Introduction to Python course – part II

Silvia Salatino, PhD

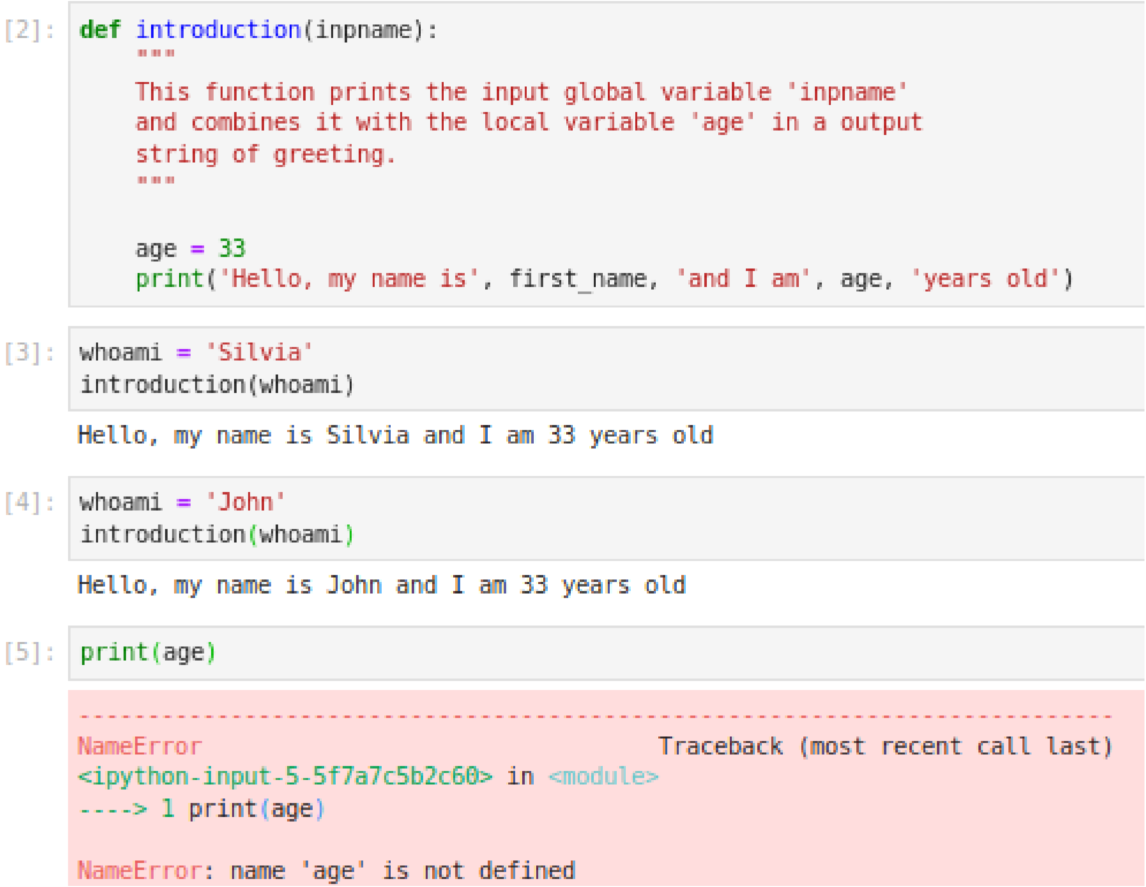
23.10.2019 – Wellcome Centre for Human Genetics, Oxford

Defining functions

Often you need to repeat the same operation multiple times with different input variables. Instead of writing the same code each time, you can use a ***function***, which is defined by the keyword “***def***”, followed by the function name and the input variables, enclosed in round brackets.

For sake of clarity, functions should be accompanied by a short but comprehensive description of what they do. This can be a comment on a single or multiple lines, preceded and closed by ***triple quotes***.

Variables declared outside of a function are called ***global*** and are anywhere in the code. Variables declared inside a function are called ***local*** and are only accessible to the current scope.



```
[2]: def introduction(inpname):
         """
         This function prints the input global variable 'inpname'
         and combines it with the local variable 'age' in a output
         string of greeting.
         """

         age = 33
         print('Hello, my name is', first_name, 'and I am', age, 'years old')
```

```
[3]: whoami = 'Silvia'
     introduction(whoami)

     Hello, my name is Silvia and I am 33 years old
```

```
[4]: whoami = 'John'
     introduction(whoami)

     Hello, my name is John and I am 33 years old
```

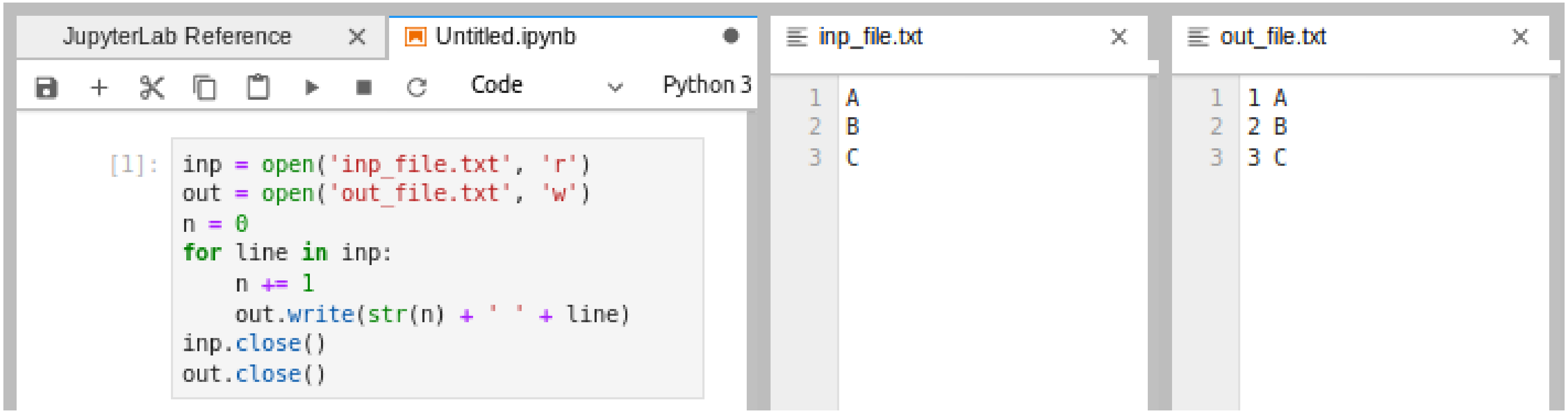
```
[5]: print(age)

     ---------------------------------------------------------------------------
     NameError                                 Traceback (most recent call last)
     <ipython-input-5-5f7a7c5b2c60> in <module>
     ----> 1 print(age)

     NameError: name 'age' is not defined
```

Reading and writing files

The ***open()*** built-in function takes as input a file name and a mode (‘***r***’ for reading, ‘***w***’ for writing, ‘***a***’ for appending to an existing file, and others...), and returns a file object. By default, files are opened in text mode (‘***t***’), but binary files could be opened too, using the ‘***b***’ mode. Once opened, a file must be closed with the ***close()*** function (***mode A***).

Alternatively, it can be opened using the ***with*** and ***as*** statements, which will close it once the code block is finished (***mode B***).

mode A →



[1]:
```
inp = open('inp_file.txt', 'r')
out = open('out_file.txt', 'w')
n = 0
for line in inp:
    n += 1
    out.write(str(n) + ' ' + line)
inp.close()
out.close()
```

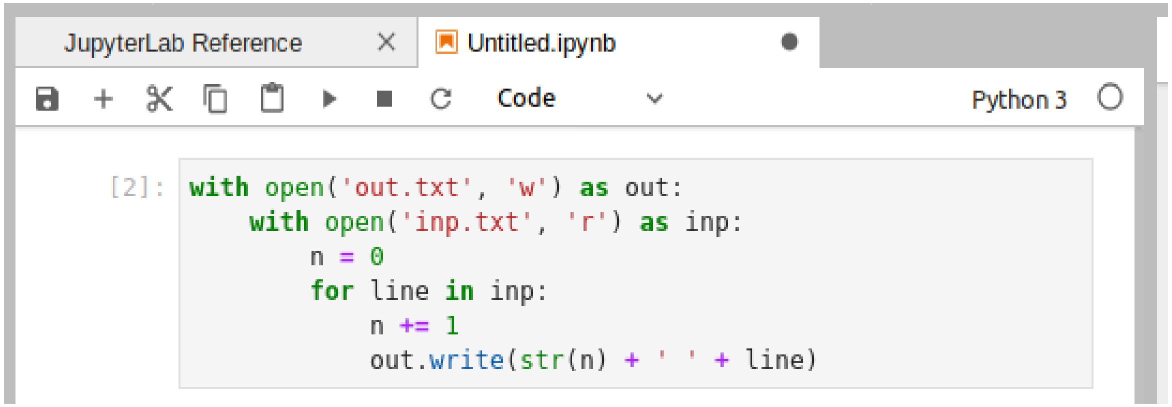
inp_file.txt

```
A
B
C
```

out_file.txt

```
1 A
2 B
3 C
```

mode B →

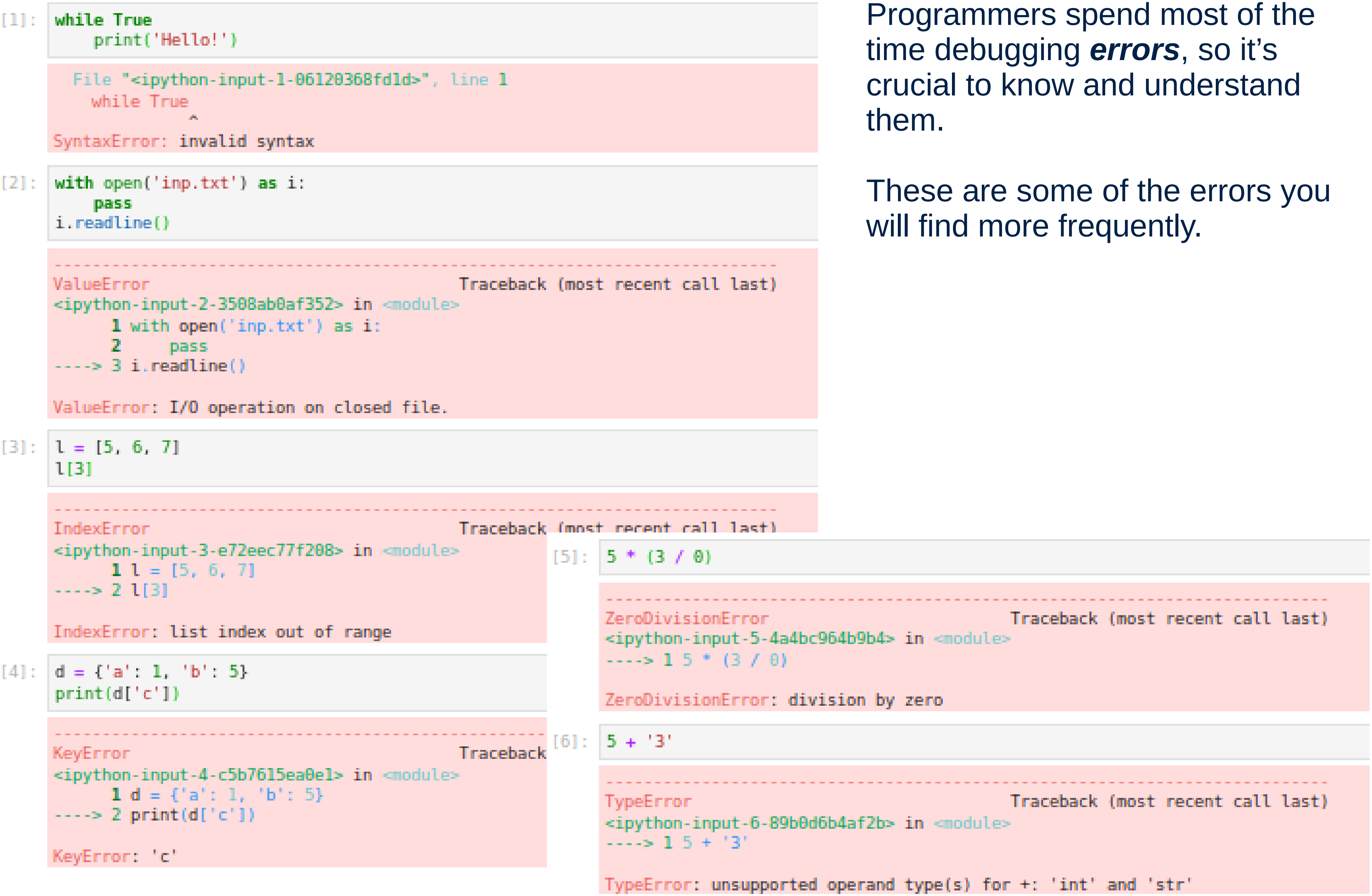


[2]:
```
with open('out.txt', 'w') as out:
    with open('inp.txt', 'r') as inp:
        n = 0
        for line in inp:
            n += 1
            out.write(str(n) + ' ' + line)
```

Understanding errors

Programmers spend most of the time debugging ***errors***, so it’s crucial to know and understand them.

These are some of the errors you will find more frequently.



```
[1]: while True
         print('Hello!')

  File "<ipython-input-1-06120368fd1d>", line 1
    while True
              ^
SyntaxError: invalid syntax
```

```
[2]: with open('inp.txt') as i:
         pass
     i.readline()

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-3508ab0af352> in <module>
      1 with open('inp.txt') as i:
      2     pass
----> 3 i.readline()

ValueError: I/O operation on closed file.
```

```
[3]: l = [5, 6, 7]
     l[3]

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-3-e72eec77f208> in <module>
      1 l = [5, 6, 7]
----> 2 l[3]

IndexError: list index out of range
```

```
[4]: d = {'a': 1, 'b': 5}
     print(d['c'])

---------------------------------------------------------------
KeyError                                Traceback
<ipython-input-4-c5b7615ea0e1> in <module>
      1 d = {'a': 1, 'b': 5}
----> 2 print(d['c'])

KeyError: 'c'
```

```
[5]: 5 * (3 / 0)

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-5-4a4bc964b9b4> in <module>
----> 1 5 * (3 / 0)

ZeroDivisionError: division by zero
```

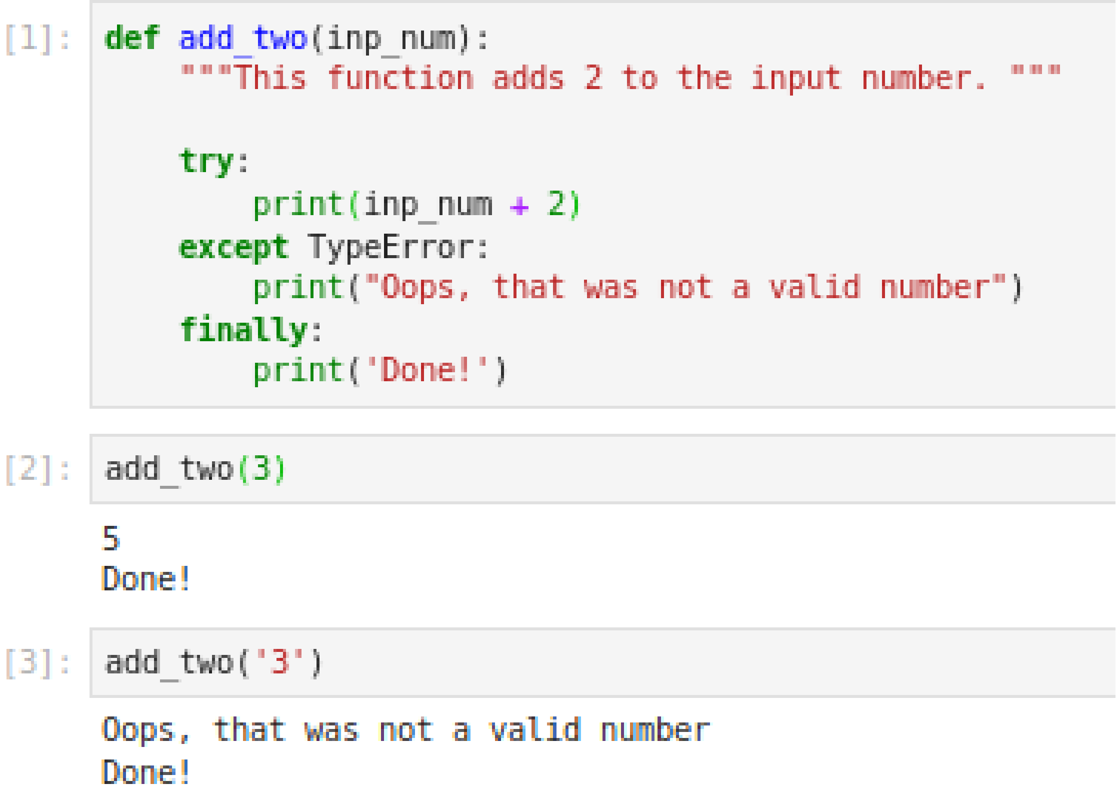
```
[6]: 5 + '3'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-89b0d6b4af2b> in <module>
----> 1 5 + '3'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Handling exceptions

Even if a given piece of code is syntactically correct, it may still cause an error when you attempt to execute it. These errors are called exceptions and can be handled using the ***try*** and ***except*** statements.

The code first attempts to run the block in the try clause. If no exception occurs, the except clause is skipped, otherwise it is executed. The ***finally*** clause is executed in any case.

If another exception, not captured by the except clause, occurs, then the unhandled exception will be displayed and the execution stopped.



```
[1]: def add_two(inp_num):
         """This function adds 2 to the input number. """

         try:
             print(inp_num + 2)
         except TypeError:
             print("Oops, that was not a valid number")
         finally:
             print('Done!')
```

```
[2]: add_two(3)
```

```
5
Done!
```

```
[3]: add_two('3')
```

```
Oops, that was not a valid number
Done!
```

Practical session

Practical session 3 – exercises

1) Write a function that takes as input a DNA sequence (e.g. **'ATGGTCA'**) and prints the string **'This DNA sequence is N base pair long'**, where N is the length of your input sequence (for the example above, it would be 7).

2) Create a file **inp.txt** containing the following RNA sequences, one per line: **'UGAAAC', 'GGGUCUUUU', 'GUUAAAACAACCCU'.** Read this file and write a new file **out.txt** containing the length of each sequence in inp.txt, one per line. Tip: each input line contains the extra character **'\n'**, which does not have to be counted, but that has to be written in the output file, in order to have separate lines.

3) Given the dictionary **bases = {'A': 'A', 'C': 'C', 'G': 'G', 'T': 'U'}** and the DNA sequence **dna = 'ATCKNGA'**, convert it to a new RNA sequence called **rna** using **bases**. Catch the KeyError exception raised by **'K'** and **'N'** and print the error message **'Wrong base found!'** together with the base that raised the exception.

Practical session 3 – solutions

1)

```
def seq_length(dna):
    """This function prints a string saying the length of 'dna'. """

    print('This DNA sequence is', len(dna), 'base pair long')

  seq_length('AAATTGGGG')   # the result must be: This DNA sequence is 9 base pair long
```

2)

```
inp = open('inp.txt', 'r')
out = open('out.txt', 'w')
for row in inp:
    out.write(str(len(row) – 1) + '\n')
inp.close()
out.close()
```

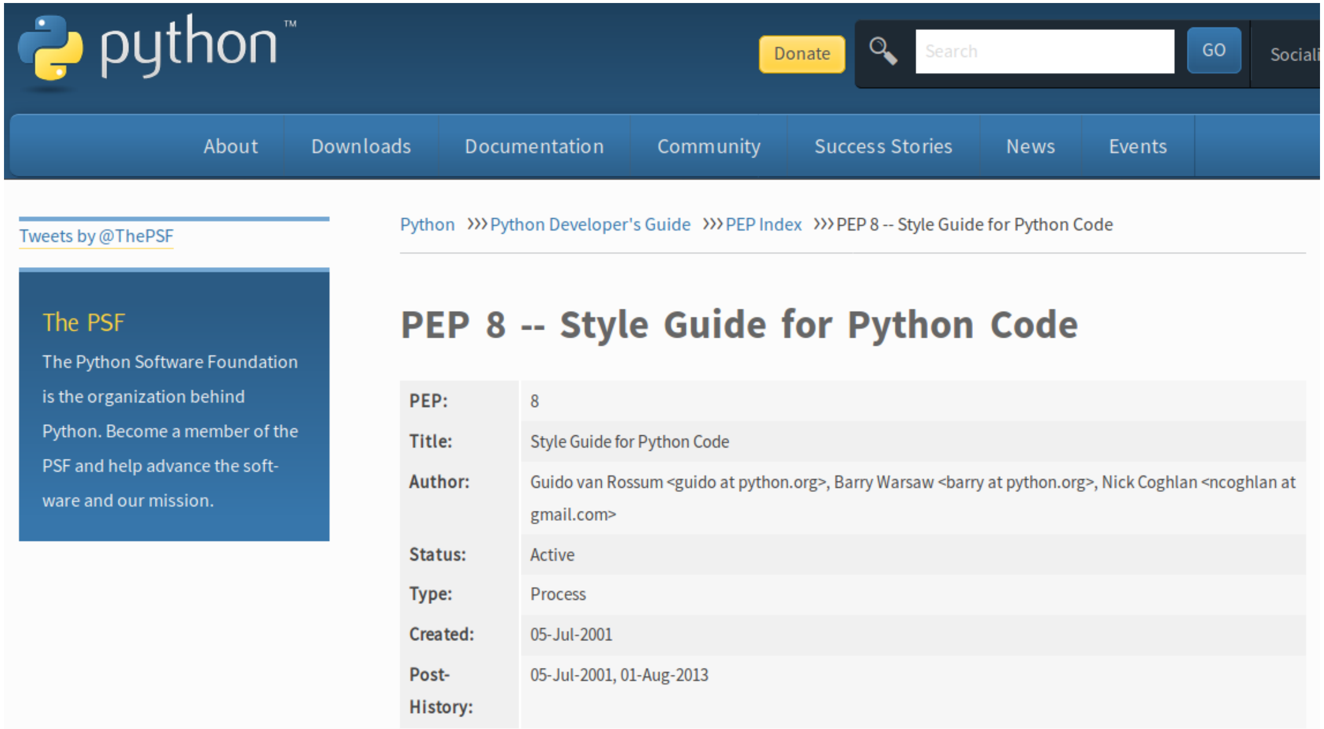
3)

```
bases = {'A': 'A', 'C': 'C', 'G': 'G', 'T': 'U'}
dna = 'ATCKNGA'
rna = ''
for letter in dna:
    try:
        rna += bases[letter]
    except KeyError:
        print('Wrong base found!', letter)

print(rna)
```

Coding style

From its invention, Python's philosophy emphasized ***code readability***, making it easier to be read and understood by others, when code is shared.

The ***Python Enhancement Proposals (PEPs),*** reviewed by the Python community and steering council, are the primary mechanism for proposing major new features and taking new design decisions.



The PSF

The Python Software Foundation is the organization behind Python. Become a member of the PSF and help advance the software and our mission.

Python ››› Python Developer's Guide ››› PEP Index ››› PEP 8 -- Style Guide for Python Code

PEP 8 -- Style Guide for Python Code

PEP:	8
Title:	Style Guide for Python Code
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

One of this proposals, ***PEP 8***, focuses on coding style and is a style guide that promotes readable and eye-pleasing code development. Here are some of the main points:

- Use ***4-space indentation*** instead of tabs (which introduce confusion)
- Code lines should not exceed ***80 characters*** (to help users with small displays)
- Functions and classes should be separated by ***blank lines***
- ***Comments*** should be on separate lines, rather than with code
- Use ***docstrings*** as much as possible (to enhance code documentation and readability)
- Use ***spaces*** around operators and after commas
- ***Name classes and functions consistently***, using UpperCamelCase for classes and lowercase_with_underscores for functions and methods.
- Code should always use ***UTF-8*** (or ***ASCII*** in Python 2), don't use other characters (to make code readable by people speaking other languages and using different keyboards)
- ***Imports*** should be on separate lines

Useful resources

The topics presented in this course can be found in more detail at https://docs.python.org/3/tutorial/ , although I would really recommend diving deeper into Python. Here is a list of some useful resources for learning Python:

- **LinkedIn Learning**: free for Uni Oxford members! https://www.linkedin.com/learning/
- **Learn Python**: https://www.learnpython.org/
- **DataCamp**: https://www.datacamp.com/
- **Coursera**: https://www.coursera.org/
- **edX**: https://www.edx.org/learn/python

...and for practicing Python (problems with solutions):

- **Rosalind**: http://rosalind.info/problems/list-view/
- **Leetcode**: https://leetcode.com/problemset/all/
- **CodingBat**: https://codingbat.com/python
- **W3 Resource**: https://www.w3resource.com/python-exercises/
- **Practice Python**: https://www.practicepython.org/

Thank you for your attention!

Questions?

silvia@well.ox.ac.uk

bioinformatics@well.ox.ac.uk