

Python, kind of a tutorial

This notebook will be available here :

<https://wiki.student.info.ucl.ac.be/Documentation/Python>
(<https://wiki.student.info.ucl.ac.be/Documentation/Python>)

... and on Moodle

What is Python ?

- high-level, general purpose, object-oriented, interactive programming language
- batteries included : **huge** standard library
- **dynamically Interpreted**

python code (.py) -> bytecode (.pyc) -> execution

Some useful links

- Python language
 - Python website: <http://www.python.org/> (<http://www.python.org/>)
 - Python documentation: <https://docs.python.org/3/> (<https://docs.python.org/3/>)
 - Python tutorial: <https://docs.python.org/3/tutorial/> (<https://docs.python.org/3/tutorial/>)
- Some tools
 - Python interpreter (build in the Python distribution)
 - IDLE: <https://docs.python.org/3/library/idle.html> (<https://docs.python.org/3/library/idle.html>)
 - iPython and iPython notebook: <http://ipython.org/> (<http://ipython.org/>)
 - Pycharm: <http://www.jetbrains.com/pycharm/> (<http://www.jetbrains.com/pycharm/>)
 - Eclipse/Pydev: <http://www.pydev.org/> (<http://www.pydev.org/>)

Writing a Python program/script

- use the .py file extension

```
In [1]: #!/usr/bin/env python3
# coding: utf-8

print("Hello world!")
print("Hello AI students!")
print("test testing test")
```

```
Hello world!
Hello AI students!
test testing test
```

- allow execution:

```
chmod +x myfile.py
```

- execute:

```
./myfile.py
```

```
python3 myfile.py
```

Grab-01.py

The Grab project : build a **web indexer**

- capable of
 - analysing a web page to find and count words (indexer)
 - analysing a web page to find external links (crawler)
 - storing all the data into a adapted structure

```
In [ ]: #!/usr/bin/env python3
# vim: set fileencoding=utf-8 :
#
# Grab.py - version 01 - check args

# 1) args management
URLPREF = "http://"

import sys
def usage():
    print("usage:", sys.argv[0], "<url to parse>")
    sys.exit(1)

if not (len(sys.argv) == 2 and sys.argv[1].startswith(URLPREF)) :
    usage()

url = sys.argv[1]
print("You ask to grab", url)
```

Variables, basic data types and strings

```
In [2]: # defining variables <- this is a comment
```

```
a = 1  
b = 2  
c = 'string'
```

```
print("a+b=", a + b, "\na-b=", a - b)  
print("a*b=", a * b, "\na/b=", a / b, "\nb^b=", b ** b)
```

```
a+b= 3  
a-b= -1  
a*b= 2  
a/b= 0.5  
b^b= 4
```

```
In [3]: d = 0.1
```

```
e = 3.4 + 2j  
print("a is a", type(a), "\nb is a", type(b), "\nc is a", type(c))  
print("d is a", type(d), "\ne is a", type(e))
```

```
a is a <class 'int'>  
b is a <class 'int'>  
c is a <class 'str'>  
d is a <class 'float'>  
e is a <class 'complex'>
```

```
In [4]: # variables have no fixed type  
b = 1  
print(b)  
b = "another string"  
print(b)
```

```
1  
another string
```

Playing with strings

```
In [5]: # string are iterable
print(b)
print(b[0])
print(b[1])
print("Positive indexes", b[0], b[1], b[2])
print("Negative indexes", b[-1], b[-2], b[-3])
print(len(b))
```

```
another string
a
n
Positive indexes a n o
Negative indexes g n i
14
```

```
In [6]: # take a slice of string
print(b)
print(b[0:2])      # first 2 chars
print(b[0:len(b)-1]) # all but the last
print(b[0:-1])      # all but the last (2nd version)
print(b[:-1])       # all but the last (3rd version)
print(b[1:])        # all but the first
print(b[:])         # a copy of b
```

```
another string
an
another strin
another strin
another strin
nother string
another string
```

```
In [7]: # strings are *unmutable*
b[1] = "c"
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-7-a7f3a5381ba4> in <module>()
      1 # strings are *unmutable*
----> 2 b[1] = "c"

TypeError: 'str' object does not support item assignment
```

```
In [8]: # playing with string : + and *
b = "Spam"
print("b+b=", b + b, "\n" + "b*3=", b*3)
print("Egg, " + (b + ", ") * 4 + b + " and Egg")
```

```
b+b= SpamSpam
b*3= SpamSpamSpam
Egg, Spam, Spam, Spam, Spam, Spam and Egg
```

Inspection and builtin help system

What are string methods ?

In [11]:

```
print(dir(b))
help(b.startswith)

['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
 '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__',
 '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
 '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum',
 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable',
 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase',
 'title', 'translate', 'upper', 'zfill']  
Help on built-in function startswith:  
  
startswith(...) method of builtins.str instance  
S.startswith(prefix[, start[, end]]) -> bool
```

Return True if S starts with the specified prefix, False otherwise.
With optional start, test S beginning at that position.
With optional end, stop comparing S at that position.
prefix can also be a tuple of strings to try.

Some string methods

```
In [19]: # calling an instance method
b = "Spam"
print(b, b.startswith("Sp"))

b = "Egg"
print(b, b.startswith("Sp"))
```

```
Spam True
Egg False
```

```
In [20]: spacious = "      " + b + " \n\n      "
spacious.strip()
```

```
Out[20]: 'Egg'
```

```
In [21]: b = "Spam"  
       b.split('a')
```

```
Out[21]: ['Sp', 'm']
```

```
In [22]: b.upper()
```

```
Out[22]: 'SPAM'
```

```
In [23]: b.lower()
```

```
Out[23]: 'spam'
```

Using modules

In [25]: `dir(sys)`

Out[25]: ['`__displayhook__`',
 '`__doc__`',
 '`__excepthook__`',
 '`__interactivehook__`',
 '`__loader__`',
 '`__name__`',
 '`__package__`',
 '`__spec__`',
 '`__stderr__`',
 '`__stdin__`',
 '`__stdout__`',
 '`clear_type_cache`',
 '`current_frames`',
 '`debugmallocstats`',
 '`getframe`',
 '`home`',
 '`mercurial`',
 '`xoptions`',
 '`abiflags`',
 '`api_version`',
 '`argv`',
 '`base_exec_prefix`',
 '`base_prefix`',
 '`builtin_module_names`',
 '`byteorder`',
 '`call_tracing`',
 '`callstats`',
 '`copyright`',
 '`displayhook`',
 '`dont_write_bytecode`',
 '`exc_info`',
 '`excepthook`',
 '`exec_prefix`',
 '`executable`',
 '`exit`',

```
In [26]: type(sys.argv)
```

```
Out[26]: list
```

Lists are mutable and iterable

```
In [27]: l = [1, 2, "a", 3]
ll = [4.5, l, 6.7]
print(l)
print(ll)
print(l[0])
print(l[-1])
print(ll[1])
```

```
[1, 2, 'a', 3]
[4.5, [1, 2, 'a', 3], 6.7]
1
3
[1, 2, 'a', 3]
```

Take a slice of a list

```
In [28]: print("l \t= ", l)
      print("l[0:2] \t= ", l[0:2]) # first 2 chars
      print("l[:-1] \t= ", l[:-1]) # all but the last (2nd version)
      print("l[1:] \t= ", l[1:]) # all but the first
      print("l[:] \t= ", l[:]) # a copy of l
```

```
l      = [1, 2, 'a', 3]
l[0:2] = [1, 2]
l[:-1] = [1, 2, 'a']
l[1:]  = [2, 'a', 3]
l[:]   = [1, 2, 'a', 3]
```

A list does support item assignment

```
In [29]: print("l = ", l)
print("ll = ", ll)
l[0] = 42
l[0] += 1
print(l)
```

```
l = [1, 2, 'a', 3]
ll = [4.5, [1, 2, 'a', 3], 6.7]
[43, 2, 'a', 3]
```

```
In [30]: print("l = ", l)
print("ll = ", ll)
ll[1][0] = 0.0 # double index
print(ll)
print(l)
```

```
l =  [43, 2, 'a', 3]
ll =  [4.5, [43, 2, 'a', 3], 6.7]
[4.5, [0.0, 2, 'a', 3], 6.7]
[0.0, 2, 'a', 3]
```

```
In [31]: print("l = ", l)
# even slice assignment
l[0:2] = ["A", "B", "C"]
print(l)
```

```
l = [0.0, 2, 'a', 3]
['A', 'B', 'C', 'a', 3]
```

Some list methods

```
In [32]: # append at the end  
l.append("new element")  
l
```

```
Out[32]: ['A', 'B', 'C', 'a', 3, 'new element']
```

```
In [33]: # pop at index (default -1)  
l.pop(1)
```

```
Out[33]: 'B'
```

```
In [34]: l
```

```
Out[34]: ['A', 'C', 'a', 3, 'new element']
```

```
In [35]: # insert at index
l.insert(1, "another element")
print(l)
dir(l)
```

```
['A', 'another element', 'C', 'a', 3, 'new element']
```

```
Out[35]: ['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__']
```

Control flow : if statement

/!\ Python blocs are delimited by indentation /!

```
In [36]: b = "Holy"
if b == "Spam": # line ending with : -> requires an indented block
    pass          # do nothing, in block statement
    print("OK")   # next in block statement
elif b == "Egg":
    print("KO")
elif b == "Holy":
    print("The", b, "Grail")
else:
    print("wtf?")

print("Out of the IF") # out of block statement
```

```
The Holy Grail
Out of the IF
```

```
In [37]: # boolean operators : and, or, not
b = "Spam"
if b.startswith("Sp") and b.endswith("am"):
    print("Spam" + b)
```

SpamSpam

Control flow : function definition (simplest form)

```
In [38]: def myfunction(arg1, arg2, arg3): # line ending with : -> requires an indented
          block
    """ this is a docstring, meant
          to give information on my function """
    print("This is myfunction")
    print("Args:", arg1, arg2, arg3)
```

```
In [39]: # call myfunction  
ret = myfunction(1, 2, 3)
```

```
This is myfunction  
Args: 1 2 3
```

```
In [40]: print(ret)
```

```
None
```

```
In [41]: help(myfunction)

Help on function myfunction in module __main__:

myfunction(arg1, arg2, arg3)
    this is a docstring, meant
    to give information on my function
```

```
In [42]: def my2ndfunction(arg1, arg2, arg3): # : => indented block
    """ this is a docstring """
    print("This is my2ndfunction" +
          "test")
    return "Args: " + str(arg1) + ", " \
           + str(arg2) + ", " + str(arg3)
```

```
In [43]: ret = my2ndfunction(1, 2, 3)
```

```
This is my2ndfunctiontest
```

```
In [44]: print(type(ret))
print("ret = ", ret)
```

```
<class 'str'>
ret = Args: 1, 2, 3
```

Grab-02.py

```
In [ ]: #!/usr/bin/env python3
# vim: set fileencoding=utf-8 :
#
# Grab.py - version 02 - grab an URL

# 1) args management
import sys
URLPREF = "http://"
def usage():
    print("usage: ", sys.argv[0], "<url to parse>")
    sys.exit(1)

if not (len(sys.argv) == 2 and sys.argv[1].startswith(URLPREF)) :
    usage()

url = sys.argv[1]
print("You ask to grab", url)

# 2) retreive the URL
import urllib.request

data = urllib.request.urlopen(url)
print("Here it is:")

for line in data:
    print(line.decode("utf-8"), end="")
```

Control flow : for statement

```
In [45]: string = "abcdef"

for letter in string: # line ending with : -> indented block
    print(letter, end=" ")

a b c d e f
```

```
In [46]: alist = ["Lancelot", "Galaad", "Bedivere", "Arthur", "Robin"]

for knight in alist:      # line ending with : -> indented block
    title = "Sir"
    if knight == "Arthur": # line ending with : -> indented block
        title = "King"
    print(title, knight)
```

```
Sir Lancelot
Sir Galaad
Sir Bedivere
King Arthur
Sir Robin
```

```
In [47]: list(enumerate(alist))
```

```
Out[47]: [(0, 'Lancelot'), (1, 'Galaad'), (2, 'Bedivere'), (3, 'Arthur'), (4, 'Robin')]
```

```
In [48]: for i, knight in enumerate(alist):
            if i == len(alist) - 1:
                print("And the last: ", end="")
                print(i, knight)
```

```
0 Lancelot
1 Galaad
2 Bedivere
3 Arthur
And the last: 4 Robin
```

Using range

```
In [49]: list(range(5))
```

```
Out[49]: [0, 1, 2, 3, 4]
```

```
In [50]: for i in range(5):  
         print(i)
```

```
0  
1  
2  
3  
4
```

```
In [51]: a = ['Mary', 'had', 'a', 'little', 'lamb']

for i in range(len(a)): # Don't do this, this is NOT pythonic! Use enumerate if you want this output
    print(i, a[i])
```

```
0 Mary
1 had
2 a
3 little
4 lamb
```

```
In [52]: range(5)    # range returns an *iterator*
```

```
Out[52]: range(0, 5)
```

```
In [53]: list(range(3, 10))
```

```
Out[53]: [3, 4, 5, 6, 7, 8, 9]
```

```
In [54]: list(range(2, 10, 2))
```

```
Out[54]: [2, 4, 6, 8]
```

```
In [55]: list(range(-10, -100, -30))
```

```
Out[55]: [-10, -40, -70]
```

File objects

```
In [57]: lines = ["First line", "Second line", "Last line"]

for line in lines:
    f.write(line + "\n")

f.close()
```

```
In [58]: import os.path  
os.path.isfile(filename)
```

```
Out[58]: True
```

```
In [59]: ff = open(filename, "r")
for line in ff:
    print(line, end=' ')
ff.close()
```

```
First line
Second line
Last line
```

```
In [60]: ff = open(filename, "r")
print(ff.read())
ff.close()
```

```
First line
Second line
Last line
```

```
In [61]: ff = open(filename, "r")
ff.readline()
```

```
Out[61]: 'First line\n'
```

```
In [62]: ff.readlines()
```

```
Out[62]: ['Second line\n', 'Last line\n']
```

```
In [63]: ff.close()
```

urllib module

```
In [65]: import urllib.request  
f = urllib.request.urlopen("http://localhost:8000") # grab the content, return  
file object
```

```
In [66]: f.readline()
```

```
Out[66]: b'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w  
3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">\n'
```

```
In [67]: f.readline()
```

```
Out[67]: b'\n'
```

```
In [68]: # this is a byte array  
f.readline().decode("utf-8")    # to produce usual string
```

```
Out[68]: '\n'
```

```
In [69]: for line in f:  
    print(line.decode("utf-8"), end="")  
  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">  
  
<head>  
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />  
    <title>Python Documentation Index</title>  
    <meta name="keywords" content="None" />  
    <meta name="description" content="None" />  
    <link rel="alternate" type="application/rss+xml" title="Community Events"  
          href="http://www.python.org/channews.rdf" />  
    <link rel="alternate" type="application/rss+xml" title="Python Recipes"  
          href="http://aspn.activestate.com/ASPN/Cookbook/Python/index_rss" />  
    <link rel="alternate" type="application/rss+xml" title="Usergroup News"  
          href="http://python-groups.blogspot.com/feeds/posts/default" />  
    <link rel="alternate" type="application/rss+xml" title="Python Screencasts"  
          href="http://www.showmedo.com/latestVideoFeed/rss2.0?tag=python" />  
    <link rel="alternate" type="application/rss+xml" title="Python Podcasts"  
          href="http://www.awaretek.com/python/index.xml" />  
    <link rel="alternate" type="application/rss+xml" title="Foundation News"  
          href="http://pyfound.blogspot.com/feeds/posts/default" />  
    <link rel="alternate" type="application/rss+xml" title="Python Enhancement P  
roposals"  
          href="http://www.python.org/dev/peps/peps.rss" />  
    <link rel="stylesheet" type="text/css" media="screen" id="screen-switcher-st  
ylesheet"  
          href="/styles/screen-switcher-default.css" />  
    <link rel="stylesheet" type="text/css" media="sc&#82;een"  
          href="/styles/netscape4.css" />  
    <link rel="stylesheet" type="text/css" media="print"  
          href="/styles/print.css" />  
    <link rel="alternate stylesheet" type="text/css" media="screen"  
          href="/styles/largestyles.css" title="large text" />  
    <link rel="alternate stylesheet" type="text/css" media="screen"  
          href="/styles/defaultfonts.css" title="default fonts" />
```

Grab-03.py

```
In [ ]: #!/usr/bin/env python3
# vim: set fileencoding=utf-8 :
#
# Grab.py - version 03 - grab an URL, isolates words

# 1) args management
import sys
URLPREF = "http://"
def usage():
    print("usage: ", sys.argv[0], "<url to parse>")
    sys.exit(1)

if not (len(sys.argv) == 2 and sys.argv[1].startswith(URLPREF)) :
    usage()

url = sys.argv[1]
print("You ask to grab", url)

# 2) retreive the URL
import urllib.request

raw_data = urllib.request.urlopen(url)

# 3) Parse it
import re
data = raw_data.read().decode("utf-8")
print(data)
print("Raw data\n" + "="*80)
input()

# 3.1) remove HTML header
data = re.sub(r"<head>.*(\n)*.*</head>", "", data, flags=re.MULTILINE)
print(data)
print("No HTML Header\n" + "="*80)
input()
```

Re module

```
In [70]: import re  
re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
```

```
Out[70]: ['foot', 'fell', 'fastest']
```

```
In [71]: re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
```

```
Out[71]: 'cat in the hat'
```

```
In [72]: re.split("(b|e)", "abcdef")
```

```
Out[72]: ['a', 'b', 'cd', 'e', 'f']
```

Match (at the beginning) and search (anywhere)

```
In [73]: re.match("c", "abcdef") # No match
```

```
In [74]: re.search("c", "abcdef")
```

```
Out[74]: <_sre.SRE_Match object; span=(2, 3), match='c'>
```

Lambda forms

```
In [75]: f = lambda x: x+42 # lambda forms are unnamed functions  
f
```

```
Out[75]: <function __main__.<lambda>>
```

```
In [76]: f(5)
```

```
Out[76]: 47
```

```
In [77]: def make_incremator(inc): # a function that returns functions!
          return lambda x: x + inc

f = make_incremator(42)
f
```

```
Out[77]: <function __main__.make_incremator.<locals>.<lambda>>
```

```
In [78]: f(5)
```

```
Out[78]: 47
```

```
In [79]: ff = make_incremator(5)  
ff(5)
```

```
Out[79]: 10
```

```
In [80]: ff = f    # ff is now an alias of f  
ff(5)
```

```
Out[80]: 47
```

Functional tool: filter

`filter(function, sequence)` apply a filter to a list i.e. returns the list of elements of `sequence` for which the function `filter` return `True`.

```
In [81]: def is_even(x):
    return x % 2 == 0

print(list(range(10)))
print(list(filter(is_even, range(10))))
```



```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 2, 4, 6, 8]
```

```
In [82]: list(filter(lambda x: x%2 != 0, range(10)))
```

```
Out[82]: [1, 3, 5, 7, 9]
```

Functionnal tool: map

`map(function, sequence)` applies `function(item)` for each `item` in the `sequence` and returns a list of the return values.

```
In [83]: def cube(x): return x*x*x  
list(map(cube, range(1, 11)))
```

```
Out[83]: [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

```
In [84]: list(map(lambda x: x**3, range(1, 11)))
```

```
Out[84]: [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

```
In [85]: seq = range(8)

def add(x, y): return x+y

list(map(add, seq, seq))
```

```
Out[85]: [0, 2, 4, 6, 8, 10, 12, 14]
```

```
In [86]: list(map(lambda x, y: x + y, seq, seq))
```

```
Out[86]: [0, 2, 4, 6, 8, 10, 12, 14]
```

Sets

```
In [87]: basket = ["apple", "orange", "apple", "pear", "orange", "banana"]
```

```
In [88]: fruit = set(basket)    # create a set without duplicates  
fruit
```

```
Out[88]: {'apple', 'banana', 'orange', 'pear'}
```

```
In [89]: "orange" in fruit
```

```
Out[89]: True
```

```
In [90]: "salad" in fruit
```

```
Out[90]: False
```

```
In [91]: a = set("abracadabra")
b = set("alacazam")
print("a \tunique letters in abracadabra: \t", a)
print("a - b \tletters in a but not in b: \t", a - b)
print("a | b \tletters in either a or b: \t", a | b)
print("a & b \tletters in a and b: \t\t", a & b)
print("a ^ b \tletters in a or b but not both: ", a ^ b)

a      unique letters in abracadabra:  {'c', 'd', 'a', 'b', 'r'}
a - b    letters in a but not in b:    {'d', 'b', 'r'}
a | b    letters in either a or b:    {'m', 'a', 'c', 'd', 'b', 'r', 'z',
'l'}
a & b    letters in a and b:          {'c', 'a'}
a ^ b    letters in a or b but not both:  {'m', 'd', 'b', 'r', 'z', 'l'}
```

Grab-04.py

```
In [ ]: #!/usr/bin/env python3
# vim: set fileencoding=utf-8 :

# Grab.py - version 04
#   - grab an URL
#   - isolate words, count occurrences
#   - store everything into a dict

import sys
import urllib.request
import re

# data
URLPREF = "http://"
MIN_WORD_LENGTH = 3

def getWords(data):
    """return the list of tuples with
       - words contained within a HTML string
       - number of occurrences of this word within the string"""

    # remove HTML header
    data = re.sub(r"<head>.*(.*\n)*.*</head>", "", data, flags=re.MULTILINE)
    # remove HTML tags
    data = re.sub(r"<[^>]*>", "", data, flags=re.MULTILINE)
    # remove HTML entities
    data = re.sub(r"&[^ ;]*;", "", data)
    # remove special chars
    data = re.sub(r"[,.;:!?\\"'()\\{}%$@#/+*`-]", " ", data)
    # isolate words
    words = data.split()
    # remove too short words and any word containing number
    words = filter(lambda x: len(x) > 3, words)
    words = list(filter(lambda x: not re.search("[0-9]", x), words))
    # lower case
    words = list(map(lambda x: x.lower(), words))
    # sort it and count occurrences
```

A word about the truth

```
In [92]: if [] or "" or {} or () or 0 or 0.0 or 0.0 + 0j or None:  
    print("Some are True")  
else:  
    print("All are False")
```

```
All are False
```

```
In [93]: if 1:  
    print("1 is True")
```

```
1 is True
```

Tuples: immutable list

```
In [94]: t = (12345, 54321, 'hello!')  
t[0]
```

```
Out[94]: 12345
```

```
In [95]: t
```

```
Out[95]: (12345, 54321, 'hello!')
```

```
In [96]: # Tuples may be nested:  
u = (t, (1, 2, 3, 4, 5))  
  
u
```

```
Out[96]: ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

```
In [97]: u[0] = 4
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-97-3803e895a7da> in <module>()
----> 1 u[0] = 4
```

```
TypeError: 'tuple' object does not support item assignment
```

```
In [98]: # Beware of mutable inside immutable!
v = (1, [2, 3], 4)
```

```
v
```

```
Out[98]: (1, [2, 3], 4)
```

```
In [99]: v[1] = range(5)
```

```
-----
TypeError                                     Traceback (most recent call last)
<ipython-input-99-ecf75485165d> in <module>()
      1 v[1] = range(5)
-----
```

```
TypeError: 'tuple' object does not support item assignment
```

```
In [100]: v[1][0] = "mutables can be modified!"
```

```
v
```

```
Out[100]: (1, ['mutables can be modified!', 3], 4)
```

```
In [101]: len(v)
```

```
Out[101]: 3
```

```
In [102]: for elt in v:  
          print(elt)
```

```
1  
['mutables can be modified!', 3]  
4
```

```
In [103]: empty = ()  
singleton = "hello", # <-- note the trailing comma
```

```
In [104]: len(empty)
```

```
Out[104]: 0
```

```
In [105]: len(singleton)
```

```
Out[105]: 1
```

```
In [106]: singleton
```

```
Out[106]: ('hello',)
```

```
In [107]: ('hello') # <-- without trailing comma
```

```
Out[107]: 'hello'
```

Dictionaries: associative tables

```
In [108]: tel = {'jack': 4098, 'sape': 4139}  
          tel['guido'] = 4127  
          tel
```

```
Out[108]: {'guido': 4127, 'jack': 4098, 'sape': 4139}
```

```
In [109]: tel['jack']
```

```
Out[109]: 4098
```

```
In [110]: del tel['sape']
          tel['irv'] = 4127
          tel
```

```
Out[110]: {'guido': 4127, 'irv': 4127, 'jack': 4098}
```

```
In [111]: list(tel.keys())
```

```
Out[111]: ['irv', 'jack', 'guido']
```

```
In [112]: 'guido' in tel
```

```
Out[112]: True
```

Building a **dict** from a **list**

```
In [113]: dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
```

```
Out[113]: {'guido': 4127, 'jack': 4098, 'sape': 4139}
```

```
In [114]: help(__builtin__.dict)
```

Help on class dict in module builtins:

```
class dict(object)
    dict() -> new empty dictionary
    dict(mapping) -> new dictionary initialized from a mapping object's
        (key, value) pairs
    dict(iterable) -> new dictionary initialized as if via:
        d = {}
        for k, v in iterable:
            d[k] = v
    dict(**kwargs) -> new dictionary initialized with the name=value pairs
        in the keyword argument list.  For example:  dict(one=1, two=2)
```

Methods defined here:

```
__contains__(self, key, /)
    True if D has a key k, else False.
```

```
__delitem__(self, key, /)
    Delete self[key].
```

```
__eq__(self, value, /)
    Return self==value.
```

```
__ge__(self, value, /)
    Return self>=value.
```

```
__getattribute__(self, name, /)
    Return getattr(self, name).
```

```
__getitem__(...)
    x.__getitem__(y) <==> x[y]
```

```
__gt__(self, value, /)
    Return self>value.
```

```
In [115]: knights = {'gallahad': 'the pure', 'robin': 'the brave'}
```

```
for k, v in knights.items():    # order NOT fixed! BEWARE !!!
    print("Sir", k[0].upper() + k[1:], v)
```

```
Sir Gallahad the pure
Sir Robin the brave
```

Grab-05.py

```
In [ ]: #!/usr/bin/env python3
# vim: set fileencoding=utf-8 :

# Grab.py - version 05
#   - grab an URL
#   - isolate words, count occurencies
#   - store everything into a dict
#   - crawl from the URL and repeat the treatment on linked pages

import sys
import urllib.request
import re

# data
URLPREF = "http://"
MIN_WORD_LENGTH = 3

storage = {}

# tools
def getwords(data):
    """return the list of tuples with
       - words contained within a HTML string
       - number of occurencies of this word within the string"""
    data = re.sub(r"<head>.*(\.\n)*.*</head>", "", data, flags=re.MULTILINE)
    data = re.sub(r"<[^>]*>", "", data, flags=re.MULTILINE)
    data = re.sub(r"&[^ ;]*;", "", data)
    data = re.sub(r"[,.;:!?'()\\{}%$€@#/+*`-]", " ", data)
    words = data.split()

    ### List comprehension to do 3 steps in a single one ####
    words = [w.lower() for w in words if len(w) > 3 and not re.search("[0-9]", w)]
    #
    # This has the same effect as
    ## words = filter(lambda x: len(x) > 3, words)
    ## words = list(filter(lambda x: not re.search("[0-9]", x), words))
```

List comprehensions

```
In [116]: freshfruit = [' banana', ' loganberry \n', 'passion fruit      ']  
          [weapon.strip() for weapon in freshfruit]  
  
Out[116]: ['banana', 'loganberry', 'passion fruit']
```

Syntax:

[value with a variable **for** variable **in** an iterable]

```
In [117]: vec = [2, 4, 6]
          [3*x for x in vec]
```

```
Out[117]: [6, 12, 18]
```

```
In [118]: [3*x for x in vec if x > 3] # add conditions
```

```
Out[118]: [12, 18]
```

```
In [119]: [3*x for x in vec if x < 2]
```

```
Out[119]: []
```

```
In [120]:
```

```
[[x,x**2] for x in vec] # build (potentially) complex objects
```

```
Out[120]: [[2, 4], [4, 16], [6, 36]]
```

With more than one variable too!

```
In [121]: vec1 = [2, 4, 6]
vec2 = [4, 3, -9]

[x*y for x in vec1 for y in vec2] # combine all possible pairs
```

Out[121]: [8, 6, -18, 16, 12, -36, 24, 18, -54]

```
In [122]: [vec1[i]*vec2[i] for i in range(len(vec1))] # explicit association by index
```

```
Out[122]: [8, 12, -54]
```

```
In [123]: [v1*v2 for v1, v2 in zip(vec1, vec2)] # zip does that
```

```
Out[123]: [8, 12, -54]
```

Sorting

```
In [124]: sorted([5, 2, 3, 1, 4]) # sorted returns the sorted object
```

```
Out[124]: [1, 2, 3, 4, 5]
```

```
In [125]: print( [5, 2, 3, 1, 4].sort() ) # sort method sort in place, returns nothing
```

```
None
```

```
In [126]: l = [5, 2, 3, 1, 4]
l.sort()
l
```

```
Out[126]: [1, 2, 3, 4, 5]
```

```
In [127]: sorted("copyright")      # works with ANY iterable
```

```
Out[127]: ['c', 'g', 'h', 'i', 'o', 'p', 'r', 't', 'y']
```

The **key** parameter

```
In [128]: sorted("This is a test string from Andrew".split())
```

```
Out[128]: ['Andrew', 'This', 'a', 'from', 'is', 'string', 'test']
```

```
In [129]: sorted("This is a test string from Andrew".split(), key=str.lower)
```

```
Out[129]: ['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']
```

```
In [130]: student_tuples = [ ('john', 'A', 15), # (name, grade, age)
                           ('jane', 'B', 12),
                           ('dave', 'B', 10) ]

# sort by age
sorted(student_tuples, key=lambda student: student[2])
```

```
Out[130]: [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

The **reverse** parameter

```
In [131]: sorted(student_tuples, key=lambda student: student[2], reverse=True)
```

```
Out[131]: [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
```

That's all folks !

With all this in mind, you can understand the Grab example.