Scientific Programming Practical 7

Introduction

Luca Bianco - Academic Year 2018-19
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.

```
"""Test input from command line"""
import sys


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

```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  1: bash
biancol@bluhp:~/Google Drive/work/courses/sciprolab1/exercises$
```

Check out: https://docs.python.org/3/

Passing parameters from command line

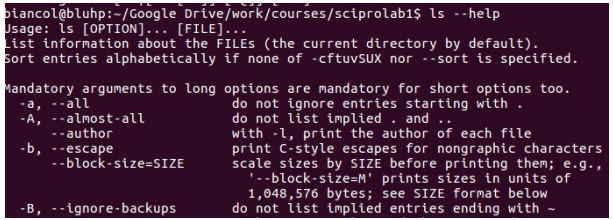
```python
"""Test input from command line"""
import sys

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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: Python

```
/usr/bin/python3.6 "/home/biancol/Google Drive/work/courses/sciprolab1/exercises/systest.py"
biancol@bluhp:~/Google Drive/work/courses/sciprolab1/exercises$ /usr/bin/python3.6 "/home/biancol/Google Dri
ve/work/courses/sciprolab1/exercises/systest.py"
Dear user, I was expecting three parameters. You gave me  0
biancol@bluhp:~/Google Drive/work/courses/sciprolab1/exercises$ /usr/bin/python3.6 "/home/biancol/Google Dri
ve/work/courses/sciprolab1/exercises/systest.py" param1 2 param3
Param 0: /home/biancol/Google Drive/work/courses/sciprolab1/exercises/systest.py (<class 'str'>)
Param 1: param1 (<class 'str'>)
Param 2: 2 (<class 'str'>)
Param 3: param3 (<class 'str'>)
biancol@bluhp:~/Google Drive/work/courses/sciprolab1/exercises$
```

Argparse

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



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
```

Argparse

Example: Let's write a program that reads and prints to screen a text file specified by the user. Optionally, the file might be compressed with gzip to save space. The user should be able to read also gzipped files. Hint: use the module gzip which is very similar to the standard file management method

```
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()
```

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 provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the Python Package Index.

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:

```
def readText(f):
    """reads the file and returns a list with
    each line as separate element"""
    myF = open(f, "r")
    ret = myF.readlines()
    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://qcbprolab.readthedocs.io/en/latest/practical7.html

Exercises

1. Modify the program of Exercise 4 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. 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 Protein Kinase A (PKA) of E.coli:

```
P75742  EBI-9168813 P76594
```