

CMPT 830 – Bioinformatics and Computational Biology

Chapter 2: Python

Written by Nisha Puthiyedth
nisha.puthiyedth@usask.ca

Python - Introduction

- Python is an easy to learn, simple and powerful programming language.
- Created in 1991 by Guido Van Rossum.
- Mainly used for web development, software development, mathematics and system scripting.
- Most recent version is Python 3, however Python 2 is still popular even though not getting updated.

Story behind the name python

- Guido van Rossum, named the language after BBC show “Monty Python’s Flying Circus”.
- Van Rossum thought to have a short, unique and mysterious name for his programming language, hence the name Python.
- Read more about history of python here,
<https://www.python.org/doc/essays/foreword/>

Python - Installation

- Many PCs and Macs will have python already installed.
- In most cases it may be python 2 or 2.x
- To check if you have python installed on a windows PC, run the following on the command line.

```
C:\Users\Your Name>python --version
```

- To check if you have python installed on a Linux or Mac, run the following on the command line.

```
python --version
```

- If python is not installed on your computer or the version of python is not 3 or 3.x, then download python 3 for free from <https://www.python.org/>

Python – First steps

- Two ways of using python to run your program

1. Using interactive interpreter prompt

- Type `python` in your terminal and press `[enter]` key
- You should see `>>>` once you started python. you can start typing and is called python interpreter prompt.
- Here is an example what you should be seeing using a Mac OS X computer.

```
$ python
Python 2.7.10 (default, Feb  7 2017, 00:08:15)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Use `[ctrl+d]` or `exit()` to quit the interpreter prompt

Python – First steps

2. Using source file

- Using python source files
- Use an editor to write your source files.
- Basic requirement for an editor is the strategy to highlight the terms with different colors.
- Using an editor save a file called `hello.py`
- Always give the file extension `.py`

```
print("Hello World")
```

```
python hello.py
```

Python – getting help

- Use the build in function `help` to get information about any function or statement
- Eg: run `help("len")` to get the details about the function `len`

```
>>>help("len")

Help on built-in function len in module __builtin__:

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

    Return the number of items of a sequence or collection.
(END)
```

- Press `q` to exit the help

Python - Pycharm


- Pycharm is a IDE (integrated development Environment) for python.
- All python tools in one place
- Available free for Windows, Macs and Linux
- <https://www.jetbrains.com/pycharm/>
- Related documents in moodle



Python - Indentations

- White space at the beginning of the line is called indentation
- In other programming languages indentation is only for readability
- Indentation in python is to represent a block of statements
- Statements that go together should have same indentation

```
>>> if 10> 5:  
... print("ten is greater than five")  
File "<stdin>", line 2  
    print("ten is greater than five")  
      ^  
IndentationError: expected an indented block
```

```
>>> if 10 > 5:  
...  print("ten is greater than five")  
  
ten is greater than five
```

Python - Basics

- Start a comment with `#` within the code - `#This is a comment`
- Python also has an extended documentation capability called docstrings denoted by triple quotes at the beginning and end.
- It can be single line or multiline.

```
>>> """This is a
multiline docstring."""

print("Hello, World!")

Hello, World!
```

Python - scripts

- Script is text file containing a series of commands
- Start the script with a shebang line (#!) followed by the location of the python program
- This will tell the system where to send the rest of the file contents
- Find the location of python using,

```
watson:~ YourName$ which python  
/usr/bin/python
```

- Use **env** to send the scripts to env and execute through python

```
#!/usr/bin/env python
```

- More details about using python in Windows
<https://docs.python.org/3/using/windows.html>

Python

- Create a script that consider a DNA sequence, print out the information and find the percent of composition of each base in the sequence.
- Let's save a DNA sequence and print it

```
#!/usr/bin/env python
DNASeq = "ATGCATAC"    # or DNASeq = 'ATGCATAC'
print("Sequence:" , DNASeq)
```

Output will be,

Sequence: ATGCATAC

- Now save this file as seqtest.py

Python

- `DNASeq = 'ATGCATAC'`

↓ ↓
variable string

- Python can identify the variable, strings, etc

```
print(type(DNASeq))  
print("Sequence:" , DNASeq)
```

Output

```
<type 'str'>
```

- `'ATGCATAC'` and `'sequence'` are strings and are identified by python.
- How?

Python

- Variable
 - Use a variable to store any information
 - A variable is created at the moment you assign a value to it
- Variable names can be anything but,
 - Start with a letter or underscore character not a number
 - Can only contain alpha-numeric characters and underscore (A-z, 0-9 and _)
 - Are case sensitive - name, Name and NAME are three different variables
- Strings
 - Surrounded by either single quotes or double quotes
 - 'hello' is the same as "hello"
 - A single character is a string with length 1

Python

- Now find the length of the sequence

```
#!/usr/bin/env python
DNASeq = "ATGCATAC"
print("Sequence:" , DNASeq)
SeqLen = len(DNASeq)
print("sequence length:" , SeqLen)
```

Output

```
Sequence: ATGCATAC
Sequence Length:8
```

- To do some calculations to summarize the attributes of the sequence, we can make use of built-in functions like **len()**
- A function can take input values and perform some task and return one or more results

Python

- Find out the count of each base (A,T,G,C) in DNASEq

```
#!/usr/bin/env python
DNASeq = "ATGCATAC"
print("Sequence:" , DNASeq)
SeqLen = len(DNASeq)
print("sequence length:" , SeqLen)
NumA = DNASeq.count("A")
NumC = DNASeq.count("C")
#do for G and T
print("count of A:",NumA) #same for C, G and T
```

Output

```
Sequence: ATGCATAC
Sequence Length:8
Count of A: 3
#And the output for C, G and T
```

- **.count** is called a method in python

Python

- Multiline printing, adding two strings, sep

```
#use the sequence that is assigned to DNASeq
NumA = DNASeq.count("A")
NumC = DNASeq.count("C")
NumG = DNASeq.count("G")
NumT = DNASeq.count("T")
print("count of A:",NumA)
print("count of C:", NumC)

#To give printing command in single line

print("count of A:"+str(NumA), "count of C:" + str(NumC), "count of T:"+
str(NumT), "count of G:"+ str(NumG), sep = "\n")
```

Output

```
count of A:3
count of C:2
count of T:2
count of G:1
```

Python

- Another way of multiline printing
- Place holder %d

```
#use the sequence that is assigned to DNASEq
NumA = DNASeq.count("A")
NumC = DNASeq.count("C")
NumG = DNASeq.count("G")
NumT = DNASeq.count("T")

#To give printing command in single line
print("count of A: %d \n count of C: %d \n count of T: %d \n count of G:
%d" % (NumA, NumC, NumT, NumG))
```

Output

```
count of A:3
count of C:2
count of T:2
count of G:1
```

Python

- Determine the composition of each base A, C, T and G in DNASeq

```
#Determine the fraction of each base using SeqLen and NumA, NumC, NumG and NumT  
that are already calculated using DNASeq.
```

```
A = NumA/SeqLen
```

```
C = NumC/SeqLen
```

```
T = NumT/SeqLen
```

```
G = NumG/SeqLen
```

```
print("Base A:"+str(A), "Base C:" + str(C), "Base T:" + str(T), "Base G:" +  
str(G),sep = '\n')
```

Output

```
Base A:0.375
```

```
Base C:0.25
```

```
Base T:0.25
```

```
Base G:0.125
```

- Operators

Python

- Arithmetic operators
- $x = 4$ and $y = 2$

Operator	Name	Example	Output
+	Addition	$x + y$	6
-	Subtraction	$x - y$	2
*	Multiplication	$x * y$	8
/	Division	x / y	2
%	Modulus	$x \% y$	0
**	Exponentiation	$x ** y$	16
//	Floor division	$x // y$	2

- Table shows examples with integers

Python

- Try the same printing option %d with the fractions

```
#Determine the fraction of each base using SeqLen and NumA, NumC, NumG and NumT  
that are already calculated using DNASEq.
```

```
A = NumA/SeqLen
```

```
C = NumC/SeqLen
```

```
T = NumT/SeqLen
```

```
G = NumG/SeqLen
```

```
print("A: %d \n C: %d \n T: %d \n G: %d" % (A, C, T, G))
```

Output

```
A:0
```

```
C:0
```

```
T:0
```

```
G:0
```

Python

- $x = 5.2345$ where different place holders are given below

operator	Type	output
%d	Integer digit	5
%f	Floating point	5.2345
%.2f	Float with two decimal points	5.23
%5d	Integer padded to five places	5
%5.1f	Float with one decimal point padded to five spaces including decimal point	5.2
%s	String	"5.2345"

- Integer is a positive or negative whole number without decimals of unlimited length
- Float is a positive or negative number that has a decimal place

Python – whole program up to this point

```
#!/usr/bin/env python
print("sequence:" , DNASeq)
SeqLen = len(DNASeq)
print("sequence length:" , SeqLen)
NumA = DNASeq.count("A")
NumC = DNASeq.count("C")
NumG = DNASeq.count("G")
NumT = DNASeq.count("T")

print("count of A:"+str(NumA), "count of C:" + str(NumC), "count of T:"+
str(NumT), "count of G:" + str(NumG), sep = '\n')

A = NumA/SeqLen
C = NumC/SeqLen
T = NumT/SeqLen
G = NumG/SeqLen

print("A:"+str(A),"C:" + str(C),"T:" + str(T),"G:" + str(G),sep = '\n')
```

Python – verify the output

```
#verify the output by running the whole program in pycharm
sequence: ATGCATGC
sequence length: 8
count of A:2
count of C:2
count of T:2
count of G:2
A:0.25
C:0.25
T:0.25
G:0.25
```


Python

- Check if the count of C = count of T, using NumA, NumC, NumG and NumT from the previous example that is NumA = 3, NumC = 2, NumT = 2, NumG = 1

```
# Check if the count of C = count of T
if NumC == NumT:
    print("Count of C equal to count of T")
```

Output

Count of C equal to count of T

```
# Check if the count of C = count of T
if NumC = NumT:
    print("Count of C equal to count of T")
```

Output

```
if NumC = NumT:
^SyntaxError: invalid syntax
```

- Comparison operators

Python

- Comparison Operators
- $x = 4$ and $y = 2$

Operator	Name	In python	Output
<code>==</code>	Equal	<code>print(x == y)</code>	False
<code>!=</code>	Not equal	<code>print(x != y)</code>	True
<code>></code>	Greater than	<code>print(x > y)</code>	True
<code><</code>	Less than	<code>print(x < y)</code>	False
<code>>=</code>	Greater than or equal to	<code>print(x >= y)</code>	True
<code><=</code>	Less than or equal to	<code>print(x <= y)</code>	False

Python

- Check if count A = count C = count T using NumA, NumC, NumT from the previous example that is NumA = 3, NumC = 2, NumT = 2, NumG = 1

```
# Check if the count of A = count of C = count of T
if NumA == NumC and NumC == NumT:
    print("A is not equal to C but C is equal to T")
```

Output

#nothing will be printed as first condition is not true.

```
# Check if the count of C equal to count of T but not equal to count of A
if NumA != NumC or NumC == NumT:
    print("A is not equal to C but C is equal to T")
```

Output

A is not equal to C but C is equal to T

- Logical operators

Python

- Logical operators
- If $x = 4$

Operator	Name	In python	Output
and	Returns True if both statements are true	<code>print(x > 5 and x < 10)</code>	False
or	Returns True if one of the statements is true	<code>print(x > 3 or x < 4)</code>	True
not	Returns False if the result is true	<code>print(not(x > 3 and x < 10))</code>	False

Python

- Getting input from the user and perform some simple checks

```
#Ask the user to input the DNA sequence
DNASeq = input("Enter a DNA sequence:")

#convert the bases to upper case
DNASeq = DNASeq.upper()
print(DNASeq)

#Remove if there is spaces in user's input
DNASeq = DNASeq.replace(" ", "")
```

Output

Enter a DNA sequence:

ATcGA TCATG

ATCGA TCATG

ATCGATCATG

- `.lower()` - to convert to lower case
- `.upper()` – convert to upper case
- `.replace()` – To remove an item by replacing with another

Python

- Print sequence only if sequence length meets the given condition

```
#condition: print count of A only if sequence length < 14
DNASeq = "ATGCATGCATGC"
SeqLen = len(DNASeq)

if SeqLen in range(14):
    print(DNASeq.count("A"))
```

Output
3

- range()

Python

- Print the message “short or long sequence” according to the given condition

```
#condition: print the message if sequence length is in between 10 and 14
DNASeq = "ATGC"
SeqLen = len(DNASeq)

if SeqLen in range(10,14):
    print("Long sequence")
else:
    print("short sequence")

Output
Short sequence
```

- range(start, stop)
- Range(start, stop, step)

Python

- Print part of the sequence from the position 3 to 5

```
DNASeq = "ATGCATGCATGC"      #enter a DNA sequence
SeqLen = len(DNASeq)          #get the number of bases
#get the seq in 3 : 5 position
DNASeq2 = DNASeq[2:5]
print(DNASeq2)
```

Output
GCA

- Slicing
- DNASeq[start:end] starting to end-1
- DNASeq[:end] starting to end-1
- DNASeq[start:] starting to end
- DNASeq[:] copy of DNASeq

Python

- Cut the given sequence using T as separator

```
DNASeq = "ATGCATGCATGC"  
#split the sequence using T as separator  
DNASeq3 = DNASeq.split("T")  
print(DNASeq3)
```

Output

```
['A', 'GCA', 'GCA', 'GC']
```

- `split()` - split the string to list

Python

- Adding more calculations to `seqtest.py`
- Melting temperature of DNA sequence, T_m
- Depends on the number of strongly binding G+C pairs and weak A+T pairs
- $T_m = (4 * (G+C)) + (2 * (A+T))$ for less than 14 bases long
- $T_m = 64.9 + 41 * ((G+C) - 16.4) / \text{sequence length}$

```
#Calculate the melting temperature of the given sequence
DNASeq = "ATGCATGCATGC"
GC = NumG + NumC
AT = NumA + NumT
Tm = (4*GC) + (2*AT)
print("Melting temperature: %f" %(Tm))
```

Output

```
Melting temperature: 36.000000
```

Python

- Add the formula for both Sequence length > 14

```
#Calculate the melting temperature of the given sequence
DNASeq = "ATGCATGCATGC"
GC = NumG + NumC
AT = NumA + NumT
if SeqLen >= 14:
    TmLong = 64.9 + 41 * (GC - 16.4) / SeqLen
    print("Melting temperature: %f" % (TmLong))
```

Output
Melting temperature: 43.375000

- Conditional statements
- if statement

Python

- Add the formulas for both sequence length < 14 and > 14

```
#Calculate the melting temperature of the given sequence
DNASeq = "ATGCATGCATGC"
GC = NumG + NumC
AT = NumA + NumT
if SeqLen >= 14:
    TmLong = 64.9 + 41 * (GC - 16.4) / SeqLen
    print("Tm long: %f" % (TmLong))
else:
    Tm = (4*GC) + (2*AT)
    print("Tm short: %f" %(Tm))
```

Output

Sequence: ATGCATGCATGC

Tm short: 12.000000

- else

Python

- Check if sequence length is < 10 , $=10$ or >10

```
#Check the length of sequence
DNASeq = "ATGCATGCATGC"
DNASeq = input("Enter a DNA sequence:")
SeqLen = len(DNASeq)
if SeqLen > 10:
    print("long sequence")
elif SeqLen == 10:
    print("sequence with 10 bases")
elif SeqLen < 10:
    print("short sequence")
```

Output
long sequence

- elif

Python

```
#Determine the percentage of each base using SeqLen and NumA, NumC, NumG and NumT
#!/usr/bin/env python
DNASeq = "ATGCATGCATGC"
SeqLen = len(DNASeq)
NumA = DNASeq.count("A")    #for all bases
A = NumA/SeqLen              #for all bases
print("A:"+str(A),"C:" + str(C),"T:" + str(T),"G:" + str(G),sep = '\n')
```

```
#is the same as
DNASeq = "ATGCATGCATGC"
SeqLen = len(DNASeq)
Bases = "ATGC"
for base in Bases:
    percent = 100 * DNASeq.count(base) / SeqLen
    print("%s: %4.1f" % (base, percent))
```

Output

```
A: 25.0      T: 25.0      G: 25.0      C: 25.0
```

- Loops
- for
- while

Python

- Assign three sequences to a single variable

```
SeqList = ["ATGCATGC", "ATCATCATC", "AGCAGCAGC"]  
print(SeqList)
```

Output

```
["ATGCATGC", "ATCATCATC", "AGCAGCAGC"]
```

```
SeqList = list(["ATGCATGC", "ATCATCATC", "AGCAGCAGC"]) #note the double brackets  
print(SeqList)
```

Output

```
["ATGCATGC", "ATCATCATC", "AGCAGCAGC"]
```

- List – collection of ordered and changeable values inside square brackets
- List()

Python

```
#replace second sequence by "ATCAT"  
SeqList[1] = "ATCAT"  
print(SeqList)
```

```
["ATGCATGC", "ATCAT", "AGCAGCAGC"]
```

```
#add another sequence to the SeqList  
SeqList.append("ATGC")  
print(SeqList)
```

Output

```
["ATGCATGC", "ATCATCATC", "AGCAGCAGC", "ATGC"]
```

```
#Remove first sequence from the list  
SeqList.remove(SeqList[0])  
print(SeqList)
```

Output

```
["ATCATCATC", "AGCAGCAGC", "ATGC"]
```

- Replace, Add and remove items to list

Python

- Some built in list methods are

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python

```
#create a tuple with two DNA sequences
SeqTuple = tuple(("ATGCATGC", "ATCATCATC")) #note the double round-brackets
print(SeqTuple)
print(SeqTuple[1])
```

Output

```
("ATGCATGC", "ATCATCATC")
ATCATCATC
```

- Tuple - is a collection of items which is ordered and unchangeable written with round brackets
- Difference between List and Tuple

Python

```
SeqTuple = SeqTuple.append("ATGC")  
print(SeqTuple)
```

AttributeError: 'tuple' object has no attribute 'append'

- Adding, removing and replacing an item is not applicable to tuple.

```
SeqTuple = tuple(("ATGCATGC", ["ATCATCATC", "ATGC"]))  
print(SeqTuple[1])
```

Output

```
['ATCATCATC', 'ATGC']
```

```
SeqTuple[1][0]= "ATGC"  
#but it works to change the items in the list  
print(SeqTuple)
```

Output

```
('ATGCATGC', ['ATGC', 'ATGC'])
```

Python

- Sets and dictionaries are the other two python collections.
- Find more details and documentation at
 - <https://docs.python.org/3/tutorial/datastructures.html>
 - <https://jpt-pynotes.readthedocs.io/en/latest/more-types.html>
 - https://python101.pythonlibrary.org/chapter3_lists_dicts.html

Python

- Create a function that prints a DNA sequence

```
#creating a function named first_function  
def first_function():  
    print("ATGCATGCATGC")
```

```
#to call, give function name followed by parenthesis  
first_function()
```

Output

ATGCATGCATGC

```
#To pass parameters  
def first_function(seq):  
    print("sequence:" + seq)
```

```
first_function("ATGCATGCATGC")
```

Output

sequence:ATGCATGCATGC

- A **function** is a block of organized, reusable code that is used to perform a single, related action.
- Parameters can be passed when we call the function

Python

- Default parameter value
- If we call the function without parameter, it uses the default value

```
#creating a function named first_function
def first_function(seq = "ATGCATGC"):
    print("sequence is " + seq)

first_function()           #call without parameter

sequence is ATGCATGC

first_function("ATGC")     #call with parameter

sequence is ATGC
```

Python

```
#Return the count of A
def BaseA(DNASeq):
    A = DNASeq.count("A")
    return A

BaseA("ATGCATGC")          #call with parameter
2

BaseA("AATGCAATGC")
4
```

- Return values – use return statement to let the function return a value

Python – Modules and Packages

- Multiple functions

```
#save the module as first_mod
def first_function():
    print("ATGCATGCATGC")

def BaseA(DNASeq):
    A = DNASeq.count("A")
    print(A)
```

- Save the file as first_mod.py
- Module is a file consisting of python codes that contains functions, variables, etc.

Python - Modules

- `import first_mod`

```
#Import the saved module.  
import first_mod  
first_mod.BaseA("ATGCATGC")
```

Output
2

- Built in modules - <https://docs.python.org/3/py-modindex.html>
- Eg: **math** – contains mathematical functions
- **`import math`**
- **`from math import sqrt`**
- **`from math import *`**
- **`import math as m`**

Python -packages

- Packages are directories (folders) that contains similar modules
- Python packages are available through <https://pypi.org>
- Commonly used python packages are,
 - Pandas - powerful data analysis toolkit - <http://pandas.pydata.org/pandas-docs/stable/>
 - Numpy - package for scientific computing - <http://www.numpy.org>
 - SciPy – library of algorithms and mathematical tools - <https://www.scipy.org>

Biopython

- Biopython is a set of freely available tools for biological computation written in python
- Python libraries and applications which address the needs of current and future work in bioinformatics
- More details,
 - <http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc4>
 - <https://biopython.org>
 - <http://www.bioinformatics.org/bradstuff/bp/tut/Tutorial002.html>
- Install Biopython: from Terminal ``pip install biopython``.

Biopython

- Create a DNA sequence and print the related details using biopython

```
#create a sequence and print the details about it.  
from Bio.Seq import Seq  
my_seq = Seq("CATGTAGACTAG")  
print("sequence %s is %d bases long" % (my_seq, len(my_seq)))  
print("reverse complement is %s" % my_seq.reverse_complement())  
print("protein translation is %s" % my_seq.translate())
```

Output

```
sequence CATGTAGACTAG is 12 bases long  
reverse complement is CTAGTCTACATG  
protein translation is HVD*
```

- Bio.seq is a module that contains functions to identify the related information like complementary sequence, protein, etc.

Biopython

Biopython has number of methods that work just like those of a python string

```
#create a sequence and print the details about it.  
from Bio.Seq import Seq  
from Bio.Alphabet import generic_dna  
my_dna = Seq("AGTACACTGGT", generic_dna)  
print(my_dna)  
print(my_dna.find("ACT")) #finds position of first ACT  
print(my_dna.count("A"))
```

Output

```
Seq('AGTACACTGGT', DNAAlphabet())  
5  
3
```

Python - exercises

1. Create the complementary sequence for the given DNA sequence.

```
#creating complementary sequence
DNASeq = "ATGCATGC"
Step1 = DNASeq.replace("A", "T")
Step2 = DNASeq.replace("T", "A")
Step3 = DNASeq.replace("G", "C")
Step4 = DNASeq.replace("C", "G")

print(Step4)
```

Output
ATGGATGG

#expected output
TACGTACG

- Something(s) seems wrong in the output. What and why?

Python - Exercises

2. Write a program that will print just the coding region of the given DNA sequence (the coding sequence of this sequence is from position 1 to 10 and from position 15 to 22).

```
#print out the coding regions
DNASeq = "ATGCATGCGCATACGTGCATGCATGAGTCATGCATCGATGCATGCTGCATCGATCGTA"
Exon1 = DNASeq[1:10]
Exon2 = DNASeq[15:22]
print(Exon1 + Exon2)
```

Output
TGCATGCGCTGCATGC

#expected output
ATGCATGCGCGTGCATGCA

- Output is wrong, why?

Python - Exercises

3. Write a program that will print the coding region in upper case and non-coding region in lower case using the DNA sequence in exercise 2.

```
#print out the coding regions in upper and non-coding in lower case
DNASeq = "ATGCATGCGCATACGTGCATGCATGAGTCATGCATCGATGCATGCTGCATCGATCGTA"
Exon1 = DNASeq[0:10]
Intron1 = DNASeq[10:15]
Exon2 = DNASeq[15:23]
Intron2 = DNASeq[23:59]
print(Exon1 + Intron1 + Exon2 + Intron2)
```

Output

```
ATGCATGCGCATACGTGCATGCATGAGTCATGCATCGATGCATGCTGCATCGATCGTA
```

#expected output

```
ATGCATGCGCatacGTGCATGCATgagtcatgcatcgatgcatgctgcatcgatcgta
```


Python - Exercises

4. Check if the given DNA sequences contain equal number of base A.

```
#Compare the count of A in two different sequences
DNA1 = "ATGCATGC"
DNA2 = "ATGCATGCTGCTGCTGC"

Base_A1 = DNA1.count("A")
Base_A2 = DNA2.count("A")

if Base_A1 == Base_A2:
    print("equal")

#Another way
if DNA1.count("A") == DNA2.count("A"):
    print("equal")
```

Output
equal

Python - Exercises

5. Print reverse of the given DNA sequence

```
#Print the reverse of a sequence
seq = "ATGC"
rev = ""
length = len(seq)
while length > 0:
    rev = rev + seq[length-1]
    length = length - 1
print(rev)
```

Output

C

CG

CGT

CGTA

#expected output

CGTA

Python - Exercises

6. Compare solution of exercise 6 with the solution of exercise 5.

```
#Print the reverse of a sequence
seq = "ATGC"
rev = ""
length = len(seq)
while length > 0:
    rev = rev + seq[length-1]
    length = length - 1
print(rev)
```

Output
CGTA

Python - Resources

- Online Tutorials
 - [Codecademy](#) – this is a great free resource and introduces the principles of python.
 - [Coursera \(Python programming\)](#) – This is a great course to begin with but goes into some more advanced topics.
 - [Tutorialspoint](#) – Free online resource that introduce python basics with simple examples .

Python - Resources

- Books

- [Python for Biologists](#) – this is an excellent introduction to building python code and then applying it to simple biological problems.
- [Practical computing for biologists](#) – Again another great resource for beginners how to use python to answer simple scientific questions.
- [Python programming for biology](#) – Another great resource for beginners with simple biological examples.