Python for scientists

Lesson 6

Built-in functions.

Reading and writing files

```
def complementary(seq):
    nt_comp = {
        'A' : 'T',
        'C' : 'G',
        'G' : 'C',
        'T' : 'A',
    }
    for nt in seq:
        compseq += nt_comp[nt]
    return compseq
```

Alvaro Sebastián Yagüe

www.sixthresearcher.com

@SixthResearcher

Built-in functions

Built-in functions

Built-in functions in Python are special functions that are always available to use, they are included in the original Python code. Their names should be avoided to be used in variables.

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

https://docs.python.org/3/library/functions.html

From Lesson 2 …

len(A)	Returns the length or number of items.
type(A)	Returns the data type stored in A.
str(A)	Produces a printable string representation of a list or dictionary.
list(A)	Produces a list from a string or dict keys.
min(A)	Returns the minimum value from A (if is a dict, from keys).
max(A)	Returns the maximum value from A (if is a dict, from keys).

A and **B** can be either a string, list or dictionary.

And from Lesson 4 …

integer = int(<string>)
float = float(<string>)
string = str(<number>)

- **Function 'int()' converts a string into an integer** (number without decimals).
- **Function 'float()' converts a string into a floating point number** (equivalent to a decimal for most of the cases).
- **Function 'str()' convert s a number into a string.**

Let's remember from Lesson 1 …

```
>>> 'First line.\nSecond line.'
'First line.\nSecond line.'

>>> print('First line.\nSecond line.')
First line.
Second line.
```

print => Python function to print data

https://docs.python.org3/tutorial/introduction.html

Printing formatted data

Printing formatted data

We already know how to **print data** with the **function 'print'**.

Arguments are separated by commas. Strings can be joined by the plus operator (+).

```
>>> city = 'Poznan'
>>> country = 'Poland'
>>> population =  1100000
>>>
>>> print(city+" is a city from "+country+" and has "+population+" inhabitants")
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    print(city+" is a city from "+country+" and has "+population+" inhabitants")
TypeError: Can't convert 'int' object to str implicitly
>>>
>>> # All arguments joined by '+' must be strings
>>> print(city+" is a city from "+country+" and has "+str(population)+" inhabitants")
Poznan is a city from Poland and has 1100000 inhabitants
>>>
>>> # The recommended way to use print is like this:
>>> print(city,"is a city from",country,"and has",population,"inhabitants")
Poznan is a city from Poland and has 1100000 inhabitants
```

Printing formatted data

print() takes as arguments several objects and converts them to strings for printing.

Arguments are given separated by commas and, by default, they are printed separated by one space and one line break is added at the end, but it can be changed.

print([objects], sep=' ', end='\n')

```
>>> print(city,"is a city from",country,"and has",population,"inhabitants")
Poznan is a city from Poland and has 1100000 inhabitants
>>>
>>> # We can change the separator between arguments
>>> print(city,"is a city from",country,"and has",population,"inhabitants", sep="")
Poznanis a city fromPolandand has1100000inhabitants
>>> print(city,"is a city from",country,"and has",population,"inhabitants", sep="\n")
Poznan
is a city from
Poland
and has
1100000
inhabitants
>>>
>>> # Also we can change the end of line
>>> print(city,"is a city from",country,"and has",population,"inhabitants",end="\n- Wikipedia data")
Poznan is a city from Poland and has 1100000 inhabitants
- Wikipedia data
```

Printing formatted data

A more elegant way for printing data is to write a unique string and **insert fields to be replaced by variables or data given at the end with the method 'format'**.

<string>**.format(**x)

```
>>> # Let's calculate and print the volumen of a cube
>>> side = 2.25
>>> # Old style printing
>>> print("The volume of a cube of side "+str(side)+" is "+str(side**3))
The volume of a cube of side 2.25 is 11.390625
>>> # Printing using the new string 'format' method
>>> print("The volume of a {} of side {} is {}".format("cube",side,side**3))
The volume of a cube of side 2.25 is 11.390625
>>> # We can specify the order of arguments
>>> print("Volume {2} corresponds to a {0} of side {1}".format("cube",side,side**3))
Volume 11.390625 corresponds to a cube of side 2.25
>>> # We can assign a keyword to each argument
>>> print("Volume {v} corresponds to a {f} of side {s}".format(f="cube",s=side,v=side**3))
Volume 11.390625 corresponds to a cube of side 2.25
```

Printing formatted data

Data types can be represented by symbols in the fields to be replaced by the 'format' method.

Type	Meaning
s	String (converts any python object using str()).
d	Signed integer decimal.
f	Floating point decimal format.
e	Floating point exponential format (lowercase).

```
>>> # We can give specific formats to the values (string, number with 2 decimal positions...)
>>> print("The volume of a {:s} of side {:.2f} is {:.4f}".format("cube",side,side**3))
The volume of a cube of side 2.25 is 11.3906
>>> print("The volume of a {:s} of side {:.2f} is {:e}".format("cube",side,side**3))
The volume of a cube of side 2.25 is 1.139062e+01
>>> # Format method can be also used to format strings before saving them into variables
>>> string_to_print = "The volume of a {:s} of side {:.2f} is {:.4f}".format("cube",side,side**3)
>>> print(string_to_print)
The volume of a cube of side 2.25 is 11.3906
```

Reading and writing files

Working with directories

The Python module 'os' allows to work with files and folders. Using its methods 'getcwd', 'makedirs' and 'chdir' we will know in which folder is Python working, we will create a new folder and we will move to this folder.

```
>>> # Imports a Python module to work with files and folders
>>> import os
>>>
>>> # Checks the Python current working directory
>>> print(os.getcwd())
C:\Program Files\Python 3
>>>
>>> # Creates a new directory
>>> os.makedirs('C:\Program Files\Python 3\lesson7')
>>>
>>> # Changes to this new directory
>>> os.chdir('C:\Program Files\Python 3\lesson7')
>>>
>>> # Checks the new working directory
>>> print(os.getcwd())
C:\Program Files\Python 3\lesson7
```

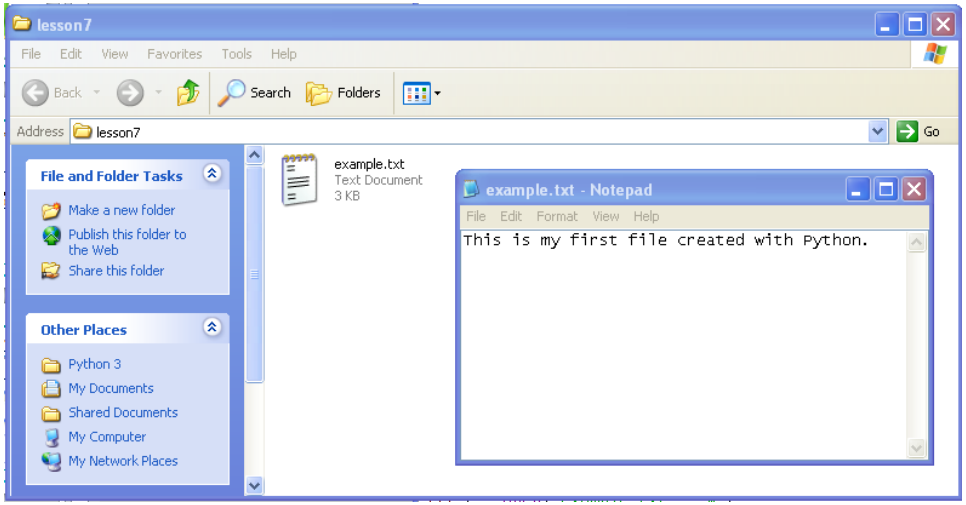
import os
os.getcwd()
os.makedirs(<path>)
os.chdir(<path>**)**

Creating files

Once that we are in the desired working directory, **we can easily create a file and write contents into it with the function 'open' and the method 'write'**.

```
>>> # Opens a file to write into
>>> f = open('example.txt', 'w')
>>>
>>> # Writes something into the file
>>> f.write("This is my first file created with Python.")
42
>>>
>>> # Closes and saves file contents
>>> f.close()
```

f = open(<path>**,'w')**
f.write(<str>**)**
f.close()



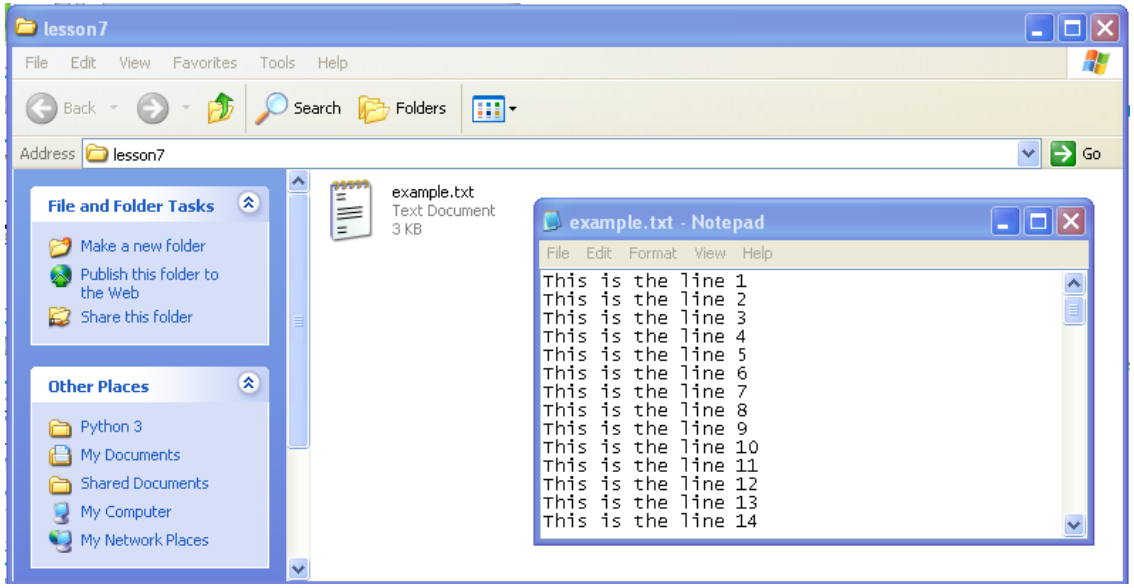
Creating files

Once that we are in the desired working directory, **we can easily create a file and write contents into it with the function 'open' and the method 'write'**.

```
>>> f = open('example.txt', 'w')
>>> for line in range(1,100):
        lout = f.write('This is the line '+str(line)+'\n')

>>> f.close()
```

f = open(<path>**,'w')**
f.write(<str>**)**
f.close()



Reading files

We can easily **read full file contents with the method 'read'**.

f = open(<path>,'r')
f.read(<size>)
f.close()

```
>>> f = open('example.txt', 'r')
>>> f.read()
'This is the line 1\nThis is the line 2\nThis is the line 3\nThis is the
line 5\nThis is the line 6\nThis is the line 7\nThis is the line 8\nThis
is the line 10\nThis is the line 11\nThis is the line 12\nThis is the line 13\nThis is the li
ne 14\nThis is the line 15\nThis is the line 16\nThis is the line 17\nThis is the line 18\nTh
is is the line 19\nThis is the line 20\nThis is the line 21\nThis is the line 22\nThis is the
line 23\nThis is the line 24\nThis is the line 25\nThis is the line 26\nThis is the line 27\n
This is the line 28\nThis is the line 29\nThis is the line 30\nThis is the line 31\nThis is t
he line 32\nThis is the line 33\nThis is the line 34\nThis is the line 35\nThis is the line 3
6\nThis is the line 37\nThis is the line 38\nThis is the line 39\nThis is the line 40\nThis i
s the line 41\nThis is the line 42\nThis is the line 43\nThis is the line 44\nThis is the lin
e 45\nThis is the line 46\nThis is the line 47\nThis is the line 48\nThis is the line 49\nThi
s is the line 50\nThis is the line 51\nThis is the line 52\nThis is the line 53\nThis is the
line 54\nThis is the line 55\nThis is the line 56\nThis is the line 57\nThis is the line 58\n
This is the line 59\nThis is the line 60\nThis is the line 61\nThis is the line 62\nThis is t
he line 63\nThis is the line 64\nThis is the line 65\nThis is the line 66\nThis is the line 6
7\nThis is the line 68\nThis is the line 69\nThis is the line 70\nThis is the line 71\nThis i
s the line 72\nThis is the line 73\nThis is the line 74\nThis is the line 75\nThis is the lin
e 76\nThis is the line 77\nThis is the line 78\nThis is the line 79\nThis is the line 80\nThi
s is the line 81\nThis is the line 82\nThis is the line 83\nThis is the line 84\nThis is the
line 85\nThis is the line 86\nThis is the line 87\nThis is the line 88\nThis is the line 89\n
This is the line 90\nThis is the line 91\nThis is the line 92\nThis is the line 93\nThis is t
he line 94\nThis is the line 95\nThis is the line 96\nThis is the line 97\nThis is the line 9
8\nThis is the line 99\n'
>>> f.close()
```

Remember always to close the file after reading or writing.

Reading files

Or **read file contents line by line with a for loop**.

```
>>> f = open('example.txt', 'r')
>>> for line in f:
        print(line, end='')

This is the line 1
This is the line 2
This is the line 3
This is the line 4
This is the line 5
This is the line 6
This is the line 7
This is the line 8
This is the line 9
This is the line 10
...
>>> f.close()
```

f = open(<path>**,'r')**
for line in f:
    <code>
f.close()

Remember always to close the file after reading or writing.

Reading files

There are many alternative ways to read a file: by lines, by size...

```
>>> f = open('example.txt', 'r')
>>>
>>> # Reads one line from the file
>>> f.readline()
'This is the line 1\n'
>>> f.readline()
'This is the line 2\n'
>>>
>>> # Reads part of the line
>>> f.readline(5)
'This '
>>> f.readline(5)
'is th'
>>> f.readline()
'e line 3\n'
>>>
>>> # Reads part of the file
>>> f.read(100)
'This is the line 4\nThis is the line 5\nThis is the line 6\n
This is the line 7\nThis is the line 8\nThis '
>>>
>>> f.close()
```

f = open(<path>**,'r')**
f.read(<size>**)**
f.readline(<size>**)**
f.close()

Exercise:
Counting bacterial genes

Exercise: Counting bacterial genes

Let's count how many protein coding genes has the *Escherichia coli* genome.

1. Download *E. coli* proteome in FASTA format:

   http://bioinfo.uncc.edu/zhx/binf8201/NC_000913.faa

   The '.faa' proteome file contains the sequences of the proteins coded by the *E. coli* genes.

2. Copy the file to 'C:\Program Files\Python 3\lesson7' folder.

3. Write a Python code that counts the number of lines that start with '>' character.

Python for scientists

Next lesson...

Python, Spyder, Jupyter and Anaconda

```
def complementary(seq):
    nt_comp = {
        'A' : 'T',
        'C' : 'G',
        'G' : 'C',
        'T' : 'A',
    }
    for nt in seq:
        compseq += nt_comp[nt]
    return compseq
```

Alvaro Sebastián Yagüe

www.sixthresearcher.com

@SixthResearcher