Scientific Programming Practical 7

Introduction

Luca Bianco - Academic Year 2020-21
luca.bianco@fmach.it

Functions: just a reminder

A function is a block of code that has a name and that performs a task. A function can be thought of as a **box that gets an input and returns an output**.

The basic definition of a function is:

```
def function_name(input) :
    #code implementing the function
    ...
    ...
    return return_value
```

1. ***Reduce code duplication***: put in functions parts of code that are needed several times in the whole program so that you don't need to repeat the same code over and over again;

2. ***Decompose a complex task***: make the code easier to write and understand by splitting the whole program in several easier functions

Passing parameters from command line

Python provides the module **sys** to interact with the interpreter.

sys.argv is a **list** representing all the arguments passed to the python script from the command line.

From the terminal:

```
python3 my_program.py param1 param2 param3
```

Passing parameters from command line

Python provides the module **sys** to interact with the interpreter.

sys.argv is a **list** representing all the arguments passed to the python script from the command line.

```
import sys

print("\n")
print(sys.argv)
print("\n")
types = [x + ":\t" + str(type(x)) for x in sys.argv]

print("\n".join(types))
```

Passing parameters from command line

Python provides the module **sys** to interact with the interpreter.

sys.argv is a **list** representing all the arguments passed to the python script from the command line.

```
import sys

print("\n")
print(sys.argv)
print("\n")
types = [x + ":\t" + str(type(x)) for x in sys.argv]

print("\n".join(types))
```

```
biancol@bluhp:/tmp$ python3 intest.py param1 parameter2 29 "hi there" 129


['intest.py', 'param1', 'parameter2', '29', 'hi there', '129']


intest.py:      <class 'str'>
param1: <class 'str'>
parameter2:     <class 'str'>
29:     <class 'str'>
hi there:       <class 'str'>
129:    <class 'str'>
biancol@bluhp:/tmp$
```

Passing parameters from command line

Python provides the module **sys** to interact with the interpreter.

sys.argv is a **list** representing all the arguments passed to the python script from the command line.

```
import sys
"""Test input from command line in systest.py"""

if len(sys.argv) != 4: #note that this is the number of params +1!!!
    print("Dear user, I was expecting 3 params. You gave me ",len(sys.argv)-1)
    exit(1)
else:
    for i in range(0,len(sys.argv)):
        print("Param {}:{} ({})".format(i,sys.argv[i],type(sys.argv[i])))
```

Check out:
https://docs.python.org/3/library/sys.html

Passing parameters from command line

```python
import sys
"""Test input from command line in systest.py"""

if(len(sys.argv) != 4):
    print("Dear user, I was expecting 3 params. You gave me ",len(sys.argv)-1)
    exit(1)
else:
    for i in range(0,len(sys.argv)):
        print("Param {}:{} ({})".format(i,sys.argv[i],type(sys.argv[i])))
```

```
biancol@bludell:~/work/courses/QCBsciprolab2020$ python3 exercises/systest.py
Dear user, I was expecting three parameters. You gave me  0
```

```
biancol@bludell:~/work/courses/QCBsciprolab2020$ python3 exercises/systest.py par1 2 parameter3
Param 0: exercises/systest.py (<class 'str'>)
Param 1: par1 (<class 'str'>)
Param 2: 2 (<class 'str'>)
Param 3: parameter3 (<class 'str'>)
```

Example: Write a script that takes two integers in input, i1 and i2, and computes the sum, difference, multiplication and division on them.

```python
import sys
"""Maths example with input from command line"""

if len(sys.argv) != 3:
    print("Dear user, I was expecting 2 params. You gave me ",len(sys.argv)-1)
    exit(1)
else:
    i1 = int(sys.argv[1])
    i2 = int(sys.argv[2])
    print("{} + {} = {}".format(i1,i2, i1 + i2))
    print("{} - {} = {}".format(i1,i2, i1 - i2))
    print("{} * {} = {}".format(i1,i2, i1 * i2))
    if i2 != 0:
        print("{} / {} = {}".format(i1,i2, i1 / i2))
    else:
        print("{} / {} = Infinite".format(i1,i2))
```

```
biancol@bluhp:~/Google Drive/work/scripts$ python3.6 /tmp/test.py
Dear user, I was expecting 2 params. You gave me  0
biancol@bluhp:~/Google Drive/work/scripts$ python3.6 /tmp/test.py 75 32
75 + 32 = 107
75 - 32 = 43
75 * 32 = 2400
75 / 32 = 2.34375
biancol@bluhp:~/Google Drive/work/scripts$ python3.6 /tmp/test.py 75 0
75 + 0 = 75
75 - 0 = 75
75 * 0 = 0
75 / 0 = Infinite
biancol@bluhp:~/Google Drive/work/scripts$ python3.6 /tmp/test.py 75 t
Traceback (most recent call last):
  File "/tmp/test.py", line 9, in <module>
    i2 = int(sys.argv[2])
ValueError: invalid literal for int() with base 10: 't'
```

Argparse

A more flexible solution…

Argparse is a command line parsing module which deals with **positional arguments** and **optional arguments**.

directory: mandatory argument

optional params

```
biancol@bludell:~$ mkdir --help
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
  -m, --mode=MODE   set file mode (as in chmod), not a=rwx - umask
  -p, --parents     no error if existing, make parent directories as needed
  -v, --verbose     print a message for each created directory
  -Z                   set SELinux security context of each created directory
                         to the default type
      --context[=CTX]  like -Z, or if CTX is specified then set the SELinux
                         or SMACK security context to CTX
      --help     display this help and exit
      --version  output version information and exit
```

Argparse

A more flexible solution…

Argparse is a command line parsing module which deals with **positional arguments** and **optional arguments**.

```
biancol@bluhp:~/Google Drive/work/courses/sciprolab1$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
  -b, --escape               print C-style escapes for nongraphic characters
      --block-size=SIZE      scale sizes by SIZE before printing them; e.g.,
                               '--block-size=M' prints sizes in units of
                               1,048,576 bytes; see SIZE format below
  -B, --ignore-backups       do not list implied entries ending with ~
```

Six steps:

1. Import the module

```
import argparse
```

2. Create the parser object

```
parser = argparse.ArgumentParser(description="This is the description of the program")
```

3. Add positional arguments

```
parser.add_argument("arg_name", type = obj,
                    help = "Description of the parameter)
```

4. Add optional arguments

```
parser.add_argument("-p", "--optional_arg", type = obj, default = def_val,
                    help = "Description of the parameter)
```

5. Parse the arguments

```
args = parser.parse_args()
```

6. Retrieve and process the arguments

```
myArgName = args.arg_name
myOptArg = args.optional_arg
```

https://docs.python.org/3/howto/argparse.html

Argparse

Example: Write a script that takes two integers in input, i1 and i2, and computes the sum, difference, multiplication and division on them.

```
import argparse
"""Maths example with input from command line"""
parser = argparse.ArgumentParser(description="""This script gets two integers  in input
and performs some operations on them""")
parser.add_argument("i1", type=int,
                    help="The first integer")
parser.add_argument("i2", type=int,
                    help="The second integer")


args = parser.parse_args()

i1 = args.i1
i2 = args.i2
print("{} + {} = {}".format(i1,i2, i1 + i2))
print("{} - {} = {}".format(i1,i2, i1 - i2))
print("{} * {} = {}".format(i1,i2, i1 * i2))
if i2 != 0:
    print("{} / {} = {}".format(i1,i2, i1 / i2))
else:
    print("{} / {} = Infinite".format(i1,i2))
```

Argparse

Example: Write a script that takes two integers in input, i1 and i2, and computes the sum, difference, multiplication and division on them.

```
biancol@bludell:~/work/courses/QCBsciprolab2020$ python exercises/systest_argparse.py
usage: systest_argparse.py [-h] i1 i2
systest_argparse.py: error: too few arguments
biancol@bludell:~/work/courses/QCBsciprolab2020$ python exercises/systest_argparse.py --help
usage: systest_argparse.py [-h] i1 i2

This script gets two integers in input and performs some operations on them

positional arguments:
  i1          The first integer
  i2          The second integer

optional arguments:
  -h, --help  show this help message and exit
biancol@bludell:~/work/courses/QCBsciprolab2020$ python exercises/systest_argparse.py 32 0
32 + 0 = 32
32 - 0 = 32
32 * 0 = 0
32 / 0 = Infinite
biancol@bludell:~/work/courses/QCBsciprolab2020$ python exercises/systest_argparse.py 32 t
usage: systest_argparse.py [-h] i1 i2
systest_argparse.py: error: argument i2: invalid int value: 't'
```

Argparse

```python
import argparse
import gzip

parser = argparse.ArgumentParser(description="""Reads and prints a text file""")
parser.add_argument("filename", type=str, help="The file name")
parser.add_argument("-z", "--gzipped", action="store_true",
                    help="If set, input file is assumed gzipped")

args = parser.parse_args()
inputFile = args.filename
fh = ""
if args.gzipped:
    fh = gzip.open(inputFile, "rt")
else:
    fh = open(inputFile, "r")

for line in fh:
    line = line.strip("\n")
    print(line)

fh.close()
```

```
biancol@bluhp:~/Google Drive/work/courses/sciprolab1$ python3 exercises/readFile_gz.py  -h
usage: readFile_gz.py [-h] [-z] filename

Reads and prints a text file

positional arguments:
  filename       The file name

optional arguments:
  -h, --help     show this help message and exit
  -z, --gzipped  If set, input file is assumed gzipped
biancol@bluhp:~/Google Drive/work/courses/sciprolab1$
```

Argparse

```python
import argparse
import gzip

parser = argparse.ArgumentParser(description="""Reads and prints a text file""")
parser.add_argument("filename", type=str, help="The file name")
parser.add_argument("-z", "--gzipped", action="store_true",
                    help="If set, input file is assumed gzipped")

args = parser.parse_args()
inputFile = args.filename
fh = ""
if args.gzipped:
    fh = gzip.open(inputFile, "rt")
else:
    fh = open(inputFile, "r")

for line in fh:
    line = line.strip("\n")
    print(line)

fh.close()
```

```
biancol@bluhp:~/Google Drive/work/courses/sciprolab1$ python3 exercises/readFile_gz.py file_samples/textFile.txt
Hi everybody,
This is my first file
and it contains a total of
four lines!
```

```
biancol@bluhp:~/Google Drive/work/courses/sciprolab1$ python3 exercises/readFile_gz.py file_samples/textFile.gz -z
Hi everybody,
This is my first file
and it contains a total of
four lines!
```

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

Previous topic
10. Full Grammar specification

Next topic
1. Introduction

This Page
Report a Bug
Show Source

The Python Standard Library

While The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that
mers, as well as modules written in Python that
provide standard
citly designed to encourage and enhance the
portability of Pyth

The Python insta
l components. For Unix-like operating systems
Python is normal
e operating system to obtain some or all of the
optional compon

In addition to the
nd modules to packages and entire application
development fra

- 1. Introduct
- 2. Built-in F
- 3. Built-in C
  - 3.1. C
- 4. Built-in T
  - 4.1. T
  - 4.2. B
  - 4.3. C
  - 4.4. N
  - 4.5. It
  - 4.6. S
  - 4.7. T
  - 4.8. B
  - 4.9. S

`gzip.compress(data, compresslevel=9, *, mtime=None)`

Compress the *data*, returning a `bytes` object containing the compressed data. *compresslevel* and *mtime* have the same meaning as in the `GzipFile` constructor above.

New in version 3.2.

Changed in version 3.8: Added the *mtime* parameter for reproducible output.

`gzip.decompress(data)`

Decompress the *data*, returning a `bytes` object containing the uncompressed data.

New in version 3.2.

Examples of usage

Example of how to read a compressed file:

```
import gzip
with gzip.open('/home/joe/file.txt.gz', 'rb') as f:
    file_content = f.read()
```

Example of how to create a compressed GZIP file:

```
import gzip
content = b"Lots of content here"
with gzip.open('/home/joe/file.txt.gz', 'wb') as f:
    f.write(content)
```

Example of how to GZIP compress an existing file:

Example: Let's write a program that reads the content of a file and prints to screen some stats like the number of lines, the number of characters and maximum number of characters in one line. Optionally (if flag -v is set) it should print the content of the file. You can find a text file here textFile.txt:

```
import argparse


def readText(f):
    """reads the file and returns a list with
    each line as separate element"""
    myF = open(f, "r")
    ret = myF.readlines() #careful with big files!
    return ret


def computeStats(fileList):
    """returns a tuple (num.lines, num.characters,max_char.line)"""
    num_lines = len(fileList)
    lines_len = [len(x.replace("\n", "")) for x in fileList]
    num_char = sum(lines_len)
    max_char = max(lines_len)
    return (num_lines, num_char, max_char)
```

```
parser = argparse.ArgumentParser(description="Computes file stats")
parser.add_argument("inputFile", type=str, help="The input file")
parser.add_argument(
    "-v", "--verbose", action="store_true", help="if set, prints the file content")

args = parser.parse_args()

inFile = args.inputFile
lines = readText(inFile)
stats = computeStats(lines)
if args.verbose:
    print("File content:\n{}\n".format("".join(lines)))
print(
    "Stats:\nN.lines:{}\nN.chars:{}\nMax. char in line:{}".format(
        stats[0], stats[1], stats[2]))
```

Example: Let's write a program that reads the content of a file and prints to screen some stats like the number of lines, the number of characters and maximum number of characters in one line. Optionally (if flag -v is set) it should print the content of the file. You can find a text file here textFile.txt:

Output with -v flag:

```
biancol@bluhp:~/Google Drive/work/courses/QCBsciprolab$ python3 fileStats.py file_samples/textFile.txt -v
File content:
Hi everybody,
This is my first file
and it contains a total of
four lines!

Stats:
N.lines:4
N.chars:71
Max. char in line:26
```

Output without -v flag:

```
biancol@bluhp:~/Google Drive/work/courses/QCBsciprolab$ python3 fileStats.py file_samples/textFile.txt
Stats:
N.lines:4
N.chars:71
Max. char in line:26
```

http://qcbprolab2020.readthedocs.io/en/latest/practical7.html

Exercises

1. Modify the program of Exercise 5 of Practical 6 in order to allow users to specify the input and output files from command line. Then test it with the provided files. The text of the exercise follows:

Write a python program that reads two files. The first is a one column text file (contig_ids.txt) with the identifiers of some contigs that are present in the second file, which is a fasta formatted file (contigs82.fasta). The program will write on a third, fasta formatted file (e.g. filtered_contigs.fasta) only those entries in *contigs82.fasta* having identifier in *contig_ids.txt*.

Show/Hide Solution

2. Write a python script that takes in input a single-entry .fasta file (specified from the command line) of the amino-acidic sequence of a protein and prints off 1) the total number of aminoacids, 2) for each aminoacid, its count and percentage of the whole. Optionally, it the user specifies the flag "*-S*" (–search) followed by a string representing an aminoacid sequence, the program should count and print how many times that input sequence appears. Download the Sars-Cov-2 Spike Protein and test your script on it. *Please use functions*.

Show/Hide Solution

3. Cytoscape is a well known tool to perform network analysis. It is well integrated with several online databases housing for example protein-protein interactions like EBI's IntAct. It is also able to read and write a very simple text file called `.sif` to represent interactions between the nodes of a network. Sif formatted files are tab separated (`\t`) and each line represents a connection between the nodes of the network. For example:

```
node1 interaction1 node2
node1 interaction2 node3
node2 interaction1 node3
```

represents two types of interactions between node1, node2 and node3. Normally nodes are represented as circles in a network (graph) and interactions as lines (that can be of different kinds) connecting nodes (edges). The following is an extract from the file pka.sif that has been downloaded by Cytoscape from the database IntAct and represents the interactions of the