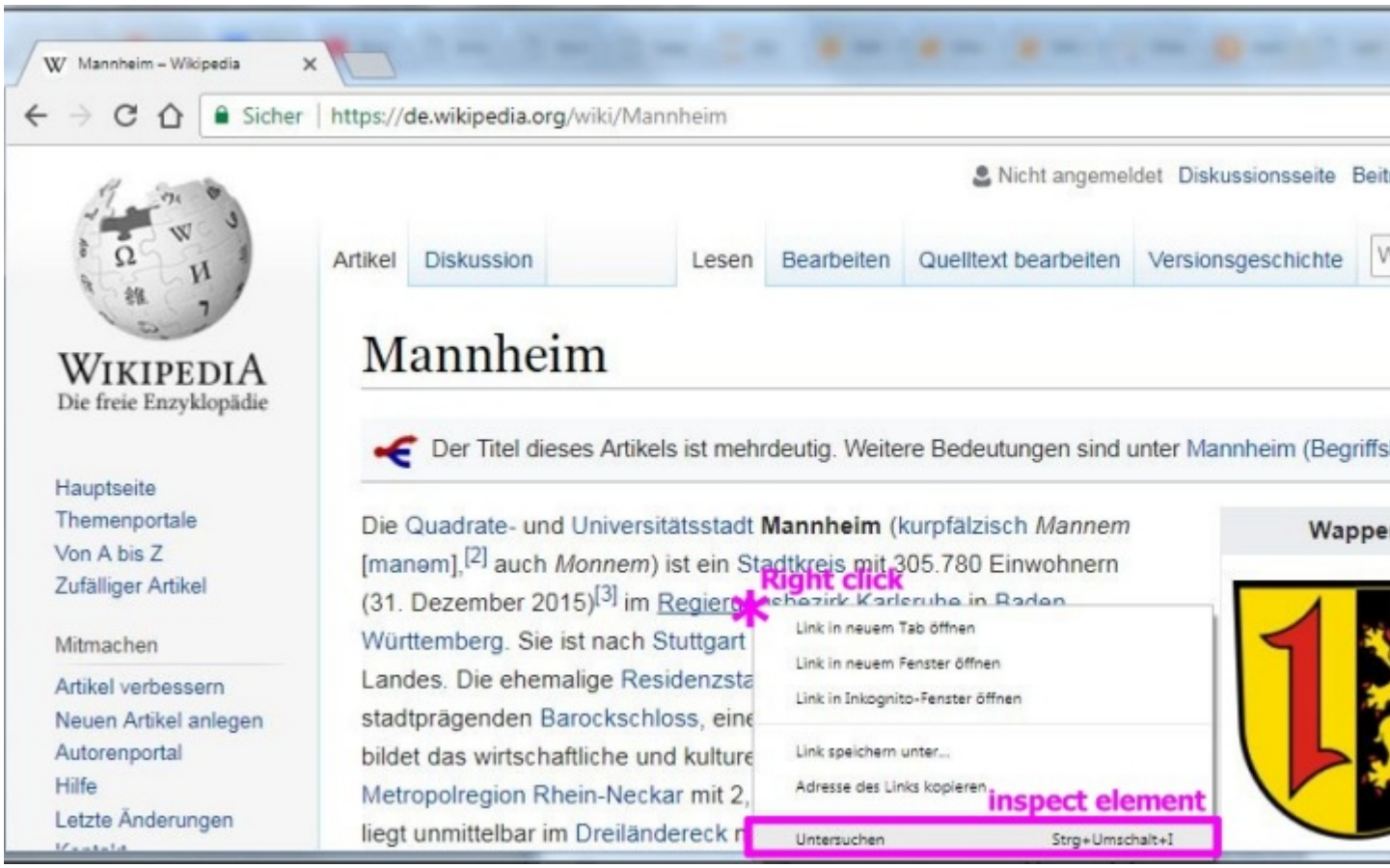
To effectively target the information we search,
we need some analytical understanding of HTML.

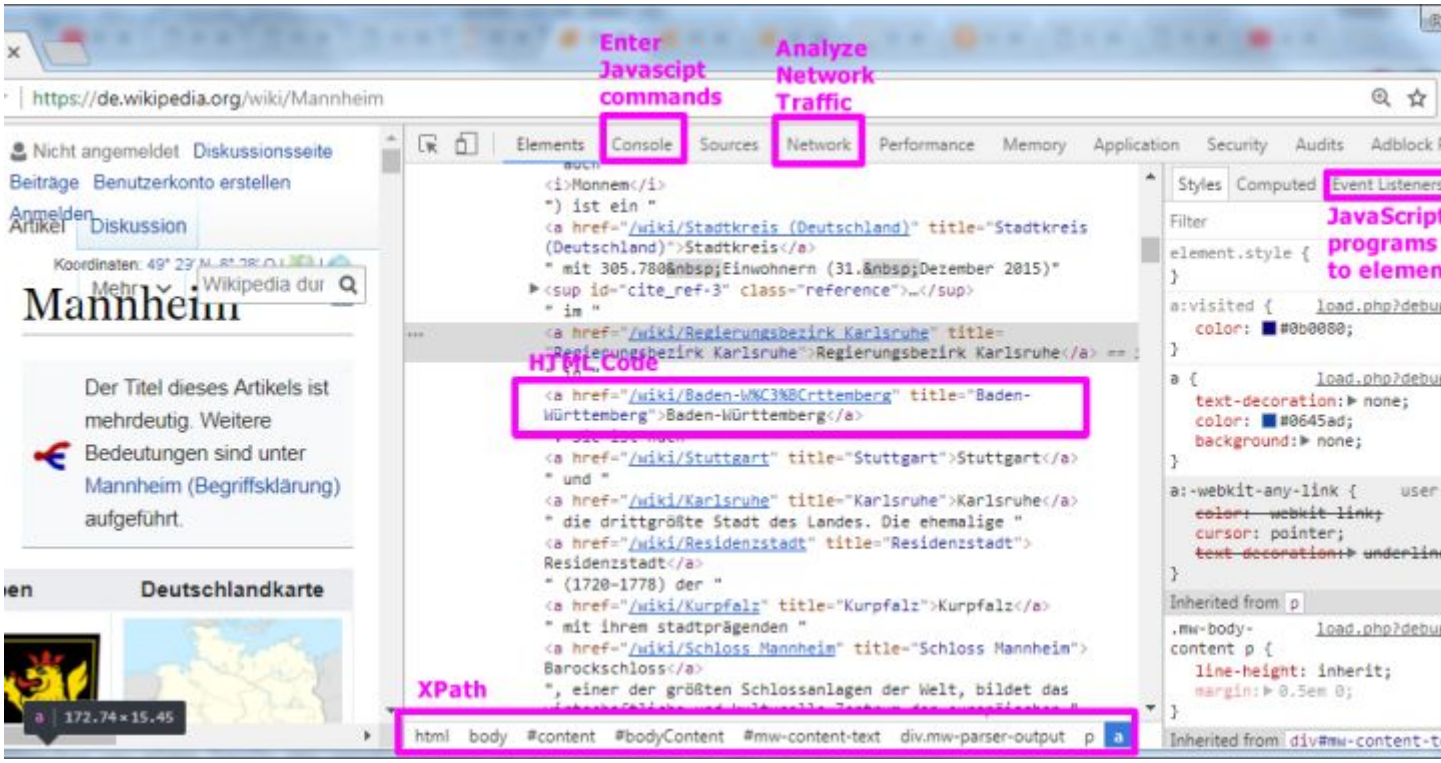
We start with a primitive HTML file.

Source (http://semalloc.zew.de:8000/edit/00/030-html-structure-intro.html)

Result (http://semalloc.zew.de:8000/view/00/030-html-structure-intro.html)

Analyze HTML pages using the chrome browser





A most simple introductory example

```
import requests # http request


from lxml import etree
from lxml.cssselect import CSSSelector


cities = ["Berlin", "Hamburg", "München", "Nürnberg", "Mannheim"]


for index1, city in enumerate(cities):
    try:
        if index1 > 0:
            break # for development and testing; only one page
        resp = requests.get("https://de.wikipedia.org/wiki/%s" % (city))
        print("looping %d: %-20s - response code is %d - %4d bytes" %  (index1, city, resp.status_code,
len(resp.text)))
        # print(resp.text[:120] + "..." + resp.text[-120:])
        # resp.headers contain meta information about the response
        if "noticed some unusual activity coming from your computer network" in resp.text:
            print("CAPTCHA")
            quit()
        rootNode = etree.HTML(resp.text)  # text into graph structure
        expr = "//td"
        tdNodes = rootNode.xpath(expr)    # simple query of graph structure
        for index2, node in enumerate(tdNodes):
            if node.text is None:
                continue
            cnt = node.text.strip()
            if cnt != "":
                print(cnt)
    except Exception as e:
        print(str(e))
        continue
```

Crawling using the module `beautiful soup`

Beautiful soup is a classic crawling method.

If you are already using it, stick with it for simple tasks.

However, you cannot use XPath expressions with beautiful soup.
Therefore we only give following example script, and then move on.

```
# -*- coding: utf-8 -*-

import sys

import time

import traceback


import requests


from send_email import sendEmail

from random import randint


from bs4 import BeautifulSoup  # pip install bs4, pip install html5lib


# Suppress pesky redundant warnings during webpage loading

import warnings

warnings.filterwarnings("ignore", category=UserWarning, module='bs4')



# If you having troubles with special characters

if (sys.stdout.encoding).upper() != "UTF-8":

    print("PYTHONIOENCODING is ", sys.stdout.encoding)

    print("set PYTHONIOENCODING=UTF-8")

    quit()



cities  = ["Berlin","Hamburg","München","Nürnberg"]



outFile = open("out-city-inhabitants-bs4.csv", 'wt', encoding='utf8')
```

```
outFile.write("city;inhabitants\n")


def cleanUp(msg):
    print(msg)
    # sendEmail(msg, msg, "yourname@zew.de")
    outFile.close()


for index1, city in enumerate(cities):
    try:
        if index1 > 1:  # for testing
            break

        urlMain = "https://de.wikipedia.org/wiki/%s" % city
        print(("%5d loading url ... %s - %s" % (index1,urlMain,city)))
        # resp = requests.get(url=urlMain, params=dict(), timeout=25)
        resp = requests.get(url=urlMain, timeout=25)
        print(("\tStatus code is %s - Encoding is %s" % (resp.status_code, resp.encoding)))

        # r.encoding = 'ISO-8859-1'
        if "noticed some unusual activity coming from your computer network" in resp.text:
            print("CAPTCHA")
            break

        #pos = resp.text.find('<body')
        #print("\t response: ",resp.text[pos:pos+100])


        # https://www.crummy.com/software/BeautifulSoup/bs4/doc/
        soup = BeautifulSoup(resp.text,"html5lib")
        tagIdx = 0
        row = [city]
        for tag in soup.find_all('td'):
```

```
            #print(tag.get('href'))
            ct = tag.contents
            if type(ct) is str:
                print("td contents:",ct[:100])
            else:
                # tag.get_text()
                for idx2, val in enumerate(ct):
                    subtree = BeautifulSoup(str(val),"html5lib")
                    s = subtree.text
                    s = str(val)
                    s = s.replace('\n',' ')
                    s = s.strip()
                    if ("Einwohner:" in s or ">Einwohner</a>" in s or "Einwohner" in s) and not("über" i
n s) :
                        print("\t %4d td contents: %s" % (tagIdx, s[:110]))
                        #par = tag.parent
                        #nextTd = par.findNext('td')
                        seq = tag
                        for q in range(1,999):
                            seq = seq.next_sibling
                            if seq.name == "td":
                                cnt = seq.text
                                #cnt = str(seq).strip()
                                cnt = cnt.replace('\n',' ')
                                cnt = cnt.strip()
                                if cnt != "":
                                    print("\t\t",cnt[:70])
                                    row = row + [cnt]
                                    # print(row)
                                    outRow = ";".join(row) + "\n"
                                    outFile.write(outRow)
                                break
            tagIdx += 1
            # if tagIdx>5:
```

```
            #     break

        outRow = "%s;%s\n" % (city,"no results")
        outFile.write(outRow)



        slp = float(5+randint(0,40))/100
        print(("Sleeping %f" % slp))
        time.sleep(slp)


    except Exception as e:
        print("exception in loop body: %s" % str(e))
        print(traceback.format_exc())
        # sendEmail("exception", traceback.format_exc(), "yourname@zew.de")
        continue



cleanUp("regular finish")
```

Crawling using `urllib3`, `lxml` and `xpath`

In this example we will crawl all links on a wikipedia page

```
# -*- coding: utf-8 -*-
import sys
import time
import requests
import traceback

from send_email import sendEmail
from random import randint

import urllib3
```

```
# stackoverflow.com/questions/11465555
# Suppress pesky redundant warnings during webpage loading
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

from lxml import etree
from lxml.cssselect import CSSSelector


# If you having troubles with special characters
if (sys.stdout.encoding).upper() != "UTF-8":
    print("PYTHONIOENCODING is ", sys.stdout.encoding)
    print("set PYTHONIOENCODING=UTF-8")
    quit()


cities = ["Berlin", "Hamburg", "München", "Nürnberg"]

ofn = "out-city-urls-lxml.csv"
outFile = open(ofn, 'wt', encoding='utf8')
outFile.write("city;urls\n")



def cleanUp(msg):
    print(msg)
    # sendEmail(msg, msg, "yourname@zew.de")
    outFile.close()



pool = urllib3.PoolManager()


for index1, city in enumerate(cities):
```

```
try:
    if index1 > 1: # for testing
        break

    # lxml.de/tutorial.html#using-xpath-to-find-text
    # urllib3.readthedocs.io/en/latest/user-guide.html
    urlMain = "https://de.wikipedia.org/wiki/%s" % city
    print(("%5d loading url ... %s - %s" % (index1, urlMain, city)))
    resp = pool.request("GET", urlMain)
    print(("\tStatus code is %s" % (resp.status)))

    #r = str(resp.data)
    r = resp.data.decode('utf-8')
    if "noticed some unusual activity coming from your computer network" in r:
        print("CAPTCHA")
        break
    #pos = resp.text.find('<body')
    #print("\t response: ",resp.text[pos:pos+100])

    root = etree.HTML(r)
    expr = "//a/@href"
    expr = "//a[string-length(@href)>60]/@href"
    print(expr)
    nodes = root.xpath(expr)

    for idx2, nd in enumerate(nodes):
        text = str(nd)
        text = text.strip()
        if text.startswith("#"):
            continue
        #print(text)
        print(u"\t\tnode no %3d - |%s|" % (idx2, text[:90]))
        outRow = "%s;%s\n" % (city, text)
        outFile.write(outRow)
```

```
            if idx2 > 10:
                break

        slp = float(5+randint(0, 40))/500
        print(("Sleeping %f" % slp))
        time.sleep(slp)

    except Exception as e:
        print("exception in loop body: %s" % str(e))
        print(traceback.format_exc())
        # sendEmail("exception", traceback.format_exc(), "yourname@zew.de")
        continue


cleanUp("regular finish")
```

Crawling using `urllib3`, `lxml` : advanced use of `xpath`

In this example we will retrieve a table cell, based on the content of *another* table cell.

```
# -*- coding: utf-8 -*-
import sys
import time
import requests
import traceback

from send_email import sendEmail
from random import randint


import urllib3
# stackoverflow.com/questions/11465555
# Suppress pesky redundant warnings during webpage loading
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

```
from lxml import etree
from lxml.cssselect import CSSSelector


# If you having troubles with special characters
if (sys.stdout.encoding).upper() != "UTF-8":
    print("PYTHONIOENCODING is ", sys.stdout.encoding)
    print("set PYTHONIOENCODING=UTF-8")
    quit()


cities  = ["Berlin","Hamburg","München","Nürnberg"]


ofn = "out-city-inhabitants-lxml2.csv"
outFile = open(ofn, 'wt', encoding='utf8')
outFile.write("city;inhabitants\n")


def cleanUp(msg):
    print(msg)
    # sendEmail(msg, msg, "yourname@zew.de")
    outFile.close()



pool = urllib3.PoolManager()


for index1,city in enumerate(cities):
    try:
        if index1 > 1: # for testing
            break
```

```
        # lxml.de/tutorial.html#using-xpath-to-find-text
        # urllib3.readthedocs.io/en/latest/user-guide.html
        urlMain = "https://de.wikipedia.org/wiki/%s" % city
        print(("%5d loading url ... %s - %s" % (index1,urlMain,city)))
        resp = pool.request("GET",urlMain)
        print(("\tStatus code is %s" % (resp.status)))

        #r = str(resp.data)
        r = resp.data.decode('utf-8')
        if "noticed some unusual activity coming from your computer network" in r:
            print("CAPTCHA")
            break
        #pos = resp.text.find('<body')
        #print("\t response: ",resp.text[pos:pos+100])

        root = etree.HTML(r)

        expr = "//td"
        # constrain by a css class; where class my contain other values
        expr = "//table[contains(@class,'wikitable')]//td"

        # Either td itself contains the text - or one of the descendents contains the text we search for
        expr = u"//td[contains(text(),'Bevölkerung') or contains(text(),'Einwohner')]"
        expr = u"//td//*[contains(text(),'Bevölkerung') or contains(text(),'Einwohner')]"

        # normalize-space() contains the text of the element and of the descendents
        expr = u"//td[contains(text(),'Einwohner') or descendant::*[contains(text(),'Einwohner')]   or
contains(text(),'Bevölkerung')  or  descendant::*[contains(text(),'Bevölkerung')]]"
        expr = u"//td[contains(normalize-space(),'Einwohner') or contains(normalize-space(),'Bevölkerung
')]"



        # www.infopathdev.com/blogs/greg/archive/2005/06/13/Using-XPath_2700_s-_2700_preceding_2D00_sibl
ing_2700_-Axis-Correctly.aspx
        expr = expr + "/following-sibling::td"
```

```
print(expr)
nodes = root.xpath(expr)


for idx2, nd in enumerate(nodes):
    #text = nd.text

    text = ""

    if True:
        for t in nd.itertext(): # recursive
            text = text + " " + t
    else:
        text = nd.xpath("normalize-space()") # recursive - removing all empty text nodes

    text = text.replace("\n"," ")
    text = text.strip()
    if len(text) < 1:
        continue
    if  True or "Einwohner" in text or "Bevölkerung" in text:
        print(u"node no %3d - |%s|" % (idx2, text[:40]))
    #print(u"node no %3d - |%s|" % (idx2, text[:40]))


outRow = "%s;%s\n" % (city,"no results")
outFile.write(outRow)


slp = float(5+randint(0,40))/500
print(("Sleeping %f" % slp))
time.sleep(slp)
```

```
    except Exception as e:
        print("exception in loop body: %s" % str(e))
        print(traceback.format_exc())
        # sendEmail("exception", traceback.format_exc(), "yourname@zew.de")
        continue
```

```
cleanUp("regular finish")
```

Crawling using a "real browser"

- Websites often contain little `JavaScript` programs
  which load additional contents.
  For instance the "Next 10" results of google search are retrieved and displayed
  by a `JavaScript` program, which triggered.

- => Merely downloading websites with urllib3 is insufficient.

- We want to open the websites in a 'real browser'
  and then access its contents with xpath queries.

- We use the `selenium module` for this.

- This cannot be done in our jupyter browser environment. You need a local Python installation.
  You also need the `chrome browser`.
  As of 2017, Firefox is not working.
  You need a `bridge program` to the `chrome` webbrowser.
  Google Chrome is best to automate
  (https://sites.google.com/a/chromium.org/chromedriver/downloads).
  Copy the most recent `ChromeDriver` into your `PATH`.
  For instance to `c:\windows`

- Python must be run as admin, to use chromedriver.

- We can again use our `xpath` knowledge.

`selenium` is a good tool, but introduces more issues.

- Memory usage grows. You might want to close
  and reopen the browser programmatically, to free memory.

- You might want to use a revolving `VPN access node`,
  so that not all requests are done by the same IP address.

Automated google search

Don't do this without VPN. Otherwise all ZEW colleagues get punished with __Captchas__.

- Google search results are loaded from inside the results page.
- Google search results therefore require an automated browser.
- Search results are saved to a files.

```
# coding=utf-8
# This Python file uses the following encoding: utf-8
from selenium import webdriver
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.common.keys import Keys
import time
import traceback
from send_email import sendEmail

try:
    browser = webdriver.Chrome()

    inpBanks = open('inp-banks.csv', 'rt', encoding='utf8')
    outFile = open('out-banks.csv', 'wt', encoding='utf8')

    print("loading url into browser...")
    browser.get('https://www.google.de/')
    time.sleep(2)

    # r.encoding = 'ISO-8859-1'
    if "noticed some unusual activity coming from your computer network" in browser.page_source:
        print("CAPTCHA encountered")
        sendEmail("selenium failure: captcha", "captcha", "buchmann@zew.de")
        quit()

    for iter, row in enumerate(inpBanks):
        bank = row.split(";")
```

```
print(', '.join(bank))
try:

    if iter > 1:  # for testing
        break

    # We might switch our IP address every x requests...
    iter += 1
    if iter % 10 == 0:
        # from openvpn import newIp
        # newIp()
        pass

    # http://selenium-python.readthedocs.io/locating-elements.html
    # chrome F12 - or right mouse key => "Investigate element"/"Element untersuchen"
    suchfeld = browser.find_element_by_name("q")
    suchfeld.clear()
    print("fld 1 name is %s" % suchfeld.get_attribute('name'))

    # each bank - search only in appropriate language
    u = "%s filetype:pdf" % (bank[1])  # composing the search term
    # uu = u.decode('utf8')
    suchfeld.clear()
    suchfeld.send_keys(u)
    time.sleep(1)
    suchfeld.send_keys('\n')
    time.sleep(4)
    # fld2 = browser.find_element_by_name("btnK");
    # fld2.click()

    results = browser.find_elements_by_css_selector("h3 > a")
    for i, fld in enumerate(results):
        # print fld.get_attribute('innerHTML')
        url = fld.get_attribute('href')
```

```
                printUrl = url
                if len(url) > 110:
                    printUrl = url[:110]
                print("  url %d is %s" % (i, printUrl))
                outRow = ['col1', bank[1], "%s" % i]
                outRow.extend([url])
                # outRow.pop() # remove last element

                outLine = ";".join(outRow) + "\n"
                outFile.write(outLine)
                # time.sleep(1)

            time.sleep(2)
            # next input

        except Exception as e:
            print("\nException in inner loop:\n")
            print(traceback.format_exc())

    print("loop end")


except Exception as e:
    print(traceback.format_exc())
    excToStr = "\nException was\n\t '%s' \n" % (str(e))
    excToStr += traceback.format_exc()
    sendEmail("google selenium failure", excToStr, "you@zew.de")


print("regular finish")
browser.quit()
inpBanks.close()
outFile.close()
# Find an element merely by its text content
```

```
def nodesByText(browser, needle):

    expr = "//*[contains(text(),'%s')]" % needle

    listOfNodes = browser.find_elements_by_xpath(expr)

    for i, nd in enumerate(listOfNodes):

        print("fld %d name is %10s  %10s" % (i, nd.get_attribute('name'), nd.get_attribute('id')))

        nd.click() # Just for fun
```

In the above example, we *virtually* click on the link to load the next page.

An immensely valuable capability of selenium browser automation
is that we can also influence the programming of the webpage directly.

We may encounter a HTML link

```
<a href='#'  onclick='jumpToPage(curPage+1)' >Next Page</a>
```

We can directly call the `jumpToPage` function via the browser-variable's `execute_script` method:

```
# Execute javascript on browser page

# For example to load the next page of results

def execJavaScript(browser):

    script = " jumpToPage('pageNumber','%d');" % 11

    print(script)

    browser.execute_script(script)

    time.sleep(24)

    print("we should be on page %d" % 11)
```

We could even extend this.

We could inject our own piece of JavaScript programming into the HTML page,
and then execute it.

Crawling tenure data of German 'landtag' politicians

This example uses the same techniques as before.
Just for a slightly different website.

```
# coding=utf-8

# This Python file uses the following encoding: utf-8

from selenium import webdriver

from selenium.common.exceptions import TimeoutException

from selenium.webdriver.common.keys import Keys

import time
```

```
import traceback
from send_email import sendEmail

browser = webdriver.Chrome()

inpLandtage = open('inp-landtag.csv', 'rt', encoding='utf8')
outFile = open('out-landtage.csv', 'wt', encoding='utf8')



try:
    for iter, row in enumerate(inpLandtage):
        landtag = row.split(";")
        print(', '.join(landtag))
        try:
            print("loading url into browser... %s" % landtag[1])
            browser.get(landtag[1])
            time.sleep(2)

            # r.encoding = 'ISO-8859-1'
            if "noticed some unusual activity coming from your computer network" in browser.page_source:
                print("CAPTCHA encountered")
                sendEmail("selenium failure: captcha",
                          "captcha", "buchmann@zew.de")
                quit()

            # Search for single element
            direkt = browser.find_element_by_xpath(
                "//table[@class='ohne-rand m-u-0']//td//a").get_attribute('href')
            print(direkt)

            # Search for a multitude of elements; notice the plural 'elements'
            #  We could select an attribute directly:  "...//a/@href", but the method needs a full eleme
nt as return value
            results = browser.find_elements_by_xpath(
```

```
                "//table[@class='ohne-rand m-u-0']//td//a")
            for i, fld in enumerate(results):
                # do something with each htrg
                print("href is %s" %  fld.get_attribute('href'))

            results = browser.find_elements_by_xpath(
                "//table[@class='ohne-rand m-u-0']//tr[3]//td[1]")
            for i, fld in enumerate(results):
                print("inner html ist:    %s" %
                      str(fld.get_attribute('innerHTML')))
                print("text inhalt ist:   %s" % str(fld.text))
                outRow = [landtag[0], str(fld.text)]
                outLine = ";".join(outRow) + "\n"
                outFile.write(outLine)

            # Other search methods are also availabe:
            results = browser.find_elements_by_css_selector("h3 > a")

        except Exception as e:
            print("\nException in inner loop:\n")
            print(traceback.format_exc())

    print("loop end")


except Exception as e:
    print(traceback.format_exc())
    excToStr = "\nException was\n\t '%s' \n" % (str(e))
    excToStr += traceback.format_exc()
    sendEmail("landtage selenium failure", excToStr, "you@zew.de")


print("regular finish")
browser.quit()
```

```
inpLandtage.close()

outFile.close()
```

Adaptive variation of search criteria

- This example is a case study for a defended website.
  We use need some programming effort to crawl the data.

- We want to investigate the financing of the companies listed at angel.co (https://angel.co/)
  angel.co conceals its company database (https://angel.co/companies).

- We only have criteria with low selectivity.
  `market=education` yields over 15.000 results.
  But only 400 results are displayed.

- It seems impossible to crawl all companies.

- We partition search by using all possible letters for company names:
  "abi", "abl", "abo", "acc"

- Each of these 'triplets' yields a fraction of companies.
  Some triplets ('qzj') yield no results.
  Other triplet yield two or three tousand results.

- We could try *all* combinations of the remaining search categories:
  `market: [E-Commerce, Education, Enterprise Software, ... ]`
  combined with all kinds of
  `stages: [Seed, Series A, Series B, ... ]`
  But that would produce thousands of combinations.

- Instead we add further search restrictions dynamically as long as there are more than 400 results.

```
# This Python file uses the following encoding: utf-8

# coding=utf-8

from selenium import webdriver

from selenium.common.exceptions import TimeoutException

from selenium.webdriver.common.keys import Keys

import time




from send_email import sendEmail




import sys
```

```
if sys.stdout.encoding != "utf-8":
    print("SET PYTHONIOENCODING=utf-8")
    print("(Current is %s)" % sys.stdout.encoding)
    print("Then retry")
    quit()


outFile = None
browser = None

def cleanUpWrtr():
    if outFile is not None:
        outFile.close()

def openWrtr(suffix):
    ofn = "out_angel_%s.csv" % (suffix)
    fl = open(ofn, 'wt', encoding='utf8')
    return fl

def openBrowser():
    try:
        browser.quit()
    except Exception as e:
        pass
    br = webdriver.Chrome()
    return br


def numResults(browser):
    counters = browser.find_elements_by_css_selector(".count");
    if len(counters) == 0:
        print("no counters")
        quit()
    strResults = counters[3].get_attribute('innerHTML')
```

```
    strResults = strResults.strip()
    arrResults = strResults.split(" ")
    strResults = arrResults[0]
    strResults = strResults.replace(",","")
    numResults = int(float(strResults))
    print("\t\t\tnumResults is %4d - (%s)" % (numResults,strResults))
    return numResults


triplets = []

# Create all possible triplets of letters
for x1 in range(97,123):
    for x2 in range(97,123):
        for x3 in range(97,123):
            xx = str(chr(x1)) + str(chr(x2)) + str(chr(x3))
            if xx <= "xwg": # up to a certain combination
                continue
            triplets.extend([xx])

# In another script, we dropped all triplets, which yielded not results; i.e. 'zqk' or 'jkj'
# These are 'fertile' triplet
triplets1 = ["abi", "abl", "abo", "acc", "ach", "acq", "acr", "act", "add", "adm", "adv", "aff", "aft",
"aga", "age", "agg", "agr", "aim", "air", "ale", "alr", "amo", "ana", "and", "ang", "ann", "ano", "ans",
 "any", "api", "app", "arc", "are", "aro", "art", "asi", "ask", "asp", "ass", "aud", "aus", "aut", "ava
", "ave", "awa", "bac", "ban", "bar", "bas", "bea", "bec", "bee", "bef", "beg", "beh", "bei", "bel", "be
n", "ber", "bes", "bet", "bev", "big", "bil", "bit", "blo", "boa", "boo", "bor", "bos", "bot", "bou", "b
ox", "bra", "bre", "bri", "bro", "bud", "bui", "bus", "but", "cal", "cam", "can", "cap", "car", "cas", "
cat", "cel", "cen", "cer", "cha", "che", "chi", "cho", "cit", "cla", "cle", "cli", "clo", "clu", "coa",
"cod", "col", "com", "con", "coo", "cor", "cos", "cou", "cov", "cra", "cre", "cri", "crm", "cro", "cul",
 "cur", "cus", "cut", "dai", "das", "dat", "day", "dea", "dec", "ded", "dee", "def", "del", "dem", "den
", "dep", "des", "det", "dev", "dia", "die", "dif", "dig", "dir", "dis", "div", "doc", "doe", "doi", "do
l", "dom", "don", "dow", "dra", "dre", "dri", "dro", "dur", "dyn", "eac", "ear", "eas", "eco", "edi", "e
du", "eff", "ele", "eli", "ema", "emb", "eme", "emp", "ena", "enc", "end", "ene", "eng", "enh", "enj", "
ens", "ent", "env", "est", "etc", "eur", "eve", "exa", "exc", "exe", "exi", "exp", "ext", "fac", "fai",
"fam", "fan"]
triplets2 = ["far", "fas", "fav", "fea", "fee", "few", "fie", "fil", "fir", "fit", "fla", "fle", "fli",
"flo", "foc", "fol", "foo", "for", "fou", "fra", "fre", "fri", "fro", "ful", "fun", "fur", "fut", "gai",
```

```
 "gam", "gat", "gen", "geo", "get", "gif", "giv", "glo", "goa", "goi", "goo", "gov", "gra", "gre", "gro
", "gui", "han", "hap", "har", "has", "hav", "hea", "hel", "her", "hig", "hir", "his", "hol", "hom", "ho
n", "hos", "hot", "hou", "how", "hug", "hum", "hun", "ide", "ima", "imm", "imp", "inc", "ind", "inf", "i
ni", "inn", "int", "inv", "ios", "iph", "iss", "ite", "its", "job", "joi", "jou", "jus", "kee", "key", "
kid", "kin", "kno", "lab", "lan", "lar", "las", "lat", "lau", "lea", "leg", "les", "let", "lev", "lic",
"lif", "lig", "lik", "lim", "lin", "lis", "lit", "liv", "llc", "loc", "log", "lon", "loo", "los", "lot",
 "lov", "low", "loy", "ltd", "mac", "mad", "mag", "mai", "maj", "mak", "man", "map", "mar", "mas", "mat
", "max", "may", "mea", "med", "mee", "mem", "men", "mer", "mes", "met", "mic", "mid", "mil", "min", "mi
s", "mob", "mod", "mom", "mon", "mor", "mos", "mot", "mou", "mov", "muc", "mul", "mus", "nam", "nat", "n
ea", "nee", "net", "nev", "new", "nex", "nic", "nig", "nor", "not", "now", "obj", "off", "oft", "old", "
onc", "one", "onl", "ope", "opp", "opt", "ord", "org", "ori", "oth", "our", "out", "ove", "own", "pac",
"pag", "pai"]

triplets3 = ["pal", "pap", "par", "pas", "pat", "pay", "pee", "pen", "peo", "per", "phi", "pho", "phy",
"pic", "pin", "pla", "ple", "poi", "pol", "pop", "por", "pos", "pot", "pow", "pra", "pre", "pri", "pro",
 "pub", "pur", "pus", "put", "qua", "qui", "rai", "ran", "rat", "ric", "rig", "ris", "rob", "roo", "rou
", "run", "saa", "saf", "sal", "sam", "san", "sat", "sav", "sca", "sch", "sci", "sco", "scr", "sea", "se
c", "see", "sel", "sen", "ser", "set", "sev", "sha", "she", "shi", "sho", "sid", "sig", "sil", "sim", "s
in", "sit", "siz", "ski", "sma", "soc", "sof", "sol", "som", "soo", "sou", "spa", "spe", "spi", "spo", "
spr", "sta", "ste", "sti", "sto", "str", "stu", "sty", "sub", "suc", "sui", "sup", "sur", "sus", "swi",
"sxh", "syn", "sys", "tab", "tai", "tak", "tal", "tap", "tar", "tas", "tea", "tec", "tel", "tem", "ten",
 "ter", "tes", "tex", "tha", "the", "thi", "tho", "thr", "tic", "tim", "tod", "tog", "too", "top", "tor
", "tou", "tow", "tra", "tri", "tru", "try", "tur", "twi", "two", "typ", "ult", "und", "uni", "unl", "up
d", "upl", "usa", "use", "usi", "uti", "val", "var", "ven", "ver", "via", "vid", "vie", "vir", "vis", "v
oi", "vol", "wal", "wan", "war", "was", "wat", "way", "wea", "web", "wee", "wel", "wer", "wha", "whe", "
whi", "who", "why", "wid", "wil", "win", "wis", "wit", "wom", "wor", "wou", "wri", "www", "yea", "yet",
"yor", "you"]


triplets = triplets1 + triplets2 + triplets3

triplets = ["pal", "gul", "sat"]


stages  = ["Seed", "Series A", "Series B", "Series C", "Acquired" ]

markets = ["E-Commerce", "Education", "Enterprise Software", "Games", "Health Care" , "Mobile" ]



print("Created %d triplets" % len(triplets1))


maxCombination = ""


try:

    numSearchLoops = 0

    for triplet in triplets:
```

```
        for stage in stages:
            for market in markets:
                try:
                    curCombination = "%s--%s--%s" % (triplet,stage,market)
                    if curCombination < maxCombination:
                        print("   skipping %20s due to %20s" %(curCombination,maxCombination))
                        continue

                    numSearchLoops += 1
                    # if numSearchLoops % 300 == 1:
                    if numSearchLoops % 100 == 1:
                        cleanUpWrtr()
                        outFile = openWrtr(triplet)
                        browser = openBrowser()

                    urlMain = 'https://angel.co/companies?company_types[]=Startup'
                    print("%3d loading url into browser... %s - %s - %s - %s" % (numSearchLoops,urlMain,
triplet,stage,market))
                    browser.get(urlMain)
                    time.sleep(0.2)

                    searchFld = browser.find_element_by_css_selector(".search-box");
                    searchFld.click()
                    time.sleep(0.1)
                    inpFld = browser.find_element_by_css_selector(".keyword-input");
                    time.sleep(0.2)
                    inpFld.send_keys(triplet)
                    inpFld.send_keys('\n')
                    time.sleep(2.1)


                    if numResults(browser) < 400:
                        maxCombination = "%s--zzzzzz" % (triplet)
                    else:
```

```
                    filters = browser.find_elements_by_css_selector(".dropdown-filter");
                    if len(filters) == 0:
                        print("no dropdown filters")
                        quit()
                    for i, fld in enumerate(filters):
                        filterName = fld.get_attribute('data-menu')
                        if filterName=="stages":
                            fld.click()
                            time.sleep(0.2)
                            filterRows = browser.find_elements_by_css_selector(".dropdown-filter di
v.filter-row");
                            for j, fr in enumerate(filterRows):
                                fRow = fr.get_attribute('data-value')
                                if fRow==stage:
                                    fr.click()
                                    break
                            time.sleep(1.2)
                            break




                    if numResults(browser) < 400:
                        maxCombination = "%s--%s--zzzzzz" % (triplet,stage)
                    else:
                        for i, fld in enumerate(filters):
                            filterName = fld.get_attribute('data-menu')
                            if filterName=="markets":
                                fld.click()
                                time.sleep(0.2)
                                filterRows = browser.find_elements_by_css_selector(".dropdown-filter
 div.filter-row");
                                for j, fr in enumerate(filterRows):
                                    fRow = fr.get_attribute('data-value')
                                    if fRow==market:
```

```
                        fr.click()
                        break
                time.sleep(1.2)
                break

numResults(browser)

time.sleep(10.2)

# continue


print("\tsearch entries complete for '%s' '%s' '%s'" % (triplet,stage,market))

# results = browser.find_elements_by_css_selector("h3 > a")
results = browser.find_elements_by_css_selector(".startup-link")
if len(results) == 0:
    outRow = [triplet,stage,market,"0","no results"]
    outFile.write(";".join(outRow) + "\n")
    continue

for x in range(0,19):
    print("\textending more btn %d" % x)
    if len(results)>x*40-1:
        try:
            moreButton = browser.find_element_by_css_selector(".more");
            moreButton.click()
            time.sleep(1.7)
            results = browser.find_elements_by_css_selector(".startup-link")
        except Exception as e:
            print("\tno more more button")
            break
```

```
                    print("\trange extension complete for '%s' - => %d results" % (triplet,len(result
s)))

                    for i, fld in enumerate(results):
                        url = fld.get_attribute('href')
                        print("\t\turl %d is %s" % (i,url))
                        outRow = [triplet,stage,market,str(i)]
                        outRow.extend([url])
                        # outRow.pop() # remove last element
                        outFile.write(";".join(outRow) + "\n")


                except Exception as e:
                    outRow = [triplet,stage,market,"0","exception in stage loop %s" % str(e).replace('\n
', '<br>')]
                    outFile.write(";".join(outRow) + "\n")
                    print("Exception in stage loop: %s" % str(e))
                    continue

                time.sleep(0.4)
                # next input


    try:
        browser.quit()
    except Exception as e:
        pass
    cleanUpWrtr()


except Exception as e:
    print("global exception -- %s" % str(e))
    excToStr =  "\nException was\n\t '%s' \n" % (str(e))
    sendEmail("selenium angel.co fail", excToStr,"you@zew.de")
```

```
try:
    browser.quit()
except Exception as e:
    pass
cleanUpWrtr()
```

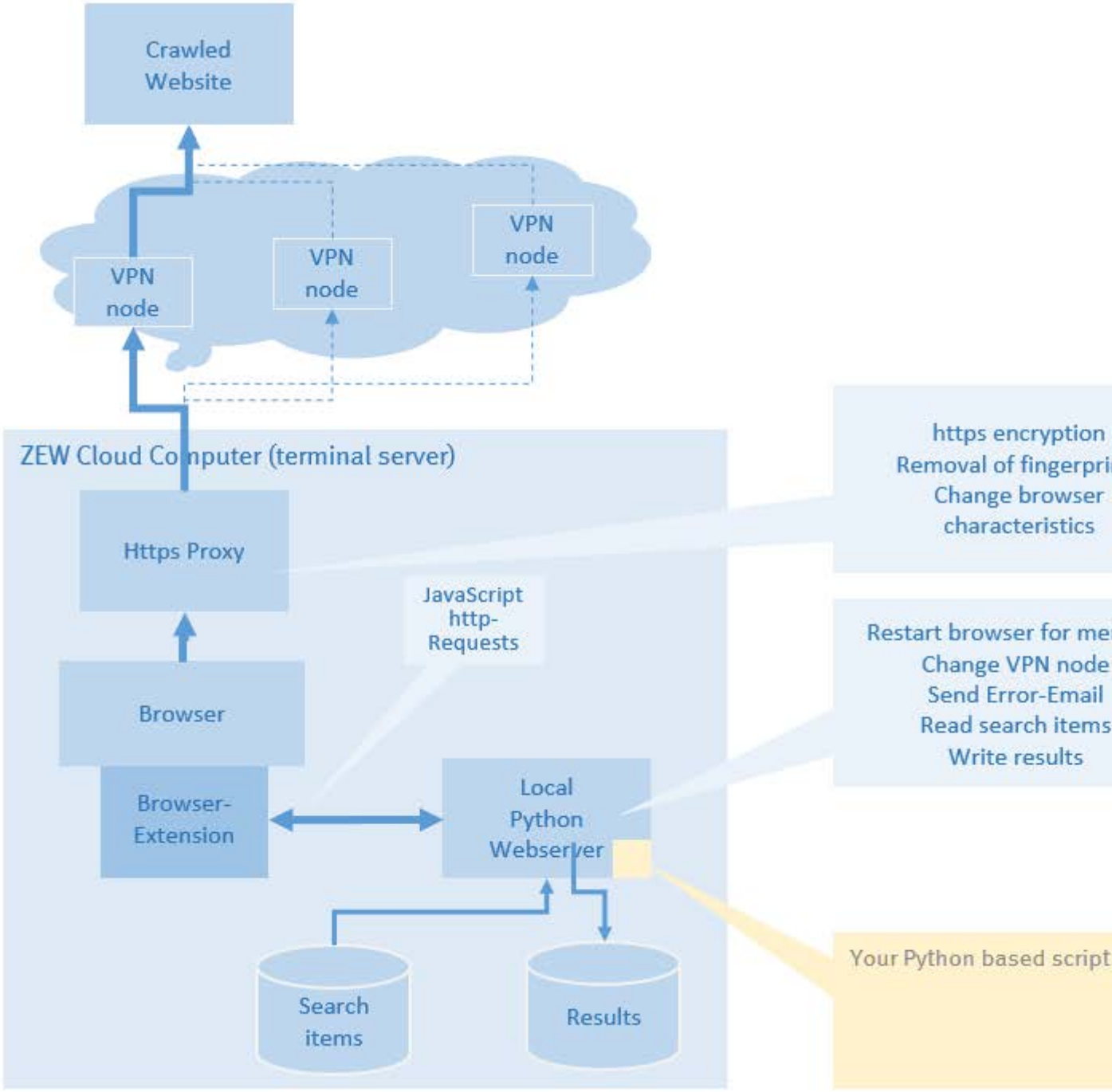
If even selenium fails

Some websites detect selenium driven browsers,
and swiftly block them.

Example: The company detail pages of angel.co

These data sets might me the most highly valuable ones, for scientific use.

If we want to get them, we have to resort to even more sophisticated ways
of browser automation, via JavaScript.
These crawling efforts require one to three weeks setup effort.



The browser extension

The browser extension serves as *invisible* automation script.
It was not yet detected by the website's defences.

The browser extension is put into the `Tampermonkey` chrome extension.

Since browser extensions have no read or write access to the file system,
we read and write to a accompanying webserver.

Since browser extensions must be programmed using JavaScript, this component is in JavaScript.

The major caveat is a loop of *nested asynchronous* function calls.

```
f1() ... f(4)
```

```
// ==UserScript==
// @name         angel.co-information-request
// @description  include jQuery and make sure window.$ is the content page's jQuery version, and this.$ is our jQuery version.
// @description  http://stackoverflow.com/questions/28264871/require-jquery-to-a-safe-variable-in-tampermonkey-script-and-console
// @namespace    http://your.homepage/
// @version      0.124
// @author       iche
// @downloadURL  https://raw.githubusercontent.com/pbberlin/dom/master/ui/tamper-monkey-citeproof-angel.js
// @updateURL    https://raw.githubusercontent.com/pbberlin/dom/master/ui/tamper-monkey-citeproof-angel.js
// @match        *://*/*
// @match        *://localhost:*/*
// @require      https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js
// @require      https://code.jquery.com/ui/1.12.1/jquery-ui.js
// @grant        none
// @noframes
// @run-at      document-end
// ==/UserScript==



// fallback http://encosia.com/3-reasons-why-you-should-let-google-host-jquery-for-you/
if (typeof jQuery === 'undefined') {
    console.log("CDN blocked by Iran or China?");
    document.write(unescape('%3Cscript%20src%3D%22/path/to/your/scripts/jquery-2.1.4.min.js%22%3E%3C/script%3E'));
}


(function ($, undefined) {
    $(function () {
```

```
        console.log("isolated jQuery start");


        //Returns a random integer between min (inclusive) and max (inclusive)
        function getRandomInt(min, max) {
            return Math.floor(Math.random() * (max - min + 1)) + min;
        }


        function AddCssAndHtml(){
            if ($('#css-hover-popup').length <= 0) {
                var s =  '';
                s += '<style type="text/css"  id="css-hover-popup" >';
                s += '.ui-draggable-handle {';
                s += '    -ms-touch-action: none;';
                s += '    touch-action: none;';
                s += '}';
                s += '#id32168 { ';
                s += '    cursor: pointer;';
                s += '    right: 10px; top: 10px; width: 150px; min-height: 40px;';
                s += '    position: absolute; z-index: 2100;';
                s += '    background-color: #aca;';
                s += '    border: 1px solid #464;';
                s += '    border-radius: 8px;';
                s += '    font-family: Monospace;';
                s += '    padding: 16px 14px;';
                s += '    font-size: 13px;';
                s += '    text-align: center;';
                s += '}';
                s += '</style>';
                $(s).appendTo('head');


                var popupUpScaffold = "<div id='bracket32168' style='position: relative;'><div id='id32168' >Click to save as Quotation. <span style='font-size:10px;'><br><br>or drag to move<span></div></div>";
                $('body').prepend(popupUpScaffold);  //next after <body;  most counter intuitive: stackoverflow.com/questions/5073016
```

```
            console.log("tamper monkey:     added button");
        }
    }

    var reqData  = {urlx: window.location.href};
    var respData = {};
    var empty = {};

    function runLoop(){


        f1 = function(){

            reqData.urlx = window.location.href;
            reqData.param_id = -10;
            reqData.val_id   = -10;
            console.log("tamper monkey: window location is ",window.location.href);
            // http://stackoverflow.com/questions/2120060
            reqData.data = document.body.parentNode.innerHTML;
            var doctypeInformation = document.body.parentNode.previousSibling;

            f2 = function() {

                f3 = function() {

                    f4 = function(respD){
                        console.log("\t\t respD",respD);
                        setTimeout(function(){
                            // remove all cookies
                            document.cookie.split(";").forEach(
                                function(c) {
                                    document.cookie = c.replace(/^ +/, "").replace(/=.*/, "=;expires
=" + new Date().toUTCString() + ";path=/");
                                }
```

```
                        );
                        window.location.href = 'https://angel.co/' + respD.comp;
                    }, respD.milliseconds);
                };

                ajaxReq('https://localhost:8082/next-comp','GET',empty, f4 );


            };

            console.log("\t\tuploaded ", window.location.href, " successfully." );
            var href = window.location.href;
            var parts = href.split("/");
            var last_element = parts[parts.length - 1];
            var arrayLength = parts.length;
            for (var i = 0; i < arrayLength; i++) {
                var el = parts[i];
                var cond1 = el.startsWith("captcha");
                var cond2 = el.startsWith("RL_POST_CAPTCHA");
                var cond3 = el.includes("RL_POST_CAPTCHA");
                if (cond1 || cond2 || cond3) {
                    console.log("\t\tCATPCHA");
                    ajaxReq('https://localhost:8082/email/captcha-encountered','GET',empty, func
tion(){} );
                    return;
                }
            }
            ajaxReq('https://localhost:8082/save-comp?comp=' + last_element,'GET',empty , f3 );
        };

        ajaxReq('https://citeproof.appspot.com\/upload-receiver','POST',reqData, f2 );
    };

    f1();
```

```
}


function gatherDocData(){
    reqData.urlx = window.location.href;
    reqData.param_id = -10;
    reqData.val_id = -10;
    console.log("tamper monkey: window location is ",window.location.href);
    // stackoverflow.com/questions/2120060
    reqData.data = document.body.parentNode.innerHTML;
    var doctypeInformation = document.body.parentNode.previousSibling;
    ajaxReq('https://citeproof.appspot.com\/upload-receiver','POST',reqData, function(){} );
}

function ajaxReq(argUrl,argMethod,argData,succFct) {

    var fctFail = function(respD){
        console.log("\t\t retry after 1820 secs");
        setTimeout(function(){
            fctPayload();
        }, 1820*1000);
    };
    var fctPayload = function(){
        ajaxReqInner(argUrl,argMethod,argData,succFct,fctFail);
    };
    fctPayload();

}

function ajaxReqInner(argUrl,argMethod,argData,succFct,failFct) {
```

```
    // document.getElementById('url_name').value= argData.loc;

    var fctSucc = function(data) {
        console.log("fctSucc",data);
        succFct(data);
    };

    var fctDone = function() {
        console.log("fctDone");
    };

    var fctError = function(XMLHttpRequest, textStatus, errorThrown) {
        console.log("fctError",XMLHttpRequest, textStatus, errorThrown);
    };

    var fctFail = function(data) {
        console.log("fctFail",data);
        failFct(data);
    };

    console.log("\t\tbefore "+argMethod+"-ing to "+argUrl+";");
    console.log("\t\tparams are",argData);

    $.ajax({
        url:      argUrl,
        type:     argMethod,
        data:     argData,
        success:  fctSucc,
        timeout:  96000,
        error:    fctError,
    }).done(fctDone).fail(fctFail);

} // post it
```

```
        $( document ).ready(function() {
            console.log("tamper monkey: document ready start; window name is",window.name);
            if (window.name==="browser_bridge_window" || true) {
                AddCssAndHtml();
                $( function() {
                    $('#id32168').draggable({ cursor:"move" });
                    //$('#id32168').onclick = gatherDocData;
                    // $('#id32168').click(gatherDocData);
                    $('#id32168').click(runLoop);
                });
                console.log("tamper monkey:     added clickhandler");

                $( function() {
                    runLoop();
                });
            }
            console.log( "tamper monkey: document ready end" );
        });


        console.log("isolated jQuery end");
    });
})(window.jQuery.noConflict(true));
```

The webserver

The web server is written in Python.

It gives input to the browser extension.

It stores the results of the crawling, submitted from the browser extension.

It revolves the VPN access.

It restarts the browser from time to time, to avert memory hogging.

```
#!/usr/bin/python
```

```
# the accompanying webserver
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
from random import randint
from urlparse import urlparse
import time
import os
import ssl
from send_email import sendEmail
from openvpn import newIp
from openvpn import execCmd

workDir = os.path.dirname(os.path.realpath(__file__))

def restartChrome(url="https://angel.co/annelutfen"):
    chromePath = 'C:\Program Files (x86)\Google\Chrome\Application\chrome.exe'
    # taskkill /F /IM chrome.exe
    execCmd(["taskkill", "/F", "/IM", "chrome.exe"],True)
    from subprocess import Popen
    p = Popen([chromePath, url]) # something long running
    # ... do other stuff while subprocess is running
    # p.terminate()
    print "chrome restarted done"



newIp()
time.sleep(5.2)
restartChrome()

PORT_NUMBER = 8082

fileSaved = "03-detail-saved.csv"
fileWork  = "02-inp-detail-all.csv"
# fileWork = "02-inp-detail-test.csv"
```

```
work  = []
saved = {}

# get all saved into dict
with open(fileSaved, "rt") as fin:
    for line in fin:
        cols = line.strip().split(";")
        key = cols[0]
        val = cols[1]
        saved[key] = val



with open(fileWork, "rt") as fin:
    for line in fin:
        cols = line.strip().split(";")
        key = str(cols[0])
        if key in saved:
            pass
        else:
            # if saved[key] is None or saved[key] == "":
            work.extend([cols[0]])



cntr = 0
for w in work:
    cntr+=1
    if cntr>30:
        break
    print("%s, " % w)



#This class will handles any incoming request from
#the browser
class myHandler(BaseHTTPRequestHandler):
```

```
    reqCounter = 0


    #Handler for the GET requests
    def do_GET(self):
        self.send_response(200)
        if self.path=="/next-comp":
            self.send_header('Content-type','application/json')
             self.send_header('Access-Control-Allow-Origin', '*')
            self.end_headers()
            myHandler.reqCounter += 1
            print    "reqCounter = %d" % myHandler.reqCounter
            while True:
                idx =  randint(0,len(work)-1) # last line is empty
                print "random idx is %d of %d - %s" % (idx,len(work),work[idx])
                if work[idx] in saved:
                    print "%s already saved" % (work[idx])
                    continue
                else:
                    if myHandler.reqCounter % 45 == 0:
                        milliseconds = 11*1000
                        self.wfile.write("{\"comp\":\"%s\",\"milliseconds\":%d}" % (work[idx],millisecon
ds))

                        try:
                            self.finish()  # contains self.wfile.flush()
                        except Exception as e:
                            print "self.finish() FAILED %s" % str(e)
                        try:
                            self.connection.close()
                        except Exception as e:
                            print "self.connection.close() FAILED %s" % str(e)
                        newIp()
                        time.sleep(5.2)
                        newUrl = "https://angel.co/%s" % work[idx]
```

```
                        restartChrome(newUrl)
                        time.sleep(5.2)
                        break
                    elif myHandler.reqCounter % 5 == 0:
                        milliseconds = 61*1000
                        self.wfile.write("{\"comp\":\"%s\",\"milliseconds\":%d}" % (work[idx],millisecon
ds))
                        try:
                            self.finish()  # contains self.wfile.flush()
                        except Exception as e:
                            print "self.finish() FAILED %s" % str(e)
                        try:
                            self.connection.close()
                        except Exception as e:
                            print "self.connection.close() FAILED %s" % str(e)
                        newIp()
                        time.sleep(1.2)
                        break
                    milliseconds = 11*1000
                    self.wfile.write("{\"comp\":\"%s\",\"milliseconds\":%d}" % (work[idx],milliseconds))
                    break
        elif self.path.startswith("/save-comp"):
            self.send_header('Content-type','text/html')
             self.send_header('Access-Control-Allow-Origin', '*')
            self.end_headers()
            try:
                comp = ""
                query = urlparse(self.path).query
                query_components = dict(qc.split("=") for qc in query.split("&"))
                comp = query_components["comp"]
            except Exception as e:
                print "exception parsing %s: %s" % (query,str(e))
            if comp != "":
                with open(fileSaved, "a") as fout:
```

```
                timestamp = int(time.time())
                fout.write(comp + ";" + str(timestamp) + "\n")
                saved[comp] = timestamp
                print "saved comp %s at %s" % (comp,timestamp)
        elif self.path.startswith("/email"):
            sendEmail("selenium: FAIL", self.path,"buchmann@zew.de")
        else:
            pass
            # self.wfile.write("Hello root or other!")
        return


try:
    #Create web server; define handler for incoming request
    server = HTTPServer(('', PORT_NUMBER), myHandler)
    server.socket = ssl.wrap_socket (server.socket, certfile='stunnel.pem', server_side=True)
    #server.socket = ssl.wrap_socket (server.socket, certfile='goproxy.pem', server_side=True)
    print 'Started httpserver on port ' , PORT_NUMBER
    server.serve_forever()

except KeyboardInterrupt:
    print '^C received, shutting down the web server'
    server.socket.close()
```

The proxy server

The proxy server should change the browser's characteristics in a consistent way.

There are 'sets' of values for operating system, browser vendor and language,
which are changed in concert.

The https support is still in research.
Current idea is:
Run the browser in http and use the proxy to shuffle headers and **then** terminate https.

This would also free the Python webserver from needing https.

```
package main


import (
          "fmt"
          "log"
          "net/http"
          "net/url"
          "regexp"
          "strings"
          "time"


          gpx "github.com/elazarl/goproxy"
)


var oss = []string{
          "Mozilla/5.0 (Windows NT 6.1; WOW64)",
          "Mozilla/5.0 (Windows NT 6.1; Win64; x64)",
          "Mozilla/5.0 (Windows NT 10.0; WOW64)",
}


var agents = []string{
          "AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111 Safari/537.36",
          "AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36",
          "AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.1500.52 Safari/537.36 OPR/15.0.1147.100",
          "Gecko/20100101 Firefox/10.0",
}


var combined = []string{}


// www.singular.co.nz/2008/07/finding-preferred-accept-encoding-header-in-csharp/
var acceptEncodings = []string{
          "gzip, deflate",
          "deflate,gzip",
```

```
        "compress;q=0.5, gzip;q=1.0",
        "gzip;q=1.0, identity; q=0.5, *;q=0",
        "gzip;q=.5,deflate",
        "*",
        // "deflate;q=0.5,gzip;q=0.5,identity",
}

// www.singular.co.nz/2008/07/finding-preferred-accept-encoding-header-in-csharp/
var acceptLanguages = []string{
        "en-US, en-gb;q=0.8, en;q=0.",
        "en-US",
        "de-de, de, en;q=0.5, fr;q=0.2",
}

//
//
var lastRequestTime = int64(0)

func getSlot() int {
        t := time.Now().Unix()
        if (t - lastRequestTime) > 22 {
                lastRequestTime = t
                t2 := t / int64(6*60) // six minutes
                return int(t2)
        }
        t2 := lastRequestTime / int64(60*3) // three minutes
        return int(t2)
}

//
var prevSlot = 0

func tick() bool {
        if getSlot() == prevSlot {
```

```
                return false
        }
        prevSlot = getSlot()
        return true
}

func getUA() string {
        l := len(combined)
        idx := getSlot() % l
        return combined[idx]
}

func getAccEnc() string {
        l := len(acceptEncodings)
        idx := getSlot() % l
        return acceptEncodings[idx]
}

func getAccLang() string {
        l := len(acceptLanguages)
        idx := getSlot() % l
        return acceptLanguages[idx]
}

var cntr = 0
var chunk = 10

func init() {
        for _, v1 := range oss {
                for _, v2 := range agents {
                        combined = append(combined, fmt.Sprintf("%v %v", v1, v2))
                        log.Printf("%v", fmt.Sprintf("%v %v", v1, v2))
                }
        }
```

```
}

func printHeader(k string, v []string) {
	for _, v1 := range v {
		if k != "Cookie" {
			log.Printf("\t%-20v %v", k, v1)
		} else {
			cooks := strings.Split(v1, ";")
			for _, v2 := range cooks {
				v2 = strings.TrimSpace(v2)
				v2, _ = url.QueryUnescape(v2)
				if strings.HasPrefix(v2, "ajs_user") || strings.HasPrefix(v2, "ajs
_group") || strings.HasPrefix(v2, "_ga") {
					continue
				}
				if len(v2) > 127 {
					v2 = v2[:127]
				}
				log.Printf("\t  %-20v", v2)
			}
		}
	}
}

func main() {

	prox := gpx.NewProxyHttpServer()
	prox.Logger.SetFlags(log.Lshortfile)
	log.SetFlags(log.Lshortfile)
	prox.Verbose = true
	prox.Verbose = false

	if false {
		re1 := regexp.MustCompile("^[\\-a-z]{0,10}[.]{0,1}(google|github|dropbox).(com|de)$")
```

```
                prox.OnRequest(gpx.Not(gpx.ReqHostMatches(re1))).HandleConnect(gpx.AlwaysMitm)
                // prox.OnRequest(...).HandleConnect(gpx.AlwaysReject)
        }

        if false {
                re1 := regexp.MustCompile("^(www.google.com)|(angel.co)$")
                prox.OnRequest(gpx.ReqHostMatches(re1)).HandleConnect(gpx.AlwaysMitm)
        }

        if true {
                re1 := regexp.MustCompile(".*(angel.co)")
                prox.OnRequest(gpx.ReqHostMatches(re1)).HandleConnect(gpx.AlwaysMitm)
        }

        // re2 := regexp.MustCompile("^.*:8082$")
        // prox.OnRequest(gpx.ReqHostMatches(re2)).HandleConnect(gpx.AlwaysMitm)

        // No idea why this does not work
        // prox.OnRequest(gpx.ReqHostIs("angel.co")).DoFunc(

        re3 := regexp.MustCompile("^(xxx)|(angel.co)$")
        prox.OnRequest(gpx.ReqHostMatches(re3)).DoFunc(
                func(r *http.Request, ctx *gpx.ProxyCtx) (*http.Request, *http.Response) {
                        pth := r.URL.Path
                        if len(pth) > 67 {
                                pth = pth[:67]
                        }
                        prox.Logger.Printf("ANGEL req to %-5v - %v%v", r.URL.Scheme, r.URL.Host, pt
h)

                        isHtml := false
                allHdr:
                        for _, v := range r.Header {
                                for _, v1 := range v {
```

```
                        if strings.Contains(v1, "text/html") {
                                isHtml = true
                                break allHdr
                        }
                }
        }
        if isHtml {

                cntr++
                log.Printf("req counter is %v; slot is %v", cntr, getSlot())
                hdrs := http.Header{} // map[string][]string

                for k, v := range r.Header {
                        if k == "Sss" || k == "X-Csrf-Token" || k == "X-Requested
-With" {
                                continue
                        }
                        // printHeader(k, v)
                        if _, ok := hdrs[k]; !ok {
                                hdrs[k] = []string{}
                        }
                        for _, v1 := range v {
                                hdrs[k] = append(hdrs[k], v1)
                        }
                }
                hdrs["User-Agent"] = []string{getUA()}
                hdrs["Accept"] = []string{"text/html"}
                hdrs["Referer"] = []string{"https://angel.co"}
                hdrs["Accept-Encoding"] = []string{getAccEnc()}
                hdrs["Accept-Language"] = []string{getAccLang()}
                if tick() {
                        hdrs["Cookie"] = []string{"_ga=GA1.2.538972881.1490181682
"}
                        hdrs["Cookie"] = []string{}
```

```
                }
                r.Header = hdrs
                for k, v := range r.Header {
                        printHeader(k, v)
                }

        }

        return r, nil
    })

if false {
    prox.OnRequest().DoFunc(
        func(r *http.Request, ctx *gpx.ProxyCtx) (*http.Request, *http.Response) {
            if r.URL.Scheme == "https" {
                r.URL.Scheme = "http"
                r.URL.Scheme = "https"
            }

            if r.URL.Path == "/favicon.ico" {
                resp := gpx.NewResponse(
                    r,
                    gpx.ContentTypeText,
                    http.StatusForbidden,
                    "No damn icon requests",
                )
                return r, resp
            }

            // r.Header.Set("X-gpx", "yxorPoG-X")
            pth := r.URL.Path
            if len(pth) > 67 {
                pth = pth[:67]
            }
```

```
                                prox.Logger.Printf("NORML req to %-5v - %v%v", r.URL.Scheme, r.UR
L.Host, pth)


                                return r, nil

                        })

        }


        log.Fatalln(http.ListenAndServe(":8080", prox))

}
```

PDF data extraction

- Rearrange the pages of a PDF file. Split/combine pages.
- Extract the contents of a PDF file
- PDF text extraction has few structural or formatting information.
  It is just the text.
  For instance, table cells are hard to discern.

PDF to HTML trick

ZEW employees can use `Adobe Acrobat` (not the mere 'Reader')
to save PDF files as HTML.
Many PDF files yield their structure to the HTML file.
You can then use the crawling methods explained above,
to extract for instance table cell data.

```
'''

    Splitting a large pdf file.

    Three pages are collected an saved as smaller file.


'''


from pyPdf import PdfFileWriter, PdfFileReader

from math import floor



inpPDF = PdfFileReader(open("questionaire.pdf", "rb"))

print("\tPDF file opened. It has %s pages" % inpPDF.numPages)
```

```
for i in range(inpPDF.numPages):
    print("page %s" % i)
    if((i%2)==0):
        outPDF = PdfFileWriter()  # NEW pdf file


        // PDF Text extraction...
        content = inpPDF.getPage(i).extractText().encode("ascii", "ignore") + "\n"
        content = content.replace("\n"," ",-1)


    outPDF.addPage(inpPDF.getPage(i)) # add current page
    if((i%2)==1):
        idxPers = int(floor(i/2)) # search  suitable person
        print "Writing file %d for %s %s (page %d)" % (idxPers, pers1[idxPers][1], pers1[idxPers][2], i
 )
        outFile = open("./out-pages-%s-%s-%s.pdf" % (i+1, i+2), "wb")
        outPDF.write(outFile)
        outFile.close()
```

Processing of statistical data.

We will learn two modules `numpy` and `pandas`.
Both are intertwined.

The numpy module (http://www.numpy.org/) (numeric python)

You install the `numpy` module by entering

```
pip install numpy
```

You enable it for your Python program by typing

```
import numpy as np
```

- The `numpy` module containes *extended* integers, floats types.
  For instance `numpy floats` can be be much bigger.

- `numpy` also contains an extended `list` called `array` (`https://docs.scipy.org/doc/numpy-dev/user/quickstart.html`).
  `array` can be multidimensional.
  And `array` has lots of additional functionality.

- numpy in detail (http://cs231n.github.io/python-numpy-tutorial/)

The pandas module (http://pandas.pydata.org/)

The `pandas` module is an some ways an extension of the numpy module and numpy arrays.

You install the `pandas` module by entering

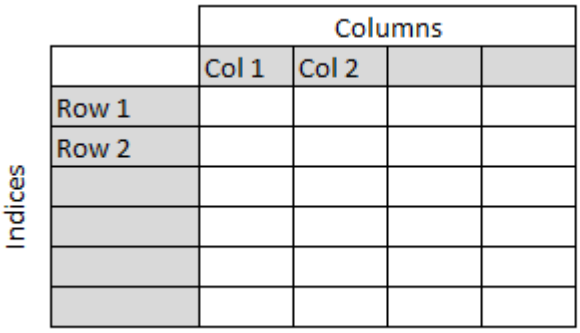
```
pip install pandas
```

You enable it for your Python program by typing

```
import pandas as pd
```

`pandas dataframe`

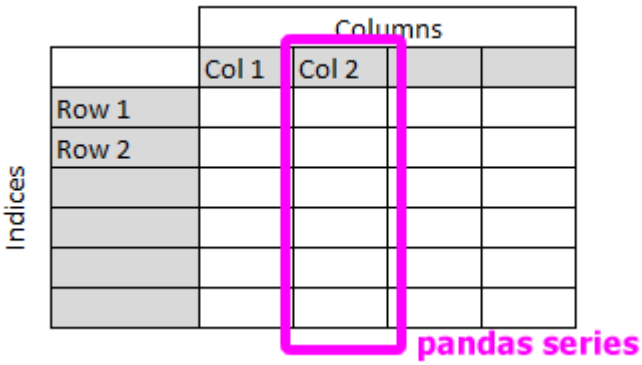
The `pandas dataframe` is the equivalent to the 'datasets' of the familiar Stata statistics package.
"Stata datasets are rectangular arrays" (Source (http://data.princeton.edu/stata/dataManagement.html)).



- `pandas dataframe` builds on the multidimensional array of numpy.
  We will develop our understanding of `pandas dataframe` step by step below.

Pandas series

- A `pandas series` can be seen as one column of a dataframe.
  There is some similarity to a variable in a Stata dataset.



- A `pandas series` is an extension of a `numpy array`
  which in turn is an extension of Python `list`.

Series creation

```
import numpy  as np
import pandas as pd



# create a ndarray
# a = np.array(object, dtype=None, copy=True,   order='K', subok=False, ndmin=0)¶
#                      min type    copy object  unchanged               min dimensions


a = np.array([1,2,3], dtype=float, copy=True) # array is copied and converted to float


print("Type of a is %s" %  str(type(a)) )


print("But it is also still a list: ", isinstance([], list) ) # isinstance also checks inherited types


s1 = pd.Series(a, index=['row1','row2','row3'], name="col1")     # creating a series from a ndarray
s1 = pd.Series({"row1":1.0,"row2":2.0,"row3":3.1,}, name="col1") # we can also create it from a dict; ke
ys become index labels
# name will be the stata var
```

```
s2 = pd.Series(np.random.randn(4), index=['row1','row2','row3','row4'])

s3 = pd.Series(5., index=['row1','row2']) # repeat 5. for all indexes; notice the dot indicating float


print(s1, end="\n\n")
print(s2, end="\n\n")
print(s3, end="\n\n")
```

Series operations

```
print(s1[1:2], end="\n\n")  # subset

# We can apply functions to all elements.
print(np.exp(s1), end="\n\n")  # works as long as all elements are of type float64, otherwise the type o
f series degrades to object

print(2*s1 + s1, end="\n\n")
# We cannot extend it or set single values:
try:
    s1.extend(4.0)
except Exception as e:
    pass

# Setting values by integer index fails
try:
    s1[3]=4.0
except Exception as e:
    pass
s1['row2'] = np.nan
s1['row2'] = 4.0  # We have to set values by index label


print('rw2' in s1, end="\n\n") # Check whether index label exists; unchecked access leads to exception
```

```
# Vector elements are always added/multiplied with similar index label; missing partners yields nan.
print(s1, "\n\n")
print(s1[:-1], "\n\n")

print(s1 + s1[:-1], "\n\n")

##  continue with the pandas docs:
### pandas.pydata.org/pandas-docs/stable/dsintro.html
```

Pandas data frame

Creation from series

```
# Reuse the names of the series
multiSeries = {s1.name: s1, s2.name: s2, s3.name: s3}   # bunch multiple series in a dictionary
df = pd.DataFrame(multiSeries) # conversion syntax – similar to s = str(0.1); type casting
print(df, end="\n\n")

multiSeries = {"col1":s1, "col2":s2[:-1], "col3": s3}
df = pd.DataFrame(multiSeries)
print(df, end="\n\n")

# Why is the difference?
# Direct creation; without index labels
df = pd.DataFrame({'col1' : [1., 2.], 'col2' : [4., 3.]})
display(df)
display(df.index)
# Direct creation; from dict
df = pd.DataFrame({'col1' : {'row1': 1., 'row2': 2.}, 'col2' : {'row2': 2., 'row3': 3.}})
display(df)
print(df.index)

# Re-Index
```

```
# Create date ranges
idx = pd.date_range('1/1/2000', periods=3)


oldIdx = df.index
df.index = idx
display(df)
print(df.index)
df.index = oldIdx
```

Data frame visualization and description

```
display(df) # nicer print
print(df.index, end="\n\n")
df.info() # extended info
print(df.index, end="\n\n")
print("Notice the omission of the NaN cells in computing the mean.")
df.describe()
```

Multi-index - Panel, three-dimensional data, n-dimensional data

If you have more than 2-dimensional data sets...

- Try a MultiIndex DataFrame
- Heed the xarray package

```
# A multi-indexed, nested data frame; example
pd.DataFrame({
    ('a', 'I'):  {('A', 'i'):   1, ('A', 'ii'):  2},
    ('a', 'II'): {('A', 'i'):   3, ('A', 'ii'):  4},
    ('b', 'I'):  {('B', 'ii'):  3, ('A', 'iii'): 4},
    ('b', 'II'): {('B', 'ii'):  3, ('A', 'iii'): 4}
})
```

Row and col selections - slicing and dicing of data frames

```
# by index labels and column names
df1 = pd.DataFrame(df, index=['row1', 'row2'], columns=['col1', 'col2'])
```

```
display(df1)
print(df1.shape)

# slice rows
subset = df[0:1]
display(subset)
print(subset.shape)


subset = df.iloc[0:1,0:2]  # slice rows, slice columns by using integers
display(subset)
print(subset.shape)


subset2 = df.loc['row1']      # slice rows, using 'integer labels'
display(subset2)


subset2 = df.loc['row2':'row3']      # slice rows, using 'integer labels'
display(subset2)


vector = df['col1']  # slice columns
vector = df.col1     # alternative writing
display(vector)

# Selection by condition
df[df.col1 > 0]

# Set of values
df[ df['col1'].isin([1.0, 2.0]) ]
# Set values by label with `at`
df.at['row1','col2'] = 0
```

```
# Set values by index with `iat`
df.iat[0,1] = 0.0

# Sophisticated: conditional change
df[df > 0] = -df
display(df.dtypes)
```

Operations on data frames

```
# Multiply two vectors
df = pd.DataFrame({'col1' : [1., 2.], 'col2' : [4., 3.]})
df['mult'] = df['col1'] * df['col2']  # element-wise multiplication
display(df)
del df['mult']

# inner products of vectors,
df['col1'].dot( df['col2'] ) == df['col2'].dot( df['col1'] )
# Add a derived col
df['flag'] = df['col1'] > 1.5
display(df)
# Set an entire column to a constant value
df['col3'] = 'constant'
display(df)
del df['col3']
df.insert(1, 'between1and2', df['col1']) # insert col1-copy as between1and2 on second position
display(df)
del df['between1and2']
# Drop any row containing missing data points
df.dropna(how='any')

# Fill missing data points with constant
df.fillna(0.0)
```

```
# Find duplicate rows
dups = df.duplicated("col1")
display(dups)

dfcum = df.cumsum()
display(dfcum)

try:
    dfc.plot(kind='line')
except Exception as e:
    print("plotting fails due to python 3.5")



# Boolean mask of missing data points
#pd.isna(df)
```

Transpose

Transpose is said to be faster than in Stata.

```
df[:5].T



# Matrix multiplication
df.T.dot(df)
```

More operations

```
# Use functions from numpy
np.add(df,2)
```

Sorting

```
df = pd.DataFrame({'col1' : {'row1': 1., 'row2': 2.}, 'col2' : {'row2': 2., 'row3': 3.}})
```

```
display(df)


# sort by values - axis=0 is vertically
df = df.sort_values(by='col1', axis=0, ascending=True, na_position='first')
display(df)


# sort_index

# axis 0 => sort by index  labels - row1,2,3
# axis 1 => sort by column labels - col1,2...

# row1, row2, row3 => row3, row2, row1
df = df.sort_index(axis=0, ascending=True)
display(df)

# col1, col2 => col2, col1
df = df.sort_index(axis=1, ascending=False)
display(df)
```

Plotting

```
dfp = pd.DataFrame({'col1' : {'a': 1., 'b': 2., 'c': 3.}, 'col2' : {'a': 2., 'b': 2., 'c': 3.}})

try:
    display(dfp.plot(kind='line'))
except Exception as e:
    print("Plotting as documented fails - or shows only the object-string - due to python 3.5")


# I did not find the difference between show() or figure - for displaying the chart
display(dfp.plot(kind='line'   , x='col1', y='col2').figure)
display(dfp.plot(kind='scatter', x='col1', y='col2').figure)
```

Pandas data frames can be saved as Excel or as Stata files.

```
#
# pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_excel.html
df.to_excel("./out.xls", sheet_name="tab1", header=True)

# pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_excel.html
# dataset_label="revolutionary_sequences",
df.to_stata("./out.dta", variable_labels={"col01":"a descr for column01"})
```

Merging

Concat for appending dataframes vertically or horizontally

```
import pandas as pd
import numpy  as np

df = pd.DataFrame(np.random.randn(6, 2))
display(df)

# vertical partitioning
pieces = [df[:4], df[4:5], df[5:]]

# show partitioned rows
for i in range (3):
    display(pieces[i])

pieces[i] = pd.DataFrame(np.random.randn(1, 3)) # if we concat a longer row, the index labels are *not*
mapped

# Concatenating it back together
# Concatenating vertically
pd.concat(pieces, axis=0)  #
# Concatenating horizontally

pieces[1].index = [0]  # We have to re-index s
```

```
pieces[2].index = [1]


pd.concat(pieces, axis=1)
```

Horizontal

```
# Merging on index labels - horizontally


import datetime


heights  = pd.DataFrame({'col1_name': ['Alice', 'Bob'], 'col_height': [1.99, 1.44]})

display(heights)


entryDates = pd.DataFrame({'col1_name': ['Alice', 'Bob', 'Cookie Monster'], 'col_entry': [datetime.date
(2010, 5, 24), datetime.date(2012, 2, 13), datetime.date(2014, 12, 31)]})

display(entryDates)



df = pd.merge(heights, entryDates, on='col1_name')

display(df)

# Cookie monster gets abandoned


# To combine by index, use join

# Appending - yet another method of unclear purpose

anotherRow = df.iloc[1]  # select second row

df.append(anotherRow, ignore_index=True)
```

Group by and aggregate functions

```
df = pd.DataFrame({
                    'hair_color' : ['red', 'blonde', 'brown', 'black', 'red', 'blonde', 'brown', 'black
'],
                    'age_group' :  ['adult'] * 4  + ['child'] * 4,
                    'score' : np.random.randn(8),
                  })
```

```
display(df)


totalsByAge = df.groupby("age_group").sum()
display(totalsByAge)



totalsByHair = df.groupby("hair_color").sum()
display(totalsByHair)



byHairAndAge = df.groupby(["age_group", "hair_color"]).sum()
display(byHairAndAge)
# pivot

df = pd.DataFrame({
                    'hair_color' : ['red', 'blonde', 'brown', 'black', 'red', 'blonde', 'brown', 'black
'],
                    'age_group' :  ['adult'] * 4  + ['child'] * 4,
                    'eye_color' :  ['green', 'brown', 'grey', 'blue'] * 2,
                    'score' : np.random.randn(8),
                  })



display(df)
piv = pd.pivot_table(df, values='score', index=["age_group", "hair_color"], columns=["eye_color"])
display(piv)
```

Date ranges

Full fledged dates

```
rng = pd.date_range(start='1/1/2012', periods=3, freq='2M')
```

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
display(ts)  # Notice: It is the last day in each month


rng = pd.date_range(start='1/1/2012', periods=3, freq='2Q')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
display(ts)  # Notice: It is the last day in each month


# To obtain months which start with first day
rng = pd.date_range(start='1/1/2012', periods=3, freq='M')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
display(ts)

# We may convert this to "periods" - if the frequency is not combined with a qualifier
ps = ts.to_period()
display(ps) # periods (this kind) have no day at all

# And back to timestamps => full blown dates
tss = ps.to_timestamp()
display(tss)  # Now it is the first day of month
```

Create periods directly

```
# This way also works with 'qualified' frequencies.
rng = pd.period_range('2012/1', '2012/6', freq='2M')  # create period range directly
ts = pd.Series(np.random.randn(len(rng)), index=rng)
display(ts)
tss = ts.to_timestamp() # Convert to full blown date
display(tss)  # Now it is the first day of month

rng = pd.period_range('2012Q1', '2012Q3', freq='Q')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
display(ts)
```

```
tss = ts.to_timestamp()

display(tss)  # Now it is the first day of month


# The entire frequency stuff is explained here.

# pandas.pydata.org/pandas-docs/stable/timeseries.html#timeseries-offset-aliases

# Delve deeper if you need it.
```

We can not 'mutate' a date or a series:

```
    for i,m in enumerate(rng):

      rng[i] = datetime.date(m.year, m.month, 1)
```

Categories data

```
df = pd.DataFrame({"id":[1,2,3,4,5,6], "raw_grade":['a', 'b', 'b', 'a', 'a', 'e']})



df  = pd.DataFrame({
    'col1_name': ['Alice', 'Bob','Cesar'],
    'col_height': [1.99, 1.44, 1.55],
    'col_cat': ["tall", "medium size", "medium size"],
})


df["col_cat"] = df["col_cat"].astype("category")



display(df)

display(df.info)

display(df.dtypes)

display(df.describe())

df.items  # Iterator over (column name, Series) pairs.


df.query("col_cat == 'medium size'")

display(df.swapaxes(0,1))

df = pd.Series(np.random.randn(4), index=[1,2,3,4])

display(df)
```

```
dfc = df.cumsum()
display(dfc)

try:
    dfc.plot(kind='line')
except Exception as e:
    print("plotting fails due to python 3.5")
```

Numpy charting

```
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display, HTML

CURTIME = datetime.datetime.now()
%matplotlib inline

x = np.linspace(0, 3*np.pi, 500)
plt.plot(x, np.sin(x**2))
#plt.title('A simple chirp')
index = pd.date_range(CURTIME-datetime.timedelta(10), periods=10, freq='D')
columns = ['A','B', 'C']
data = np.array([np.arange(10)]*3).T
df1 = pd.DataFrame(data, index=index, columns=columns)
df1 = df1.fillna(0) # fill with 0s
display(df1)
#HTML(df1.to_html()) # I cannot see a distinction to display()
a = np.random.random((2,3))
display(a)
display(a.sum())
display(a.min())
```

Example - loading data

In [3]:

```
import pandas as pd


# We could add 'Sparbriefe' as alternative data to 'Bankschuldverschreibungen'
# https://index.fmh.de/fmh-index/zinsentwicklung/detailversion/




#difi= pd.read_csv('./difi-2017-10-no-sonderfragen.csv', sep=';')
difi= pd.read_csv('./inp-difi-excerpt.csv', sep=';')



# display(difi.describe)
difi.head(5)
# descriptive statistics; central tendency, dispersion and shape of a dataset’s distribution
# terrible to read

print(difi.shape,  end="\n\n")

print(difi.dtypes, end="\n\n")
difi2 = difi.iloc[ 130:133 ,[0,1,2,9,21,22]]  # slice rows, slice columns
# difi2 = difi.iloc[:,2:10]
display(difi2)
print(difi2.dtypes, end="\n\n")
# Trying `apply`
difiX = difi2.apply(np.sum, axis=0) # unsuited - 'because it lumps all rows together
display(difiX)
difiX = difi2.apply(pd.to_numeric, errors='ignore') # keeps \N
difiX = difi2.apply(pd.to_numeric, errors='coerce') # \N to NaN; but also timestamp to Nan
display(difiX)
print(difiX.dtypes, end="\n\n")
```

Simple OLS example

In [1]:

```
# http://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLS.html


import numpy as np

import statsmodels.api as sm


Y = [1,3,4,5,2,3,4]

X = range(1,8)

X = sm.add_constant(X)


model = sm.OLS(Y,X)

results = model.fit()

results.params

results.tvalues

print(results.t_test([1, 0]))
```

```
                             Test for Constraints
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
c0             2.1429      1.141      1.879      0.119      -0.789       5.075
==============================================================================
```

Example application

We create two data frames. One from a Bankscope-Excel file. Another data frame from CSV data. We aggregate the CSV data and we reformulate the timestamps. We match both data frames and run a simple regression.

```
import pandas as pd

import numpy  as np

import datetime


# We could add 'Sparbriefe' as alternative data to 'Bankschuldverschreibungen'

# index.fmh.de/fmh-index/zinsentwicklung/detailversion/
```

```
def toFloat64(value):
  try:
    return np.float64(value)
  except Exception as e:
    return np.nan
    # return np.float64(0.0)



def toInt64(value):
  try:
    return np.int64(value)
  except Exception as e:
    return np.nan
    #return np.float64(0.0)
''' Reading excel has many options:
        pd.read_excel(io, sheet_name=0, header=0, skiprows=None, skip_footer=0, index_col=None, names=No
ne,
        usecols=None, parse_dates=False, date_parser=None, na_values=None, thousands=None, convert_float
=True,
        converters=None, dtype=None, true_values=None, false_values=None, squeeze=False)
'''

rates= pd.read_excel('./inp-bundesanleihen-bankschuldverschreibungen-pfandbriefe-02.xlsx',
    sheetname="Rates", index_col=0)

rates.columns = ['spr_schuv_over_bunds', 'spr_pfand_over_bunds'] # rename columns
rates['timestmp1'] = rates.index # save original index to new column
rng = pd.period_range('200001', '201710', freq='M')  # create new period range from scratch
#rng = pd.date_range(start='1/1/2012', periods=3, freq='M') # this would create a date range
rates.index = rng # make it the index
rates = rates.to_timestamp() # periods to timestamps
rates['timestmp2'] = rates.index # save timestamps to column
rates.index.name = 'YearMonth'
```

```
rates = rates.to_period() # back to periods


display(rates.head(3))

display(rates.tail(3))



del rates['timestmp1']

del rates['timestmp2']



# survey would be conducted in Jan 2017.

# prognosis is requested for July 2017.

# Thus we compute the change over six month.

# Positive difference means increase in spreads, negative equivalent

rates['spr_schuv_dx'] =  rates.spr_schuv_over_bunds.shift(-6) - rates.spr_schuv_over_bunds

rates['spr_pfand_dx'] =  rates.spr_pfand_over_bunds.shift(-6) - rates.spr_pfand_over_bunds



# display(rates.head(3))

# display(rates.tail(3))


# Showing head and tail rows in one table:

totalRows = rates.shape[0] # number of all rows

display(rates.iloc[[0,1,totalRows-2,totalRows-1,],:])   # slice for display


print(rates.dtypes,"\nindex", rates.index.dtype, end="\n\n\n") # show types of columns, type of index

#difi= pd.read_csv('./difi-2017-10-no-sonderfragen.csv', sep=';')

difi= pd.read_csv('./inp-difi-excerpt.csv', sep=';')  # reading a subset of the real estate finance excp
ectations


difi2 = difi.iloc[ 0: ,[0,1,21,22]]  # slice rows, slice columns => create subset of columns with foreca
sts

#display(difi2.describe)
```

```
#display(difi2.head(5))
nRows = difi2.shape[0]
display(difi2.iloc[[0, 1, nRows-2, nRows-1],:]) # slice for display

print("shape: ", difi2.shape,  end="\n\n")
print(difi2.dtypes, "\nindex", difi2.index.dtype, end="\n\n")  # show types of columns, type of index
# 1 - rising, 2 - unchanged, 3 - falling, 4 - do not know, 0 - no answer
# spread_pfand contains empty values. spread_schuld is an integer
# cleaning up - columnwise

difi2['spread_pfand']  = difi2['spread_pfand'].map(toFloat64)
difi2['spread_schuld'] = difi2['spread_schuld'].map(toInt64)  # leads to float64, because int does not h
ave nan.


difi2['spread_pfand' ].replace(4.0,np.nan,inplace=True) # 4.0 is encoding of "don't know"
difi2['spread_pfand' ].replace(0.0,np.nan,inplace=True) # 0.0 is encoding of "no answer given"
difi2['spread_schuld'].replace(4.0,np.nan,inplace=True)
difi2['spread_schuld'].replace(0.0,np.nan,inplace=True)
#display(difi2) # check results


display(difi2.iloc[[0, 1, nRows-2, nRows-1],:]) # slice for display
print(difi2.dtypes, "\nindex", difi2.index.dtype, end="\n\n")  # show types of columns, type of index
difiTotals = difi2.groupby(["pg.survey_id", "pg.person_id"]).sum() # values
#display(difiTotals)

difi3 = difi2.groupby(["pg.survey_id"]).mean()
difi3 = difi2.groupby(["pg.survey_id"]).agg(['mean', 'count'])  # pg.survey_id becomes index; and seems
to be converted to type period


# get first level columns keys
```

```
# print(difi3.columns.get_level_values(0))



# flatten the resulted multi index
newCols = []
for i, col in enumerate(difi3.columns.values):
    newCol = '__'.join(col).strip()
    # print(i, newCol)
    newCols.append(newCol)

difi3.columns = newCols



# An example for 'syntactic sugar' for above block would be:
# difi3.columns = ['__'.join(col).strip() for col in difi3.columns.values]



display(difi3)
del difi3['pg.person_id__mean']
difi3.rename(columns={'pg.person_id__count': 'countTotal'}, inplace=True)
del difi3['spread_pfand__count']
del difi3['spread_schuld__count']







# So far, data is centered around two.
# We center it around zero:
difi3['spread_schuld__mean'] = difi3['spread_schuld__mean'] - 2
difi3['spread_pfand__mean']  = difi3['spread_pfand__mean']  - 2



# Because of the counter-intuitive encoding of  1 - rising but 3 - falling
difi3['spread_schuld__mean'] = - difi3['spread_schuld__mean']
```

```
difi3['spread_pfand__mean']  = - difi3['spread_pfand__mean']
# Now positive values indicate forecasting rising rates; and vice versa



display(difi3)
print(difi3.dtypes, "\nindex: ",difi3.index.dtype, end="\n\n")  # show types of columns, type of index
# The survey periods are oddly encoded as an integer YearMonth: YYYYMM
# Let's create an index compatible to above: period(M)
# The existing index is supposed to be int64

# Period or Integer are converted to datetime64
def toDateTime64(x):
    if type(x) is pd.Period:
        return x.to_timestamp()
    t1 = np.int64(x/100)*100
    t2 = np.int64(x-t1)
    x = datetime.date(int(t1/100), t2, 1)   # simple datetime.date - but insufficient
    x = np.datetime64(x)  # elevate to numpy-datetime; thus we can call 'to_period()' on the column late
r on.
    return x

difi3['work_column'] = difi3.index # create new column
difi3.index = difi3['work_column'].map(toDateTime64)
difi3.index.name = 'YearMonth'
# difi3.rename(index={1: 'YearMonth1'})  # alternative rename

difi3 = difi3.to_period('M') # convert to period
#difi3 = difi3.to_timestamp() # or back to timestamp

del difi3['work_column']

display(difi3)
print(difi3.dtypes, "\nindex: ",difi3.index.dtype, end="\n\n")  # show types of columns, type of index
# We COULD align the time series to account for lag
```

```
# For instance: difi data three months ahead of observed spread data

if False:
    from pandas.tseries.offsets import *
    difi4 = difi3.copy()
    difi4 = difi4.to_timestamp()
    difi4.index = difi4.index - DateOffset(months=3, days=0)
    difi4 = difi4.to_period('M')
    display(difi4)
    print(difi4.dtypes, "\nindex: ",difi4.index.dtype, end="\n\n")  # show types of columns, type of ind
ex
# Prepare the column on which to merge:
rates['MergeOn'] = rates.index
difi3['MergeOn'] = difi3.index

merged1 = pd.merge(rates, difi3, on='MergeOn')
merged1.index = merged1['MergeOn']
#display(merged1.head(10))

# reorder and reduce columns
# display(merged.columns.tolist())
merged2 = merged1[[
    'spread_schuld__mean',
    'spr_schuv_dx',
    'spread_pfand__mean',
    'spr_pfand_dx',
    'countTotal' ,
]]


# pd.options.display.float_format = '{:,.4f}'.format   # change the global style
styledDf = merged2.style.format({
    'spr_pfand_dx': '{:,.2f}', 'spr_schuv_dx':'{:,.2f}',
    'spread_schuld__mean': '{:,.5f}', 'spread_pfand__mean':'{:,.5f}',
```

```
})
display(styledDf)
print(merged2.dtypes, "\nindex: ",merged2.index.dtype, end="\n\n")  # show types of columns, type of index
import statsmodels.formula.api as sm

# stackoverflow.com/questions/19991445/run-an-ols-regression-with-pandas-data-frame
# full docs
# www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLS.html

# model = sm.ols(formula="spr_schuv_dx ~ spread_schuld__mean + spread_pfand__mean", data=merged2)
model = sm.ols(formula="spr_schuv_dx ~ spread_schuld__mean", data=merged2)
result = model.fit()
print( result.params, "\n\n")
print( result.tvalues,"\n\n")
print( result.summary())
import statsmodels.formula.api as sm

result = sm.ols(formula="spr_schuv_dx ~ spread_schuld__mean", data=merged2).fit()
print( result.params,"\n\n")
print( result.tvalues,"\n\n")
print( result.summary(),"\n\n")
```

Collecting data from mutliple excel files with multiple tabs into a CSV file

```
import pandas as pd
from pprint import pprint
import math
import glob
import re
import unicodecsv as csv

# make panda output wide...
```

```
pd.set_option('display.width', 120)
pd.set_option('display.max_columns', 12)



# Write results to CSV
def openWrtr(suffix='sfx'):
    ofn = "out_%s.csv" % (suffix)
    outFile = open(ofn, 'wb')
    w = csv.writer(outFile, encoding='utf-8',delimiter=';',quotechar = '"',quoting = csv.QUOTE_MINIMAL)
    return w

# Close output CSV file at the end
def cleanUpWrtr():
    try:
        wrtr.close()
    except Exception as e:
        pass
        # print("could not close wrtr; %s" % str(e))



wrtr = openWrtr()  # open a writer

countriesLongShort = {
    "Belgium" : "BE","Bulgaria" : "BG","Denmark" : "DK","Germany" : "DE","Estonia" : "EE","Finland" : "F
I","France" : "FR","Greece" : "EL","Ireland" : "IE","Italy" : "IT","Croatia" : "HR","Latvia" : "LV","Lit
huania" : "LT","Luxembourg" : "LU","Malta" : "MT","Netherlands" : "NL","Austria" : "AT","Poland" : "PL",
"Portugal" : "PT","Romania" : "RO","Sweden" : "SE","Slovenia" : "SI","Slovakia" : "SK","Spain" : "ES","C
zech Republic" : "CZ","Hungary" : "HU","United Kingdom" : "UK","Cyprus" : "CY","Albania" : "AL","Montene
gro" : "ME","Macedonia" : "MK","Serbia" : "RS","Turkey" : "TR","Iceland" : "IS","Liechtenstein" : "LI","
Norway" : "NO","Switzerland" : "CH"
}

countriesLongShortLc = {
    "belgium" : "BE", "bulgaria" : "BG", "denmark" : "DK", "germany" : "DE", "estonia" : "EE", "finland"
 : "FI", "france" : "FR", "greece" : "EL", "ireland" : "IE", "italy" : "IT", "croatia" : "HR", "latvia"
 : "LV", "lithuania" : "LT", "luxembourg" : "LU", "malta" : "MT", "netherlands" : "NL", "austria" : "AT
```

```
", "poland" : "PL", "portugal" : "PT", "romania" : "RO", "sweden" : "SE", "slovenia" : "SI", "slovakia"
 : "SK", "spain" : "ES", "czech republic" : "CZ", "hungary" : "HU", "united kingdom" : "UK", "cyprus" :
"CY", "albania" : "AL", "montenegro" : "ME", "macedonia" : "MK", "serbia" : "RS", "turkey" : "TR", "icel
and" : "IS", "liechtenstein" : "LI", "norway" : "NO", "switzerland" : "CH"
}

# Extract country from file name.
def countryFromFile(file):
    orig = file
    if file.startswith("ETA_Datascheme_"):
        file = file[len("ETA_Datascheme_"):]
    if file.endswith(".xlsm"):
        file = file[:len(file)-len(".xlsm")]
    if file.lower() == "vorlage":
        return False

    return file


# Panda cells are empty in various ways...
def checkEmpty(cell):
    if type(cell) == float:
        if math.isnan(cell):
            return True
    if type(cell) == str:
        if cell == "":
            return True
    return False


regex1 = re.compile(r"[^0-9A-Za-z_]", re.IGNORECASE)
regex2 = re.compile(r"[_]+")

# Umlaute to underscores
# Multiple underscores to one
```

```
def normalize(param):
    param = regex1.sub("_",param)
    param = regex2.sub("_",param)
    return param


print(normalize("Dänemark   hat __Öse__n"))




# Load an excel file and sheet using pandas.
#
# Returns a pandas dataframe "df"
#
# Terminology
# index   - row labels
# columns - column labels
# header  - row for column labels; 0-based
def loadCountry(fileName,sheetName,colIdx=0,rowIdx=1):
    exc = None
    try:
        exc = pd.read_excel(fileName,
            sheetname=sheetName,
            index_col=colIdx,
            header=rowIdx,
            skiprows=0 )
    except Exception as e:
        print("could not open tab %s in file %s - %s" % (sheetName,fileName,str(e)))
        quit()
    return exc

# Offsets for rows and columns changed
def loadMeta(fileName,sheetName):
```

```
    return loadCountry(fileName,sheetName,colIdx=-1,rowIdx=0)


# Some data from a pandas dataframe
def overview(df):
    print("\ncols: ")
    print(df.axes[1])
    print("\n")
    print("\nrows: ")
    print(df.axes[0][:8])
    print("\n\n\nthe head-----")
    print(df.head())


# For muting print() statements - change to printDummy...
def printDummy(arg1):
    pass

# Loading Rainers worksheet-parameters file
df = loadMeta('Values_Excel_Database.xlsx','Tabelle1')

# overview(df)

# Some further data from the data frame.
# Demonstrating [rows]-[cols] subsets
if False:
    print("\n\n\nrows 8 to 10-----")
    print(df[8:10].head())

    print("\n\n\none col - some rows")
    print(df['PayTax1Base'][2:4])

    print("\n\n\nsome cols - some rows")
    print(df[1:7][3:5])
```

```
    print("\n\n\nfilter")
    filter = df[(df.index == 'WarrantyProvision')]
    filter = df[(df.index == 'WarrantyProvision') & (df['BalanceSheetReserves'] == 'yes')]
    print(filter)


effectiveCols = []
printDummy("\n\n\n")
for x in df.axes[1]:
    if x.startswith("Unnamed:"):
        continue
    effectiveCols.append(x)

print("effective cols are: ",effectiveCols)

sheetNamesToRows = {} # collecting Rainers sheetname-param info into an interable struct

printDummy("\n\n\n")
for x in effectiveCols:
    printDummy("rows for col %s are" % (x))
    rowx = df[x]
    sheetNamesToRows[x] = []
    for cell in rowx:
        if checkEmpty(cell):
            continue
        printDummy(cell)
        sheetNamesToRows[x].append(cell)
    printDummy("\n\n")


# pprint(sheetNamesToRows)

# Now we collect the excel files for the countries
```

```
# ====================================

# glob.glob('./[0-9].*')
files = glob.glob('ETA_Datascheme_*.xlsm')
filesCleansed  = []
filesCountries = []
for file in files:
    country = countryFromFile(file)
    if country == False:
        continue
    filesCountries.append(country)
    filesCleansed.append(file)

print("filesCountries are",filesCountries)
print("filesCleansed  are",filesCleansed)


# the years are just given
yearCols = [1998, 2002, 2005, 2007, 2009, 2011, 2013, 2015]


for shName, params in sheetNamesToRows.items():
    if shName == "Tax_Company_Modules":
        continue
    if shName != "PayrollTax":
        continue
    print("tab  %s" % shName)
    for file in filesCleansed:
        print("file %s" % file)
        sht = loadCountry(file,shName) # loadCountry("ETA_Datascheme_Austria.xlsm",shName)
        if sht is None:
            continue
        for param  in params:
            hasValues = False
```

```
            try:
                shtRow = sht[(sht.index == param)]
            except Exception as e:
                print("Could not get the row for param %s in tab %s file %s - %s" % (param,shName,file,s
tr(e)))
                quit()
                continue

            for yr in yearCols:
                for cell in shtRow[yr]:
                    if checkEmpty(cell):
                        continue
                    hasValues = True

            if not hasValues:
                print("  no values: %s" % param)
                continue

            print("  param %s" % param)
            shtRow = sht[(sht.index == param)]
            # print("    row: ",shtRow)
            for yr in yearCols:
                # print("      yr: ",yr, shtRow[yr])
                for cell in shtRow[yr]:
                    if checkEmpty(cell):
                        continue
                    print("        ",yr,cell)
                    outRow = [shName,normalize(param),yr,countriesLongShort[countryFromFile(file)],cell]
                    wrtr.writerow(outRow)


        print("\n\n")
```

```
cleanUpWrtr()
```

Outlook

- Data mining
- Otree programming
- (Alexa programming)
- Robot programming
- Quantopian financial data analysis
- Blockchain stress testing
- Lambda computations with Amazon Web Services

Parallel execution with lambda

```
# http://pywren.io/
import pywren


def my_function(b):
    x = np.random.normal(0, b, 1024)
    A = np.random.normal(0, b, (1024, 1024))
    return np.dot(A, x)


pwex = pywren.default_executor()
res = pwex.map(my_function, np.linspace(0.1, 100, 1000))
```

Appendix

There is an online sandbox
https://www.jdoodle.com/python3-programming-online

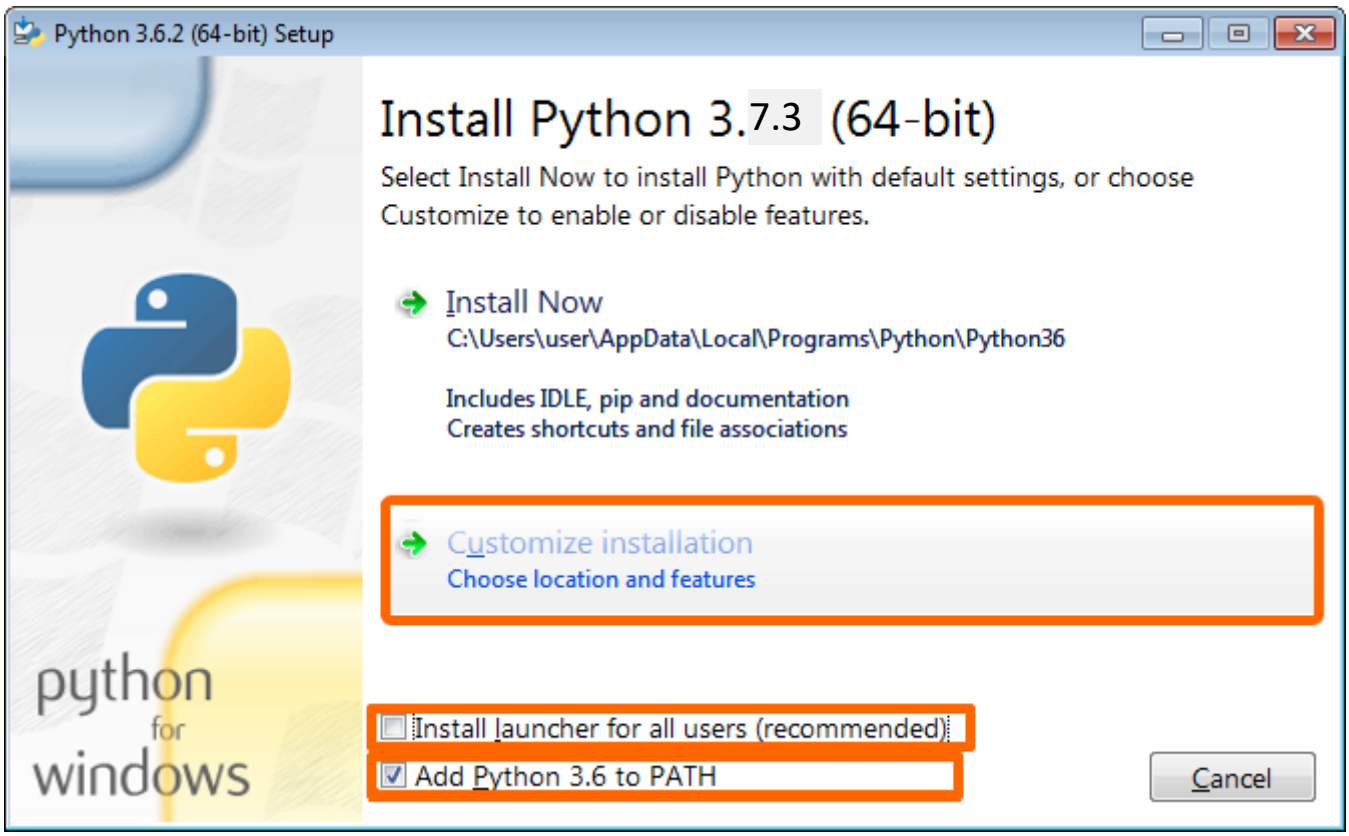
Install Python on your Windows computer

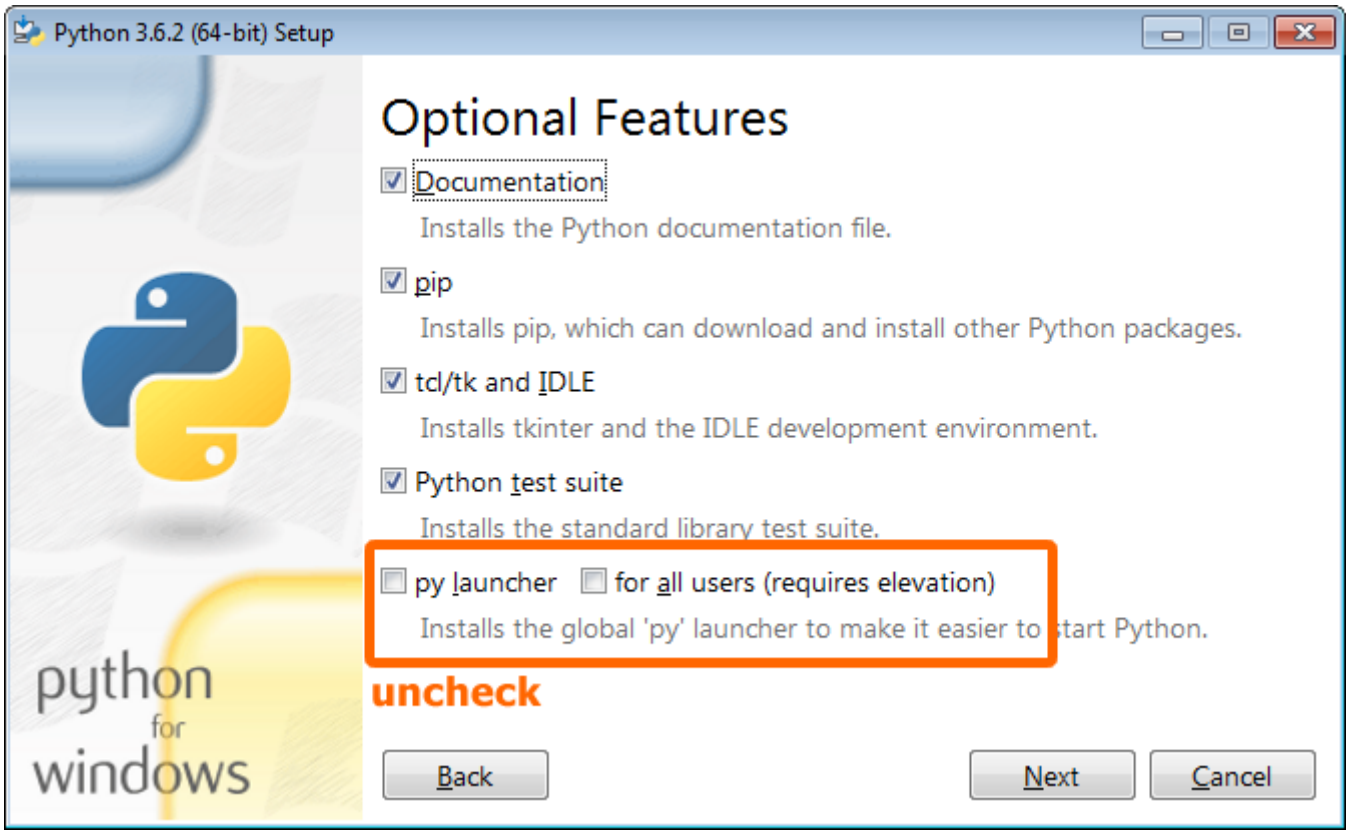
- Please have your working environment ready at the start of the course

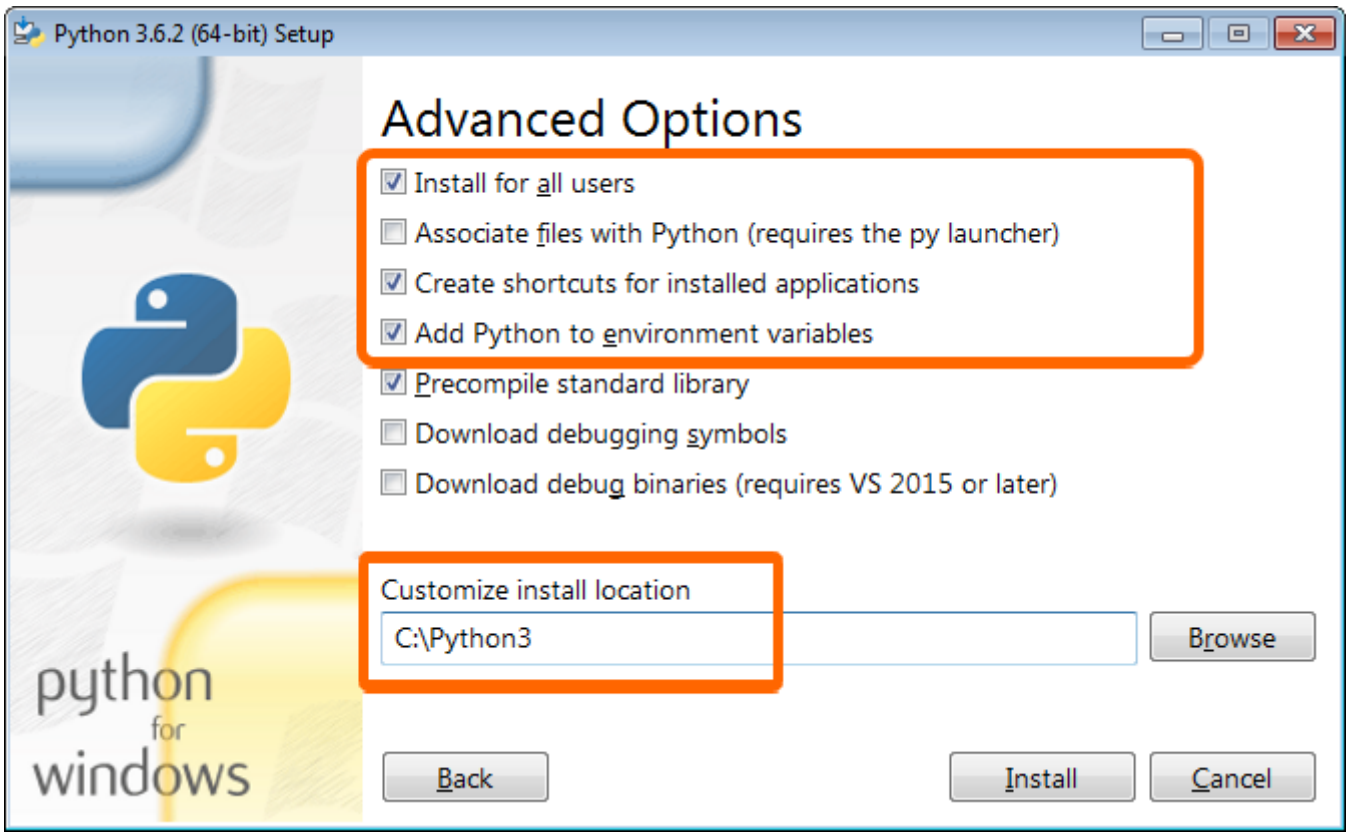
- If you have administrator privileges on your computer, install Python on it.
  Otherwise request the installation from your IT department.
  Download the Python installer program:
  - 64-Bit Windows: https://www.python.org/ftp/python/3.7.3/python-3.7.3-amd64.exe (https://www.python.org/ftp/python/3.7.3/python-3.7.3-amd64.exe)
  - 32-Bit Windows: (https://www.python.org/ftp/python/3.7.3/python-3.7.3.exe)

Graphic installer (explicit install)

- Login as admin (or you will asked later by Windows)
- Double click on the installer and select the options as follows







or you may choose

Check your Python installation

```
"c:\Python3\python.exe" --version
```
or
```
python --version
```

Both should yield
`Python 3.7.3` or `Python 3.7.x`

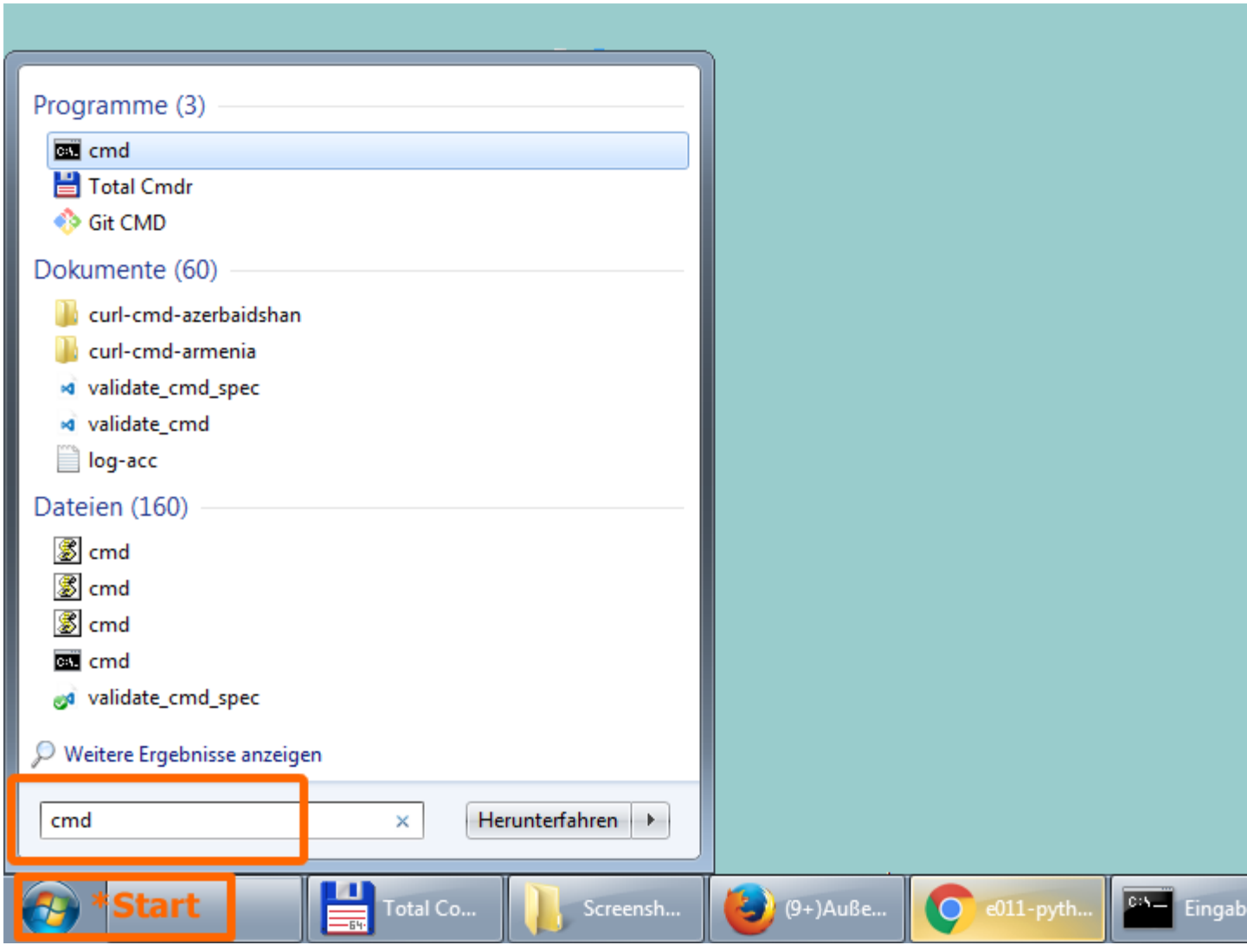
Accept the command window

There is no way around it.

The good news is, that you only have to learn ten little things and that the repulsion quickly subsides.

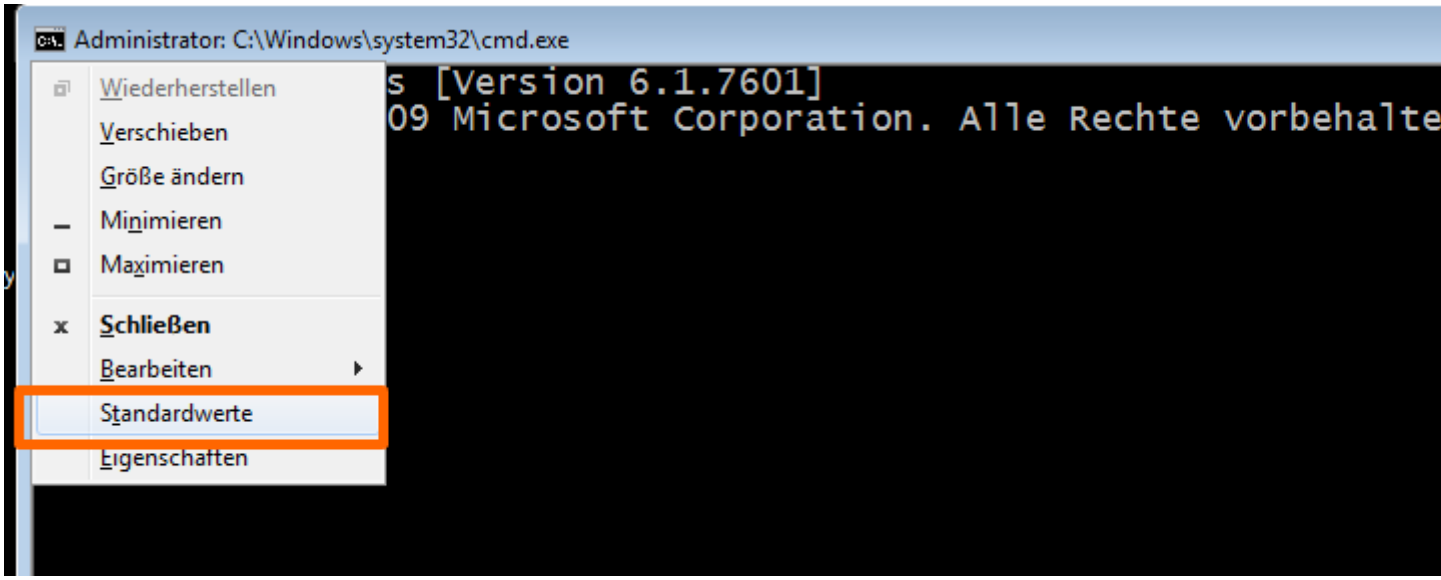
Customize the command window

Open the command window

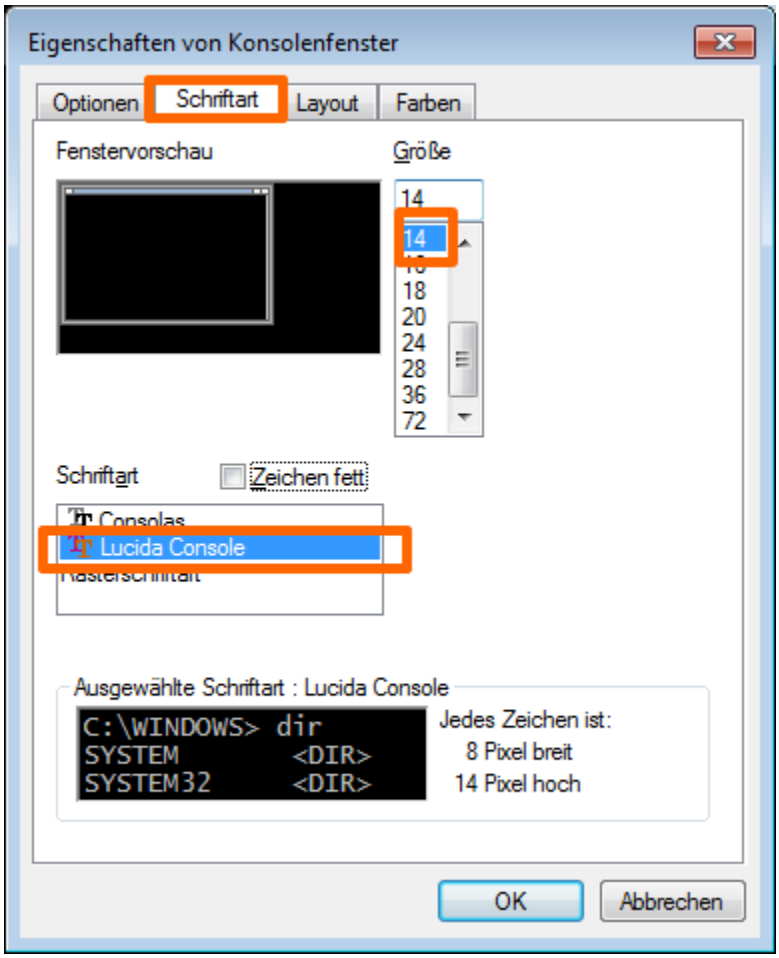


The default settings of the command window are unworkable.
We need to change them once.

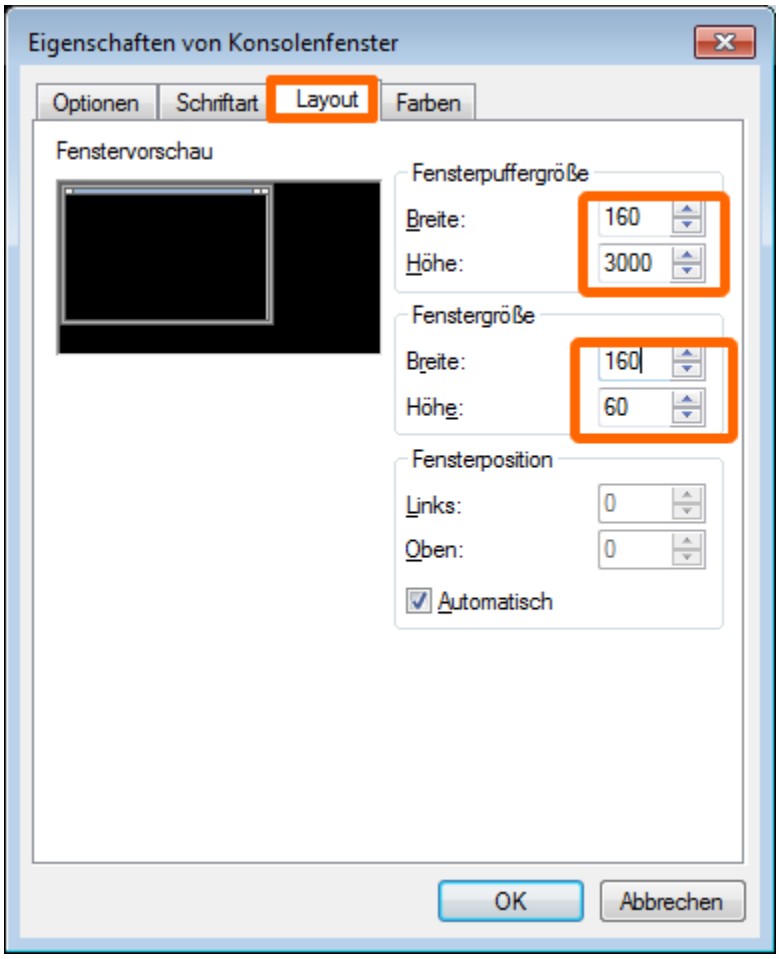
Select `default settings` (Standardwerte)



Set a finer font:



Make the window wider and taller:



To see the changes:
Close the command window wiht `exit`
Reopen it (`Start-cmd`)

Work with the command window

- See the files on windows
  `dir`
  (directory)

- Change directory one up
  `cd ..`

- Change to your home directory on windows
  `cd %USERPROFILE%\Desktop`

- Repeat the last command on windows
  `F3`

- Repeat recent commands
  `CursorUp`
  `CursorDown`

- Clear output on windows
  `cls`
  (clear screen)

- Close the command window
  `exit`

On Mac/Linux

- See the files on mac/linux
  `ls -lAh`
  (list)

- Change to your home directory on mac/linus
  `cd $HOME`

Install and update Python modules with `pip`

`pip` stands for `Python Installation Program`

It serves extensions to the Python language, so called `modules`

Open command window.

Install a certain module; for example `pandas`

```
pip show pandas
```

```
pip install pandas
```

```
pip install pandas --upgrade
```

You can list all installed modules with

```
pip list
```

Run python code

Directly in command window

```
python -c "print('Hello World')"
```

Python Shell (starting in command window)

```
python
```

```
1+1
```

```
quit()
```

Execute a file with python commands

```
python my-python-program.py
```

Leave the command window

```
exit
```

Install a local Python editor

VSCode is an open source project, that swept the board in 2017.
Leading Microsoft and Google programmers are using it.
It is also very friendly to beginners.

https://code.visualstudio.com/Download (https://code.visualstudio.com/Download)

After downloading and installing it...

- Install the extension `Python by` **`Microsoft`**
  (This *includes* the Microsoft Python extension listed above)

Other extensions you may find interesting

- Stata Language
- Bookmarks by Alessandro Fragnani

`Git` not required

VSCode asks for the setup path of the `Git` software.
Git is not required for VSCode.

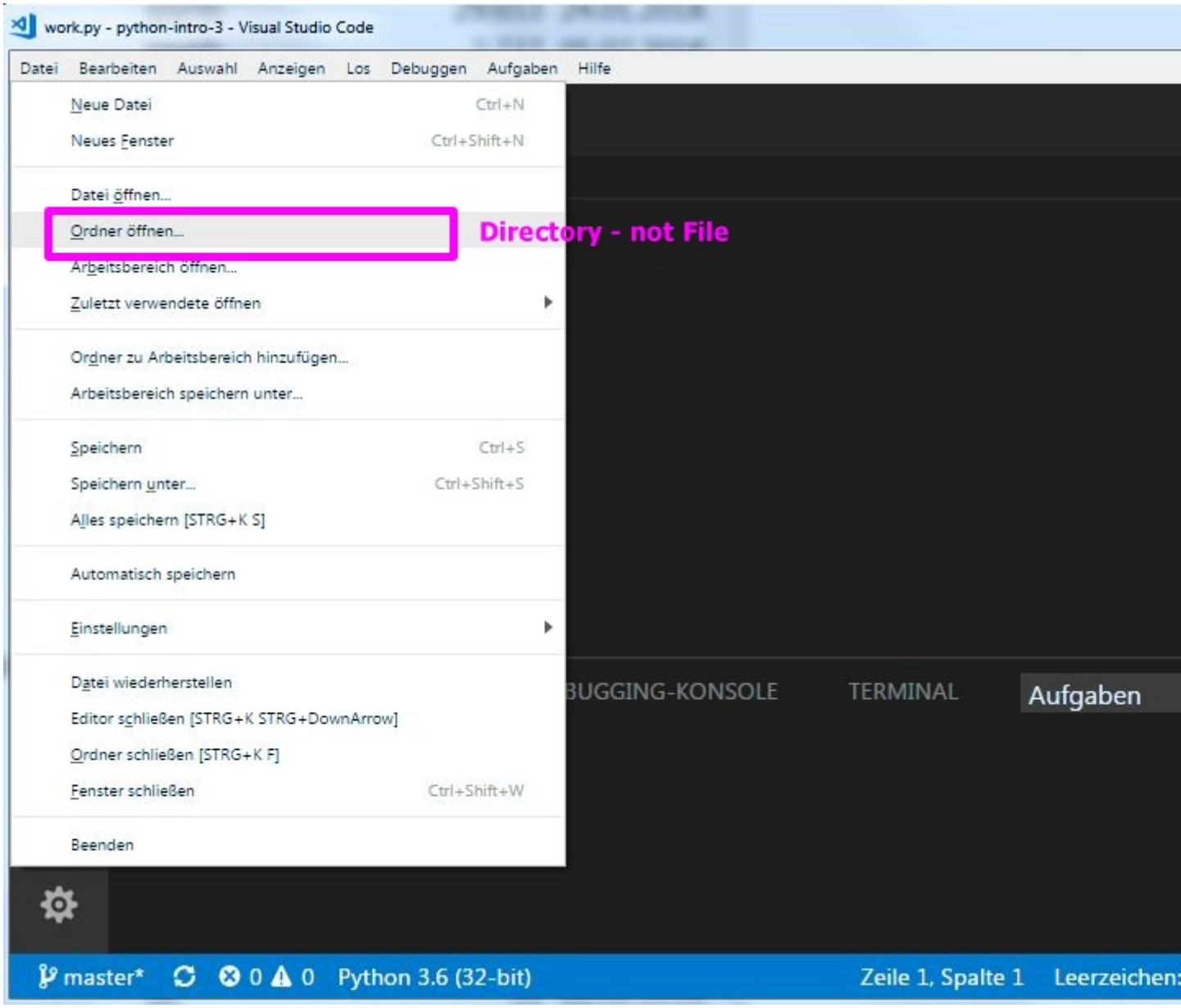
Configure running Python directly from VSCode

[Extended instructions (from microsoft websites) (https://go.microsoft.com/fwlink/?LinkId=733558)](https://go.microsoft.com/fwlink/?LinkId=733558)

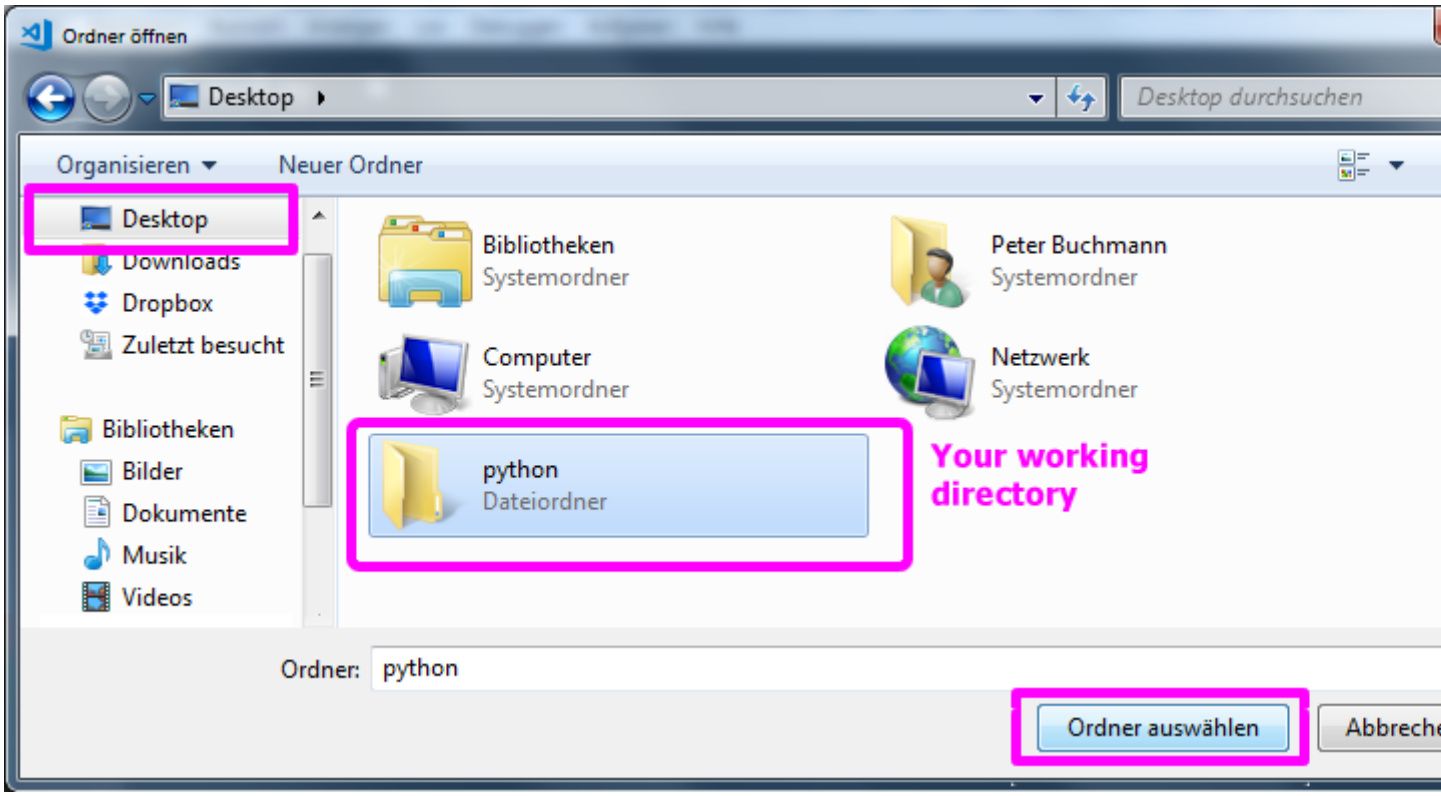
- Decide on a work directory
  for instance

  `c:\user\[username]\Desktop\python`
  or
  `c:\user\[username]\Documents\python`

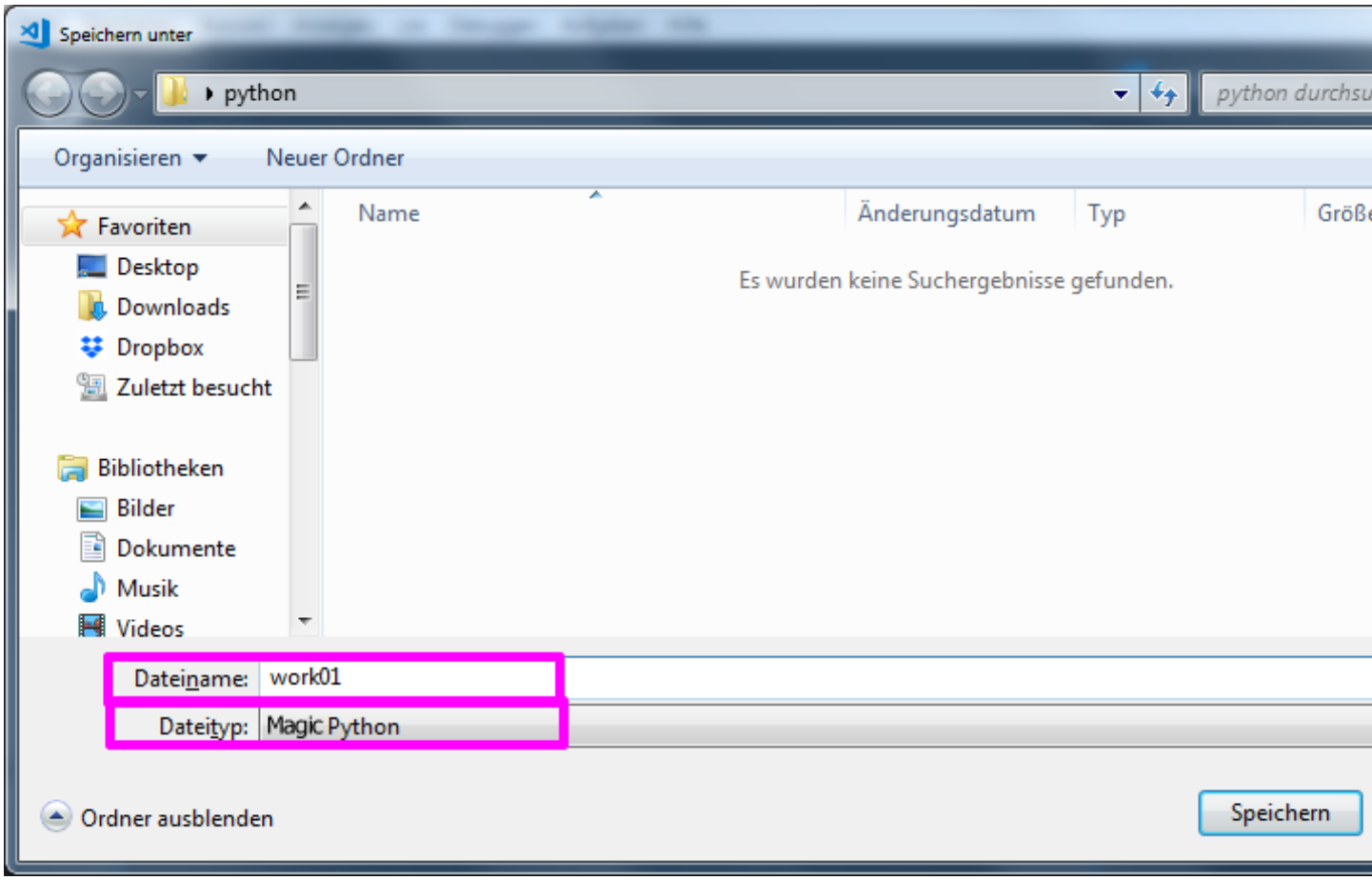
- Start VSCode



- Navigate to `...\Desktop\` (where your python *working directory* from above is located)

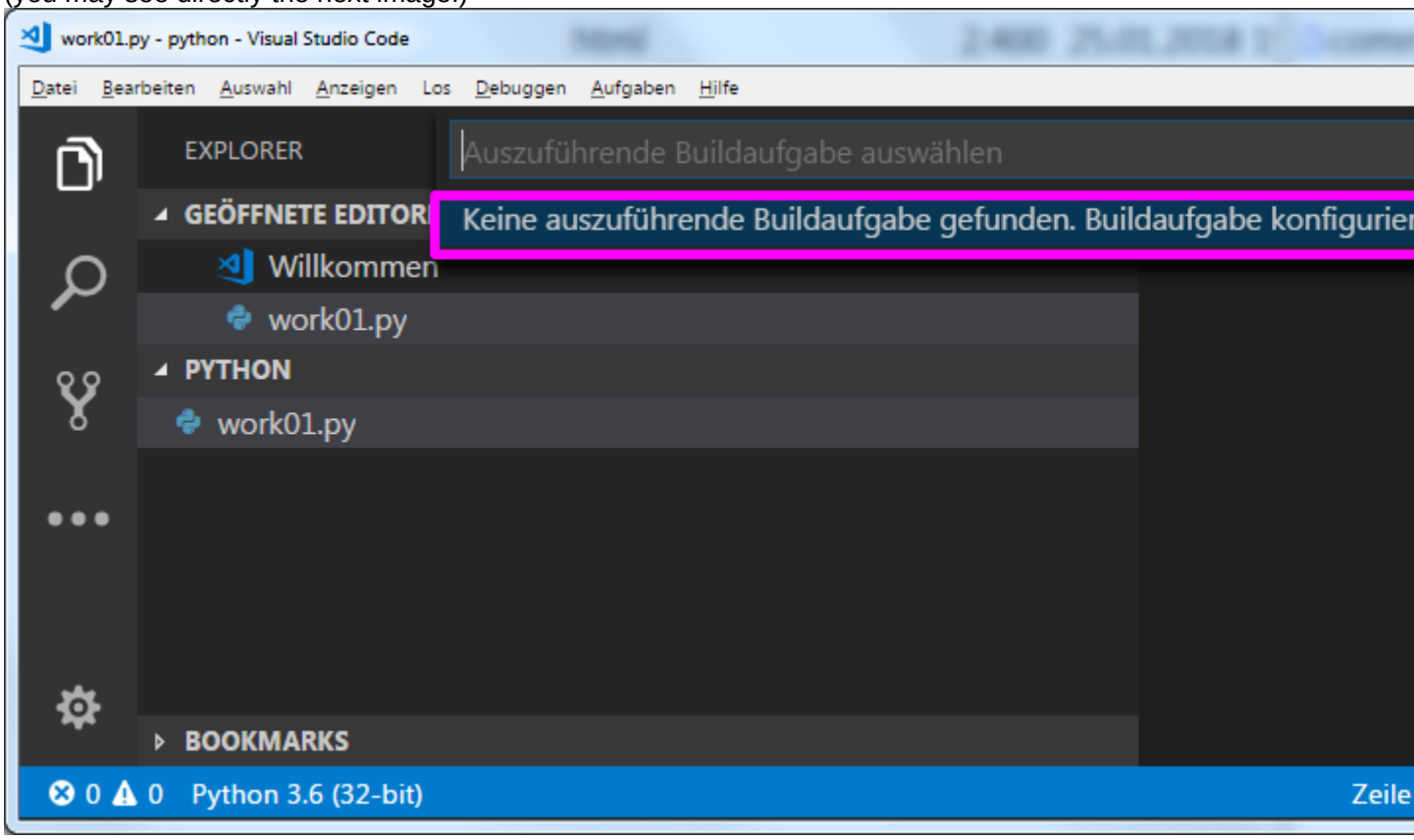


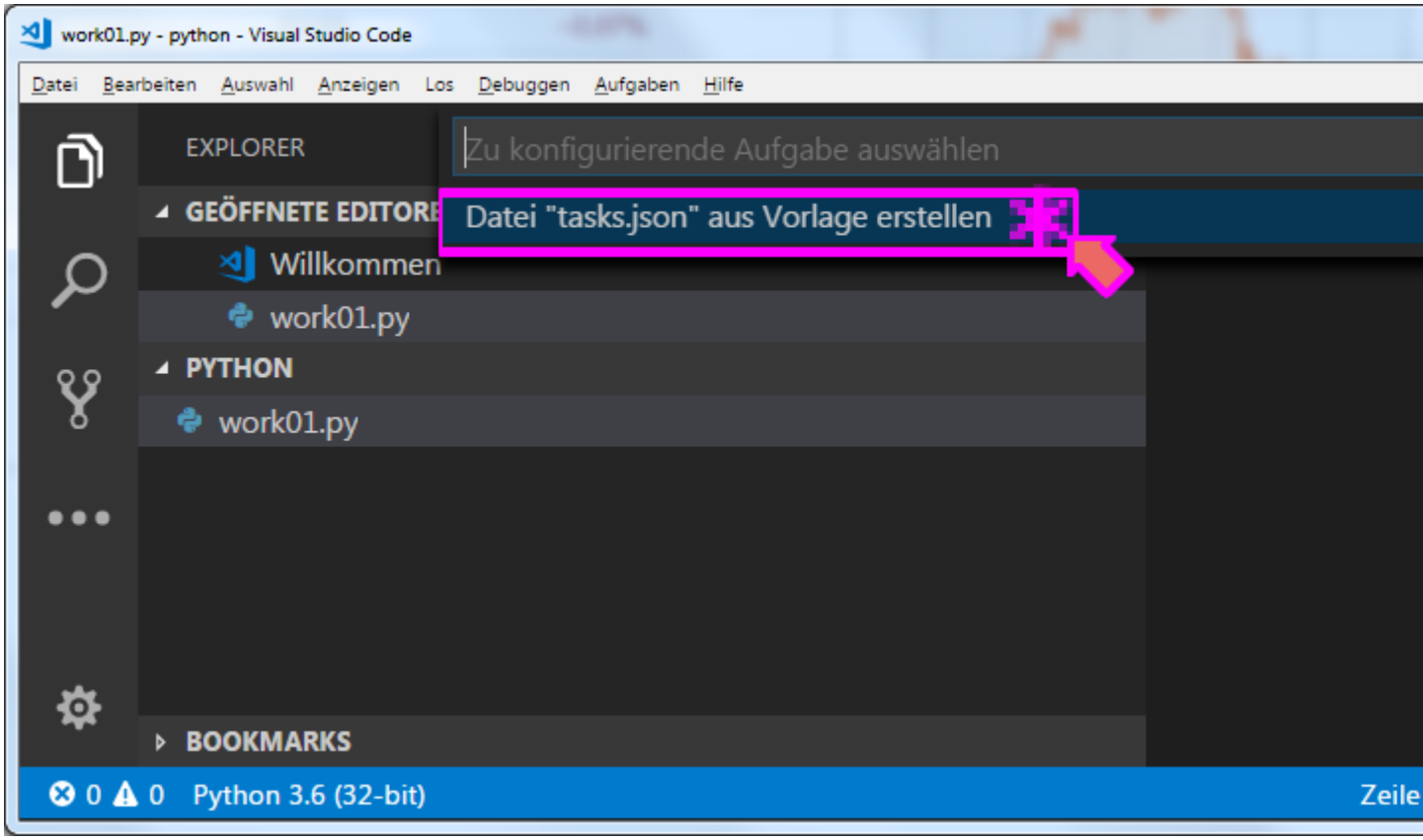
- Now chose menu `file - new`
- And `save as` MagicPython file:

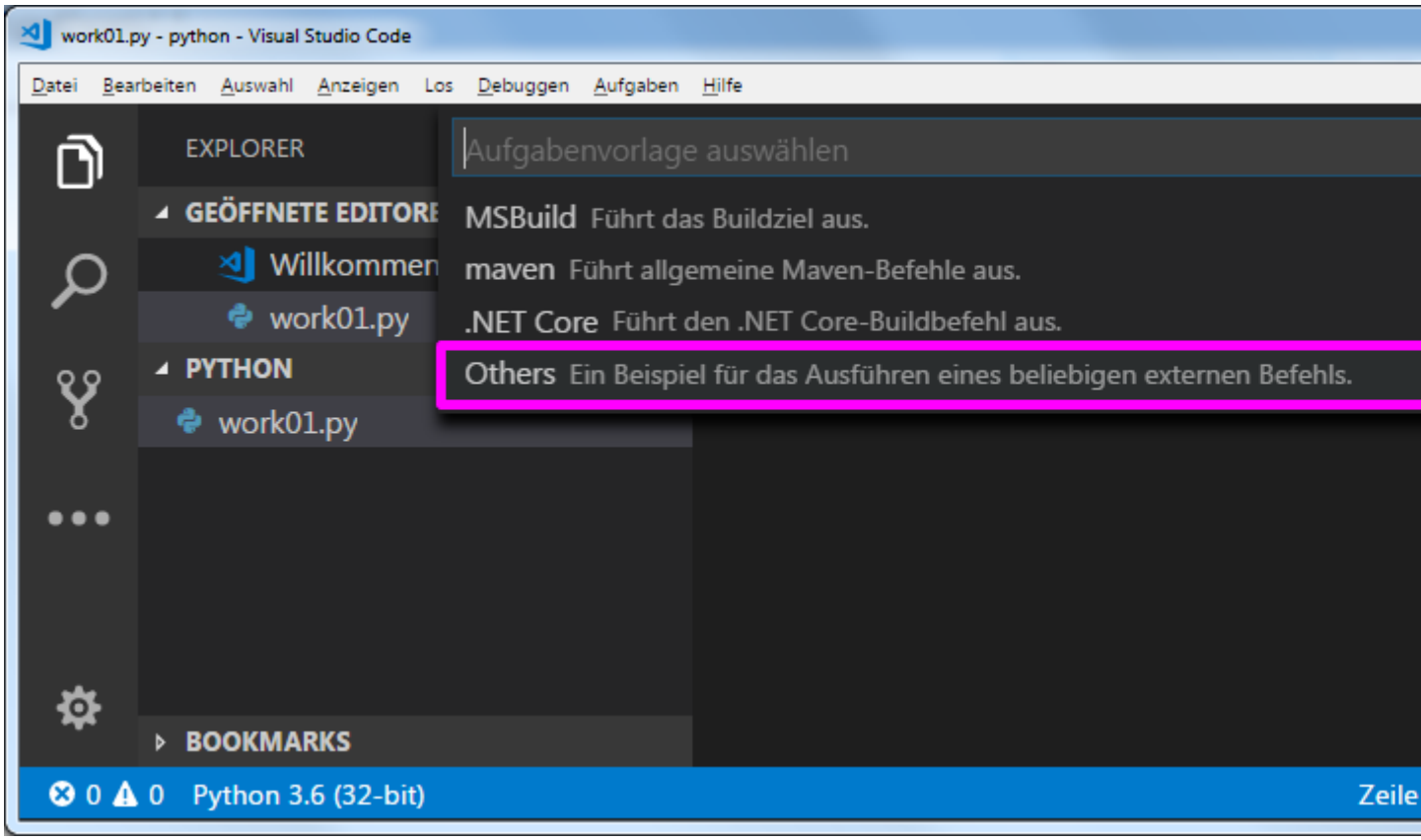


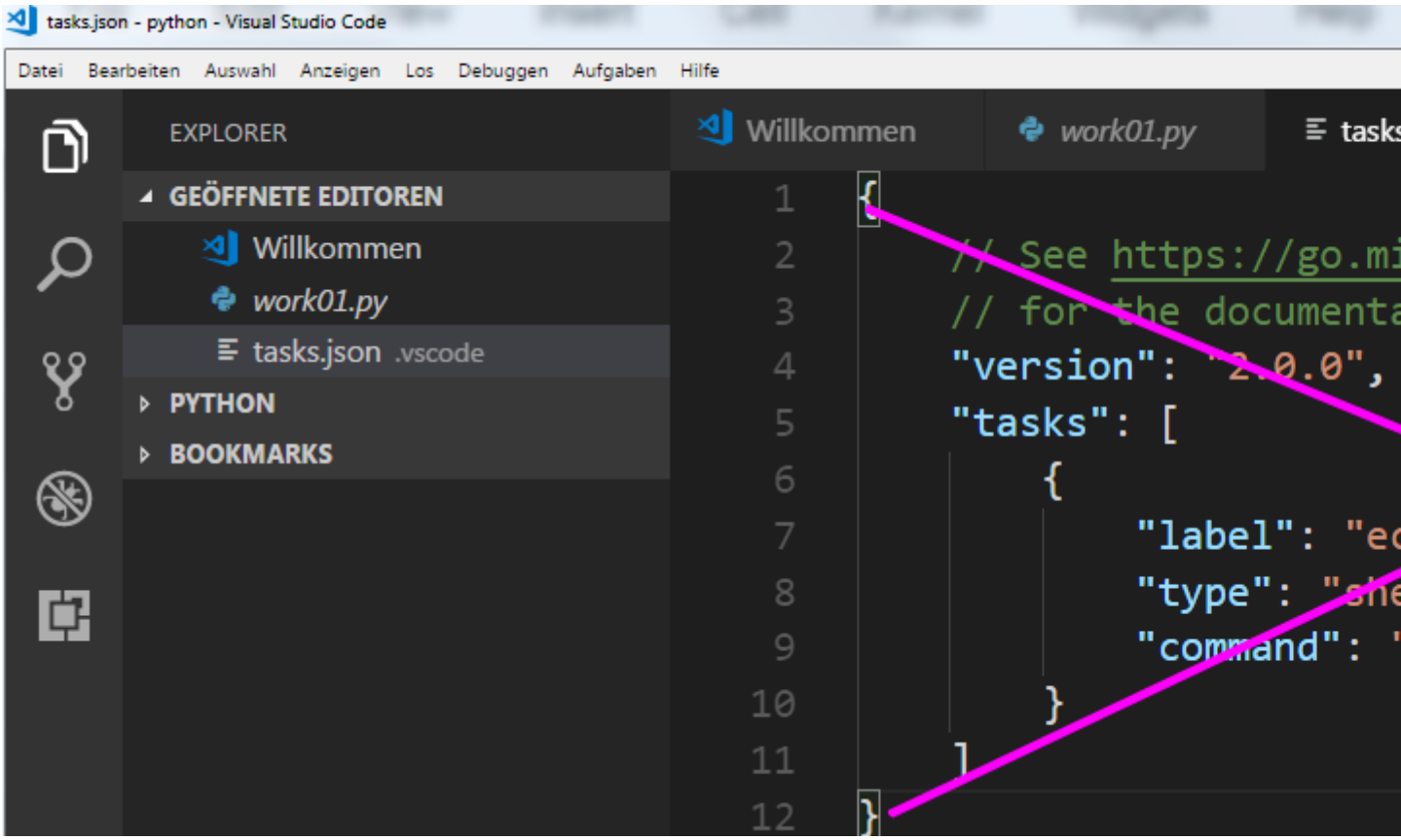
CTRL+SHIFT+P
`tasks - configure task`

(you may see directly the next image:)







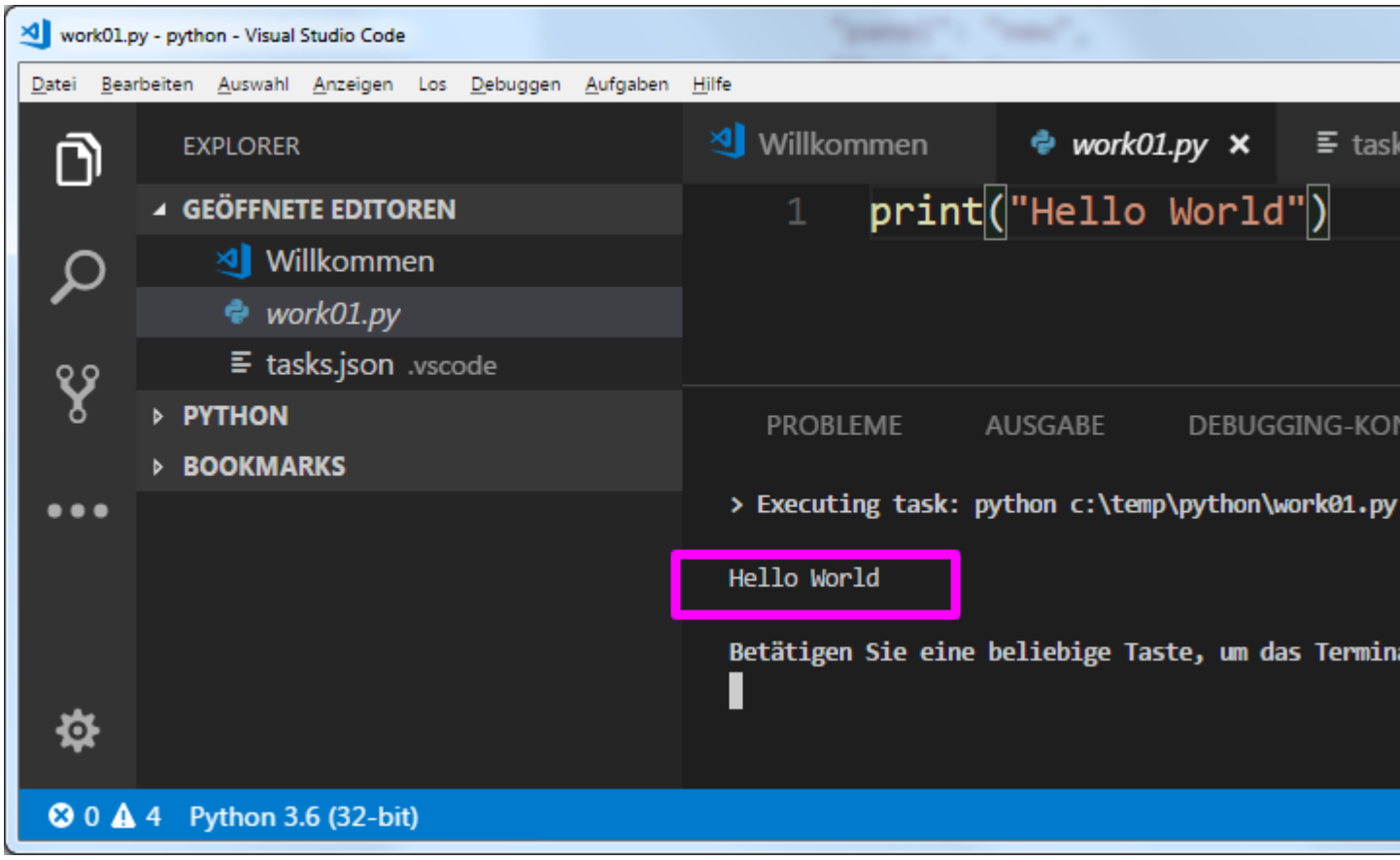


- Overwrite the entire content with the following

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "label": "Run Python File",
            "command": "python ${file}",
            "type": "shell",
            "group": {
                "kind": "build",
                "isDefault": true
            },
            "presentation": {
                "reveal": "always",
                "panel": "new",
                "focus": true
            }
        }
    ]
}
```

[(Source: stackoverflow) (https://stackoverflow.com/questions/45409845/configure-vs-code-version-2-0-0-build-task-for-python)](https://stackoverflow.com/questions/45409845/configure-vs-code-version-2-0-0-build-task-for-python)

- Now you can run ("build") your Python program with `CTRL+SHIFT+B`



Useful editor commands (for VSCode but also other Editors)

- Use CTRL+# to comment - uncomment
- TAB and SHIFT+TAB to indent - outdent

Maintenance of installed modules

You can carry out maintenance roughly every six month.

Update all modules

```
import pip

from subprocess import call


packages = [dist.project_name for dist in pip.get_installed_distributions
()]

call("pip3 install --upgrade " + ' '.join(packages), shell=True)
```

Updating modules from a requirements file

Some python programs contain files named requirements...txt.
These contain needed modules with minimum versions.

Create a file `requirements_base.txt`:

```
otree-core>=0.7.4

Django==1.8.8          # otree-heroku demand this exact version
```

To install or update such modules

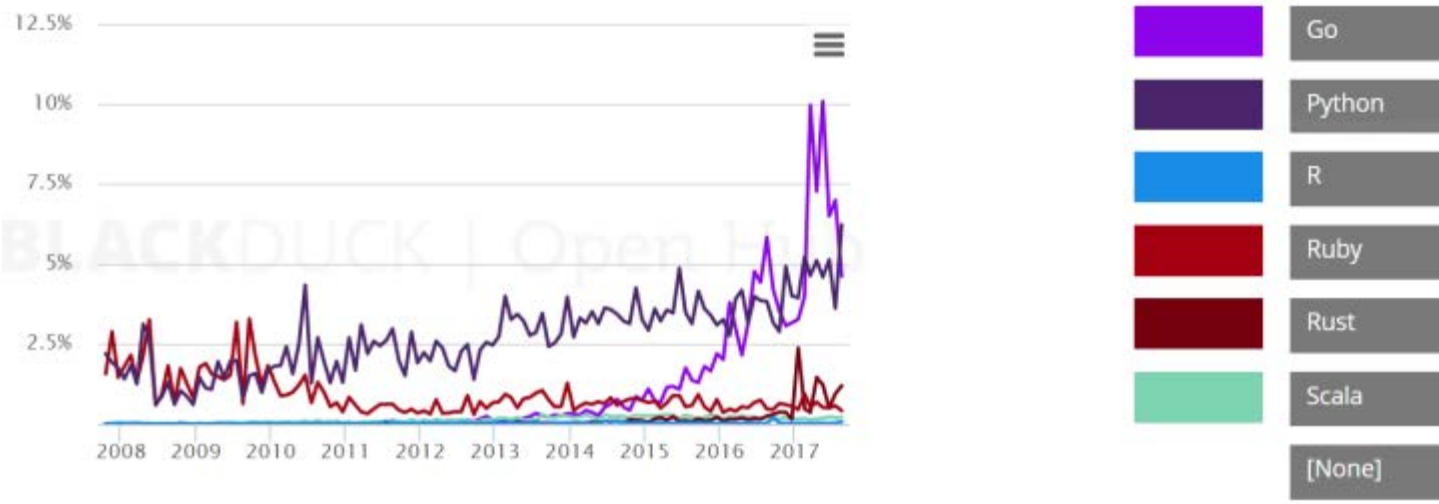
```
call("pip3 install -r requirements_base.txt --upgrade", shell=True)
```

Stats for the break

Most loved - most demanded (https://www.heise.de/developer/meldung/Programmiersprachen-2017-Vielfalt-ist-gefragt-3861018.html)

	Most loved	Most wanted	Most dreaded
1	Rust	Python	Visual Basic 6
2	Smalltalk	JavaScript	VBA
3	TypeScript	Go	CoffeeScript
4	Swift	C++	VB.NET
5	Go	Java	Matlab
6	Python	TypeScript	Objective-C
7	Elixir	C#	Assembly
8	C#	Swift	Perl
9	Scala	Ruby	Lua
10	Clojure	Rust	Hack

Lines of code changed on github (https://www.openhub.net/languages/compare?utf8=%E2%9C%93&measure=loc_changed&language_name%5b%5d=-1&language_name%5b%5d=golang&language_name%5b%5d=python&language_name%5b%5d=r&language_name%5b%5d=ruby&language_name%5b%5d=rust&language_name%5b%5d=scala&language_name%5b%5d=-1&commit=Update)



What programmers currently learn (https://research.hackerrank.com/developer-skills/2018/)

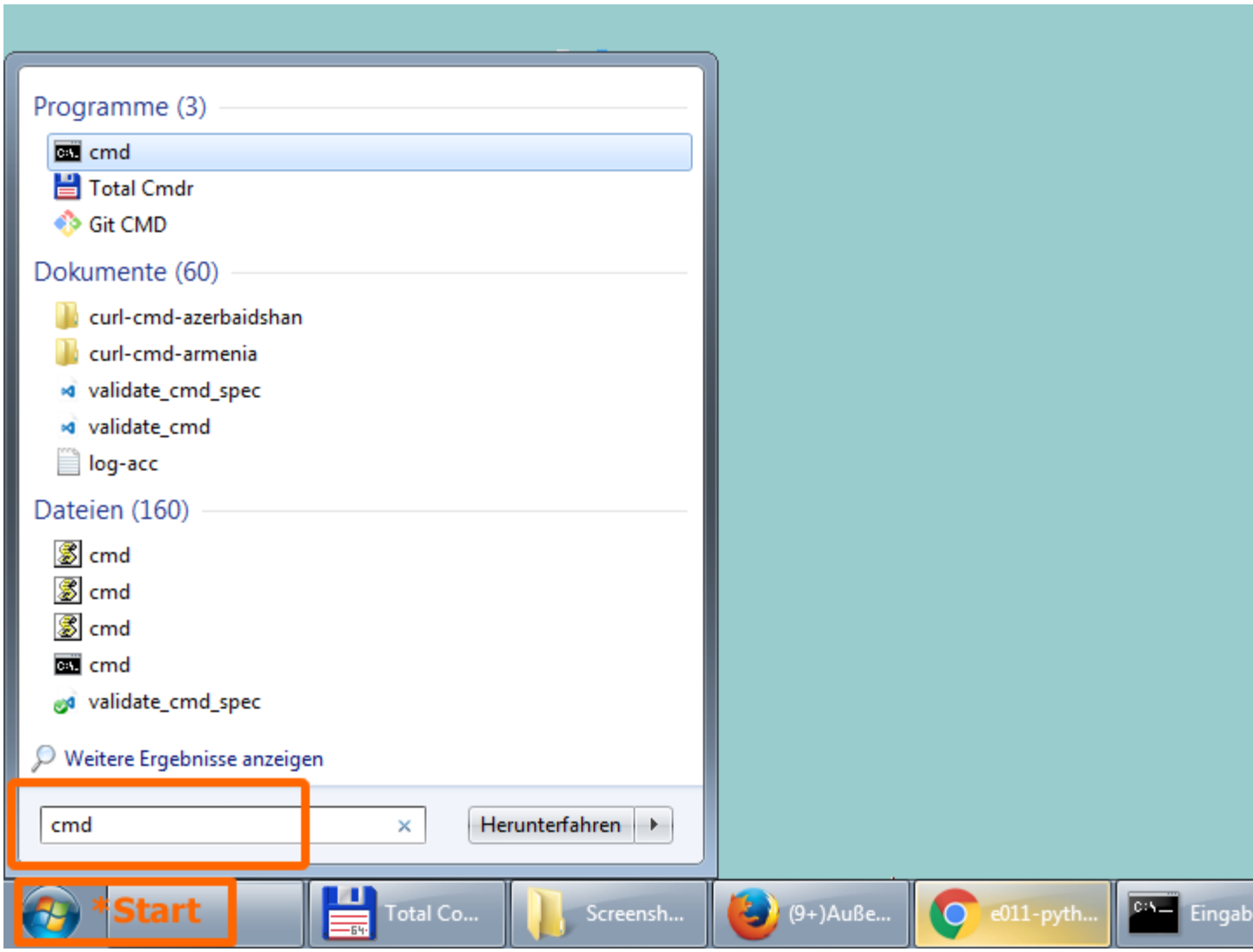
Trend towards fewer keywords (https://stackoverflow.com/questions/4980766/reserved-keywords-count-by-programming-language)

Python keywords (https://www.programiz.com/python-programming/keyword-list)

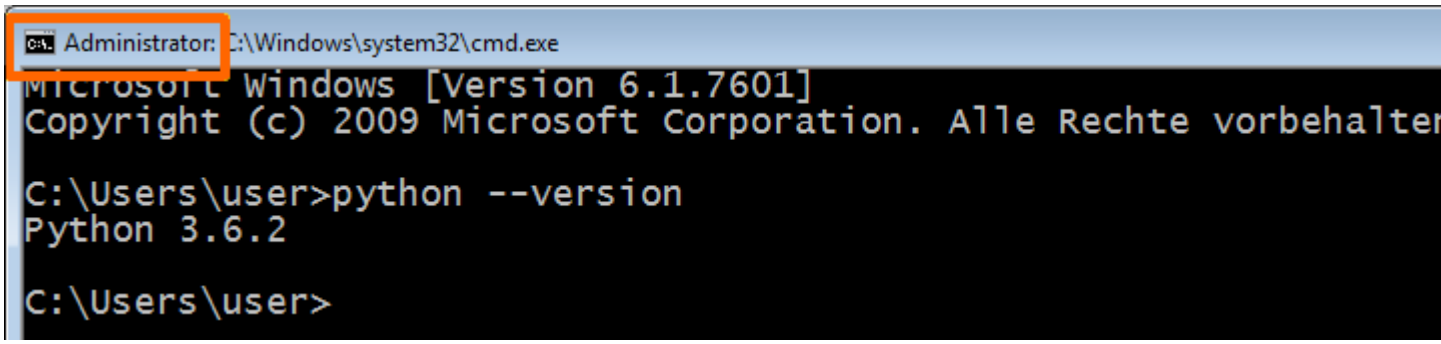
Silent Python install for central software distribution

This *alternative* to explicitly installing Python (above).

- Copy the downloaded program to `c\:temp`
  (create the directory if necessary)
- Click Start.
- In the Start Search box, type `cmd`



- Then press `CTRL+SHIFT+ENTER` (`STRG+HOCHTASTE+ENTER`)
  to start `cmd` program as `administrator`



Paste the following setup command into the command window. Use the menu as shown in picture below.

```
c:\Temp\python-3.6.4-amd64.exe  ^
  /quiet InstallAllUsers=1 PrependPath=1 Shortcuts=0  ^
  AssociateFiles=0  ^
  Include_launcher=0  InstallLauncherAllUsers=0  ^
```

```
SimpleInstallDescription="ZEW Zentral Installation Python 36"  ^
TargetDir=c:\Python3
```