Introduction to programming in Python for biologists

Course Presentation

Instructor

- Antonio Carvajal-Rodríguez.
- Office 23
- Department of Genetics. University of Vigo.

Contents

- ✓Programming in Python
- ✓Introduction to object oriented programming

Evaluation

- ✓Class participation: 10%
- ✓Troubleshooting and programming exercises: 70%
- ✓Final programming exercise: 20%

Introduction

- Initial settings
- Python: interactive vs. file mode
- First programming concepts:

  - ✓program & variables
  - ✓code blocks: indentation
  - ✓data types: immutable vs. mutable
  - ✓built-in functions: *print*, *len*
  - ✓comparison and logical operators
  - ✓decisions: the *if* statement

Introduction

- Initial settings

```
biosdev@biosdev~VirtualBox:~$
biosdev@biosdev~VirtualBox:~$ nano .bash_aliases

        export PATH="$PATH:${HOME}/ProgBioinf"

biosdev@biosdev~VirtualBox:~$  (close and open the terminal) echo $PATH
biosdev@biosdev~VirtualBox:~$  cd ProgBioinfs
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $
```

Introduction

- Python: interactive vs file mode

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ python

Python 2.7.6 (default, Mar 22 2014, 22:59:38)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information
>>>
>>>help()
Help> keywords
Help> quit
>>>
>>> print("programming for bioinformatics")
>>> print("programming\nfor\nbioinformatics")
>>> quit()
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $
```

Introduction

- Python: interactive vs file mode

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ subl first.py (or gedit first.py)
        #!/usr/bin/env python

        DNASeq='GATACA'

        print("Sequence: %s" %DNASeq)
        print("Sequence:" + DNASeq)

        text="Sequence: " + DNASeq
        print(text)
# end of the file
```

Introduction

- Python: interactive vs file mode

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ first.py

Sequence: GATACA
Sequence:GATACA
Sequence: GATACA

biosdev@biosdev~VirtualBox:~/ ProgBioinfs $
```

Introduction

- Python: interactive vs file mode

```
>>>help(len)
Help on built-in function len in module __builtin__:

len(...)
    len(object) -> integer

    Return the number of items of a sequence or mapping.
>>>
>>> text="gatac a"
>>>len(text)
7
>>>
```

Introduction

- Python: interactive vs file mode

```
>>> if len(text)>0:
...         print(“data")
... else:
...         print("none")
...
data
>>> text=“”
>>> len(text)
0
>>>dir(text)
>>>help(text.count)
>>>text=‘GATACA’
>>>text.count(‘A’)
3
```

Problem 1

Exercise 1.1: Develop a program called *DNA_calc.py* for computing the melting temperature MT of a DNA sequence. Use as example the sequence contained in *sequence.txt* (you can copy and paste it into your code). However, the program must work for sequences of any length. The formula for MT is

- ✓MT = 4*(C+G) + 2*(A+T) for short sequences (< 14 nt) and
- ✓MT = 64.9 + 41*(C+G-16.4)/SeqLength for larger sequences.

Hints:

- ✓Use string method *.count*()
- ✓Use print to show the result

Problem 1

- More programming concepts in Python:
  - ✓input from the user: raw_input
  - ✓string formatting with the print function
- The string data type
  - ✓*.upper*()
  - ✓*.replace*()
  - ✓*.find*()
  - ✓*.split*()

Problem 1

- User input

```
>>> text=raw_input(“Introduce the DNA sequence: ")
Introduce the DNA sequence: aagt
>>> print(text)
aagt
>>> print(text.upper())
AAGT
>>> help(“”.find)
>>> help(“”.replace)
>>> help(“”.split)
```

Problem 1

- String formatting

```
>>> nA=4; nC=7
>>> print("There are %d A and %d C bases." % (nA, nC) )
There are 4 A and 7 C bases.
>>>
>>> print("The ratio A/(A+C) is %.2f of %d A+C bases." %(nA/(nA+nC), nA+nC))
The ratio A/(A+C) is 0.36 of 11 A+C bases.
>>>
>>> print("The ratio A/C is %f." %(nA/nC))
The ratio A/C is 0.571429.
```

Problem 1

Exercise 1.2: Develop a program called *complementary_dna.py* for computing the complement of a DNA sequence introduced from the keyboard.

Hints:

✓Use string method *.replace*()

Extra: Any non-nucleotide symbol must be complemented by an ‘x’, e.g. the complement of GATAHA should be CTATxT

Problem 1

Exercise 1.3: Develop a program called *RFL.py* that given the motif GAATTC compute the length of the two fragments produced after cutting a DNA sequence at the position just before the motif. Use the same sequence of exercise 1.1.

Hints:

✓Use string method *.find*()

Extra: Allow user-defined motif and read the sequence from file.

Problem 2

- Loops and lists (and tuples)
- Modifying Python variables
- Other data types: dictionaries
- Comprehensions
- File IO
  - ✓read input file
  - ✓Parse content
  - ✓write output file

Problem 2

- Lists

```
>>> dir(list)
>>> help(list.append)
>>> L=[ ]
>>> L.append(0)
>>> print(L)
[0]
>>> L.append(1)
>>> print(L)
[0, 1]
>>>
```

Problem 2

- Loops

```
>>> slist=['G','A','T','A','C','A']
>>> print slist[0]
G
>>> print slist[1]
A
>>> for nucleotide in slist:
...         print(nucleotide)
...
G
A
T
A
C
A
```

Problem 2

Exercise 2.1: Develop a program called *parseseq.py* that converts from a string in Phylip format:

```
2 10
Pop1_1    GTTATACCTC
Pop1_2    CCTATACCTC
```

to another string in the Fasta format:

```
>Pop1_1
GTTATACCTC
>Pop1_2
CCTATACCTC
```

Hints

✓Use .*split*() method

Problem 2

- Modifying Python variables

```
>>> seq='GATACA'
>>> seq2=seq
>>> print(seq+"\n"+seq2)
GATACA
GATACA
>>> seq="other"
>>> print(seq+"\n"+seq2)
other
GATACA
>>> seq[0]='A'   #  what happens??
```

Problem 2

- Modifying Python variables

```
>>> slist=['G','A','T','A','C','A']
>>> slist2=slist
>>> print(str(slist)+"\n"+str(slist2))
['G', 'A', 'T', 'A', 'C', 'A']
['G', 'A', 'T', 'A', 'C', 'A']
>>> slist[0]='A'
>>> print(str(slist)+"\n"+str(slist2))
['A', 'A', 'T', 'A', 'C', 'A']
['A', 'A', 'T', 'A', 'C', 'A']
>>>
```

Problem 2

- Modifying Python variables

```
>>> slist3=slist[:]  #but wait for problem 4.4 you could need copy.deepcopy()
>>> print(str(slist)+"\n"+str(slist3))
['A', 'A', 'T', 'A', 'C', 'A']
['A', 'A', 'T', 'A', 'C', 'A']
>>> slist[0]='G'
>>> print(str(slist)+"\n"+str(slist3))
['G', 'A', 'T', 'A', 'C', 'A']
['A', 'A', 'T', 'A', 'C', 'A']
>>>
```

Problem 2

- I/O

```
>>> help(open)
>>> file= open(“sequence.txt”, 'r')
>>> text= “”
>>>
>>> for line in file:
...         text+=line
...
>> print(text)
ACTGATCGATTACGTATAGTAGAATTCTATCATACATATATATCGATGCGTTC
>>>file.close()
>>>file2 = open(“sequence2.txt", 'w')
>>> file2 .write(text)
>>>file2.close()
```

Problem 2

- Dictionaries

```
>>> codon={'A': 'GCT', 'C':'TGT'}
>>> print(codon['A'])
GCT
>>>help(zip)
>>> list1=['A', 'C']
>>> list2=['GCC', 'TGC']
>>> codon=dict(zip(list1,list2))
>>> print(codon['A'])
GCC
```

- Comprehensions

```
>>>list=range(1,11)
>>> squares= [numbers**2 for numbers  in list]
```

Problem 2

Exercise 2.2: Develop a program called *parsecodon.py* that after reading the input file AA.txt, generate a table of the kind

Id	Aminoacid (codon)
A	Alanine (GCT, GCC, GCA, GCG)

and write the table in a file called CodonTable.txt

Hints:

✓Use list and dictionary with the Id as key

Problem 3

- User-defined functions
- Command line arguments
- Handle errors
- Modules: urlib, random…
- Exercises

Problem 3

- User-defined functions

```
>>> def sum(x,y):
...         return x+y
...
>>> print(sum(2,5))
7
>>>
```

Problem 3

- User-defined functions

```
>>> def getword(str_seq,sep,position):
...         return seq.split(sep)[position-1]
...
>>> words="programming for bioinformatics course"
>>> print(getword(words," ",3))
bioinformatics
>>> print(getword(words," ",1))
programming
>>> print(getword(words," ",4))
course
>>>
```

Problem 3

- User-defined functions

```
>>> numbers="1,2,3,4,5,6,7,8,9,10"
>>> print(getword(numbers,",",3))
3
>>> print(getword(numbers,",",10))
10
>>>
```

Problem 3

- User-defined functions

```
>>> def sum(x,y,result):
...     result=x+y
...     print(result)
...
>>>r=0
>>>print(r)
0
>>> sum(4,3,r)
7
>>> print(r)
0
>>>
```

Problem 3

- User-defined functions

```
>>> def sum(x,y,result):
...     result[0]=x+y
...     print(result[0])
...
>>>r=[0]
>>> print(r[0])
0
>>> sum(4,3,r)
7
>>> print(r[0])
7
>>>
```

Problem 3

- Command line arguments

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ subl sum.py
        #!/usr/bin/env python
        import sys
        sum=0
        if len(sys.argv)>1:
                list=sys.argv[1:]
                for s in list:
                   sum+=int(s)
        print(sum)
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ sum.py 1 3 4 7
15
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ sum.py
0
```

Problem 3

- Command line arguments

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ sum.py NOTNUMBERS

Traceback (most recent call last):
  File "/home/biosdev/ProgBioinf/sum.py", line 7, in <module>
    sum+=int(s)
ValueError: invalid literal for int() with base 10: 'NOTNUMBERS'
```

Problem 3

- Handle errors: try & except

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ subl sum2.py
    #!/usr/bin/env python
    import sys
    sum=0
    if len(sys.argv)>1:
        try:
                list=sys.argv[1:]
                for s in list:
                    sum+=int(s)
                print(sum)
        except ValueError as e:
                print(e.args[0])
# end of file
```

Problem 3

- Handle errors: try & except

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ sum2.py 1 3 4 7
15
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ sum2.py NOTNUMBERS
invalid literal for int() with base 10: 'NOTNUMBERS'
```

Problem 3

Exercise 3.1: Similarly as you already did in exercise 2.1 develop a parser between Phylip and Fasta formats now using a function called *PtoFparser* that receives as argument a string of sequences in Phylip format and returns another string with the sequences in Fasta format. The main program that calls the function must receive by console an argument indicating the name of the input file with the sequences in Phylip format and would return the result in a new file with the same name as the input one but with the extension .fas.

Problem 3

- Modules: urlib, random

Problem 3

- Modules: urlib

```
>>> import urllib
>>> NewUrl ="http://practicalcomputing.org/aminoacid.html"
>>> WebContent = urllib.urlopen(NewUrl)
>>> for Line in WebContent:
...     print(Line.strip())
...
```

Problem 3

Exercise 3.2: Develop a program called *molw.py* that reads the content from the url in the previous example and write it in a file called *name.html* where "name" is any name passed as argument when calling the program. If no argument the default name is "aa". Thus if the call is

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ molw.phy
```

It should generate an output file *aa.html* but if we call

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ molw.phy Table
```

It generates an output file *Table.html*.

Problem 4

- Modules: random

```
>>> # rolling the dices
>>> import random
>>> NumofRolls =3
>>> NumofDices =5
>>> Rolls=[]
>>> Sides= (1,2,3,4,5,6)
>>> for X in range(NumofRolls ):
...     Resample=[random.choice(Sides) for Y in range(NumofDices)]
...     Rolls.append(Resample)
...
>>>
```

Problem 4

Exercise 4.1: The example just given in the previous slide is a dice roller. A) Identify the line of code producing the randomness and explain it. B) Develop a program called *DiceRoller.py* that receives as argument the number of rolls, make the plays and print the results.

Extra: Select the best hand given the following rank from highest to lowest: five of a kind, four, full, three, two-pair, one-pair.

Problem 4

Exercise 4.2: Recall our previous exercise and consider a sample of *N* DNA sequences. If instead of the number of sides and dices we just consider *N,* we have a resampling with replacement method that can be used to perform a technique called bootstrapping. Thus, change your program to receive as argument the name of a sequence file in Phylip format and perform the following tasks:

Problem 4

Exercise 4.2: Recall our previous exercise and consider a sample of *N* DNA sequences. If instead of the number of sides and dices we just consider *N*, we have a resampling with replacement method that can be used to perform a technique called bootstrapping. Thus, change your program to receive as argument the name of a sequence file in Phylip format and perform the following tasks:

A) Read *SeqPhylip.text* file and use two lists to store separately the names and the sequences. You must test that:

```
assert len(names)==50
assert len(names)==len(sequences)
```

Problem 4

Exercise 4.2: Recall our previous exercise and consider a sample of *N* DNA sequences. If instead of the number of sides and dices we just consider *N,* we have a resampling with replacement method that can be used to perform a technique called bootstrapping. Thus, change your program to receive as argument the name of a sequence file in Phylip format and perform the following tasks:

B) From the total number of 50 sequences select sequences 6 and 7 and cut these two sequences maintaining only the nucleotide positions between 2000 and 7000. Test:

```
assert Names[0][len(Names[0])-1]==str(6)
assert Names[1][len(Names[0])-1]==str(7)
assert  len(Data[s]) == SeqEnd-SeqIni # for each sequence s
```

Problem 4

Exercise 4.2: Recall our previous exercise and consider a sample of *N* DNA sequences. If instead of the number of sides and dices we just consider *N,* we have a resampling with replacement method that can be used to perform a technique called bootstrapping. Thus, change your program to receive as argument the name of a sequence file in Phylip format and perform the following tasks:

C) Resample the sequences 100 times by generating, each time, two new sequences using the random.shuffle(*s*) method where *s* is a sequence.

Problem 4

Exercise 4.2: Recall our previous exercise and consider a sample of *N* DNA sequences. If instead of the number of sides and dices we just consider *N,* we have a resampling with replacement method that can be used to perform a technique called bootstrapping. Thus, change your program to receive as argument the name of a sequence file in Phylip format and perform the following tasks:

D) Compute and print the mean and variance of the number of purines (A+G) in the two original sequences and the mean and variance of the average number of purines through the resamplings. Explain the result.

Problem 5

- Object oriented programming
- Objects and classes
  - ✓attributes
  - ✓methods
  - ✓ initialization method
  - ✓ docstrings
- User-defined modules

Problem 5

- Object oriented programming

Object: Any thing can be viewed as an object. From the computer science point of view an object is a collection of data with an associated behavior.

Class: A kind of objects.

Example: If I say that my pet Missu is an object then the class of Missu is Cats. The class Cats have a superclass called Animals so Missu belongs to both. However Missu does not belong to the class Dogs.

Problem 5

- Object oriented programming

Object: Any thing can be viewed as an object. From the computer science point of view an object is a collection of data with an associated behavior.

Class: A kind of objects.

Another Example: The alpha-amylase gene is an object belonging to the Gene class.

Problem 5

- Class Gene

Attributes (public or private):

✓name
✓ locus
✓chromosome
✓sequence
✓taxon

Methods (public or private):

✓name

Problem 5

- Class Gene

  If you create a class called Gene and upload the class to the internet to be used by other programmers. Public means that the code of a "client" (a programmer using your class) can access this attribute or method. A private attribute means the opposite i.e. the code of a client cannot access this attribute directly (may exists a specific method to do so). Similarly for methods. A private method can only be directly used when the original class is programmed .

  Hint: You should never try to directly access the private attributes or methods of an object from outside of its class definition.

Problem 5

- Class Gene

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ subl gene.py
        #!/usr/bin/env python
“””This is docstring example in module gene.py from PFB course."""
        class Gene(object):
                def__init__(self, name=“”,locus=-1,chrom=-1,seq=“”,taxon=“”):
                        self.__name=name
                        self.locus=locus
                        self.__chromosome=chrom
                        self._sequence=seq
                        self.__taxon=taxon

                @property  # this is called a ‘decorator’ in the python world
                def  name(self):
                        return self.__name
```

Problem 5

- Class Gene

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ subl gene.py
                ...
                @name.setter
                def name(self, newname):
                        assert newname != “”
                        self.__name=newname
                #end file

        try:
                        if __name__ == "__main__“:
                           print("Use gene.py only as an imported module")
```

Problem 5

- Class Gene

```
biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ subl genomics.py

#!/usr/bin/env python
import gene
amy=gene.Gene("amy")
print(amy.name)
print(amy.locus)
print(amy.taxon) # ???

biosdev@biosdev~VirtualBox:~/ ProgBioinfs $ genomics.py
```

Problem 5

Exercise 5.1: Develop a module called *sequence.py* that contains the class Gene and the class Sequences. Gene is the same as before but with all attributes being private. Sequences should has as private attributes *name*, *sequence, format* and *cursor* plus the following public methods:

✓cursor: returns or set the current cursor value (use @)
✓name: returns the name of the sequence number *cursor* (from 1 to *n*)
✓sequence: returns the sequence with a given name.
✓sample_size: returns the number of sequences.
✓length: returns the length of the sequences if it is the same for all if not return -1.
✓purines: for a given sequence *cursor* returns a tuple with the number of A's and G's.
✓read: read a sequence file and store the corresponding information in the attributes.

Exercise 5.2:

Develop a program that imports the *Sequences* class and use it for reading the file SeqPhylip.txt. Compute the mean and variance of purines in the sequences.

Bibliography

- **Practical Computing for Biologists**. Haddock & Dunn. Sinauer Associates, Inc.; First edition (November 5, 2010).

- **Python Programming for the Absolute Beginner, 3rd Edition.** Michael Dawson. Cengage Learning PTR; 3rd edition (January 1, 2010)