BINF2111 – Introduction to Bioinformatics Computing

To function or not to function that is the question?

Richard Allen White III, PhD
RAW Lab
Lecture 19 – Tuesday Nov 9th, 2021

Learning Objectives

- Import

- Functions

- AT_content function

- Function or to not to function?

- Testing and importing

- Quiz 19

Imports

Time/date time import

```
>>> import time
>>> from datetime import date
>>> today = date.today()
>>> today
```

Output?

Imports

Time/date time import

```
>>> import time
>>> from datetime import date
>>> today = date.today()
>>> today
```

Output >>> datetime.date(2021, 11, 9)

Today == date.fromtimestamp(time.time())
Output ?

Imports

Time/date time import

```
>>> import time
>>> from datetime import date
>>> today = date.today()
>>> today
```

Output >>> datetime.date(2021, 11, 9)

Today == date.fromtimestamp(time.time())
Output True

Imports

Time/date time import

```
>>> import datetime
>>> datetime.datetime.now()
>>>
```

Output?

Imports

Time/date time import

```
>>> import datetime
>>> datetime.datetime.now()
>>>
```

Output >>> datetime.datetime(2021, 11, 8, 20, 25, 48, 294368)

In this example you import the datetime **class**. You have to explicitly call the datetime **type** and the **function** now.

Common Imports

These are some commonly used imports

import zlib
Allows you to compress your data

import math
Allows for mathematical functions

import re
string matching patterns (will talk about more in regular expressions)

Common Imports

These are some commonly used imports

import sys

Creates a list of the system arguments.

import os

Allows you to send commands to the shell.

Import argparse

makes it easy to write user-friendly command-line interface

Common Imports

These are some commonly used imports

import numpy

Comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

import pandas

For data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Common Imports

These are some commonly used imports

import scikit-learn

machine learning library, tools for predictive data analysis

import matplotlib, seaborn, plotly

Plotting imports

Functions

There are two kinds of functions in Python.

1) **Built-in functions** that are provided as part of Python - input(), type(), float(), int() ...

2) **Custom Functions** that we define ourselves

In Python a function is some reusable code that takes argument(s) as input, does some computation, and then returns a result or results
As we showed earlier you can import functions that have been previously written

AT vs. GT content function

```
#!/usr/bin/python3
my_dna = 'ACTGATCCATATATATTTT'
length = len(my_dna)
a_count = my_dna.count('A')
t_count = my_dna.count('T')
g_count = my_dna.count('G')
c_count = my_dna.count('C')
at_content = (a_count + t_count) / (length)
gc_content = (1 - at_content)
print('AT content is: ' + str(at_content))
print('GC content is: ' + str(gc_content))
```

AT vs. GT content function

This script counts the AT content of a piece of dna. But if we want to do it later, we have to reassign the variables, and re-run.

If we write a function, we can count the AT content of any string as we encounter it or read it in, without reassigning a_count and t_count etc.

Functions

Define a function using : **def**

Call/invoke the function by using the function name, parenthesis and arguments in an expression

The first statement can be optional documentation of the function

The code block within the defined function is indented and starts with a colon :

The statement return [expression] exits the function, and can pass back an expression when used.

Example

```
def printme(str):
    "This prints a passed string into this function“
    print str
    return str
```

Functions

The function will not execute on its own, call the function to make it run.

```
#Function definition is here
def printme( str ):
    "This prints a passed string into this function“
    print str
    return str

#Calling the printme function
printme(“First call of user-defined function!")
printme(“Second call of user-defined function")
```

Once we have defined a function, we can invoke it as many times as we like

Arguments

Arguments are passed into the function as input when the function is called.
Arguments are usually variables but can be values input directly.
The argument is a placeholder for the values we want to pass to the function.

```
# Function definition is here
def printme( str ):
      "This prints a passed string into this function“
      print str
      return

# Now you can call the printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

Arguments

Arguments are passed into the function as input when the function is called.
Arguments are usually variables but can be values input directly.
The argument is a placeholder for the values we want to pass to the function.

```
# Function definition is here
def printme( str ):
      "This prints a passed string into this function“
      print str
      return

# Now you can call the printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

Arguments/Functions - AT counter

Now lets define a function that calculates A/T content.

```
def get_at_content(dna):
    length = len(dna)
    a_count = dna.count('A')
    t_count = dna.count('T')
    at_content = (a_count + t_count) / length
    return at_content

my_at_content = get_at_content("ATTTACTACATACGCGAGCGAG")
print(str(my_at_content))
```

Output?

Arguments/Functions - AT counter

Now lets define a function that calculates A/T content.

```
def get_at_content(dna):
    length = len(dna)
    a_count = dna.count('A')
    t_count = dna.count('T')
    at_content = (a_count + t_count) / length
    return at_content

my_at_content = get_at_content("ATTTACTACATACGCGAGCGAG")
print(str(my_at_content))
```

Output – 54.54%

Arguments/Functions - AT counter

```
#!/usr/bin/python3

def get_at_content(dna):
    length = len(dna)
    a_count = dna.count('A')
    t_count = dna.count('T')
    at_content = (a_count + t_count) / length
    return at_content

my_at_content = get_at_content("ATTTACTACATACGCGAGCGAG")
print(str(my_at_content))
print(get_at_content("ATCGCGCGCATCGTY"))
print(get_at_content("atatatatatcgcggcatcagt"))
print("AT content is " + str(get_at_content("AATTAGGCGC")))
```

Outputs?

Arguments/Functions - AT counter

Outputs?

0.5454545454545454
0.3333333333333333
0.0
AT content is 0.5

Function?

Arguments/Functions - AT counter

```
#!/usr/bin/python3

def get_at_content(dna):
    length = len(dna)
    a_count = dna.upper().count('A')
    t_count = dna.upper().count('T')
    at_content = (a_count + t_count) / length
    return round(at_content,2)

my_at_content = get_at_content("ATTTACTACATACGCGAGCGAG")
print(str(my_at_content))
print(get_at_content("ATCGCGCGCATCGTY"))
print(get_at_content("atatatatatcgcggcatcagt"))
print("AT content is " + str(get_at_content("AATTAGGCGC")))
```

Outputs?

Arguments/Functions - AT counter

Outputs?

0.55
0.33
0.64
AT content is 0.5

Round up with second argument?

Arguments/Functions - AT counter

```
#!/usr/bin/python3

def get_at_content(dna, sig_figs):
    length = len(dna)
    a_count = dna.upper().count('A')
    t_count = dna.upper().count('T')
    at_content = (a_count + t_count) / length
    return round(at_content, sig_figs)

test_dna = “ATCATACATACAT”
print(get_at_content(test_dna,1))
print(get_at_content(test_dna,2))
print(get_at_content(test_dna,3))
```

Output?

Arguments/Functions - AT counter

Outputs?

0.8
0.77
0.769

Arguments/Functions - AT counter

```
#!/usr/bin/python3

def get_at_content(dna, sig_figs):
    length = len(dna)
    a_count = dna.upper().count('A')
    t_count = dna.upper().count('T')
    at_content = (a_count + t_count) / length
    return round(at_content, sig_figs)

test_dna = “ATCATACATACAT”
print(get_at_content(test_dna,1))
print(get_at_content(test_dna,2))
print(get_at_content(test_dna,3))
```

Sometimes functions have default arguments. Ex. open filename is automatically set to read. You can define default arguments

Multiple parameters

More than one parameter can be used when defining a function. When the function is called, we pass the required arguments for the function (as defined) or in correct order.

```
# Function definition is here
def printInfo ( name, age ):
print "Name: ", name
print "Age: ", age
return;

# Calling printinfo function
printInfo( age=50, name="miki" )
```

```
Name:  miki
Age:  50
```

Multiple parameters

More than one parameter can be used when defining a function. When the function is called, we pass the required arguments for the function (as defined) or in correct order.

```
# Function definition is here
def printInfo ( name, age ):
print "Name: ", name
print "Age: ", age
return;

# Calling printinfo function
printInfo( name=“miki”, age=50 )
```

```
Name:  miki
Age:  50
```

Multiple parameters

More than one parameter can be used when defining a function. When the function is called, we pass the required arguments for the function (as defined) or in correct order.

```
# Function definition is here
def printInfo ( name, age ):
print "Name: ", name
print "Age: ", age
return;

# Calling printinfo function
PrintInfo(50, “miki”)
```

Name: 50
Age: miki

Return values

A function will take its arguments, do some computation and return a value to be used.

The “return” keyword is used.

A return value can be anything. Lists can be return values.

```
# Function definition is here
def sum( arg1, arg2 ):
   “Add both the parameters and return them."
   total = arg1 + arg2
   print "Inside the function : ", total
   return total

# Now you can call sum function
total = sum( 10, 20 )
print "Outside the function : ", total
```

Return values

A “fruitful” function is one that produces a result (or return value)

The return statement ends the function execution and “sends back” the result of the function

You don’t have to return a value, you can also print a value at the end of the function, or you can return nothing (void function).

Importing your own programs

Import using the from command (don't need the .py) and the function that is imported.

```
#/usr/bin/env python

from atContent import get_at_content

print
(get_at_content("ATACATATATAAACCAGGGGATACAG"))
```

To function or not to the function that is the question?

Organize your code into “paragraphs” - capture a complete thought and “define it”

Don’t repeat yourself - make it work once and then reuse it

If something gets too long or complex, break up into logical parts and “encapsulate” with functions.

Make a library of common functions that you will use over and over in bioinformatics.

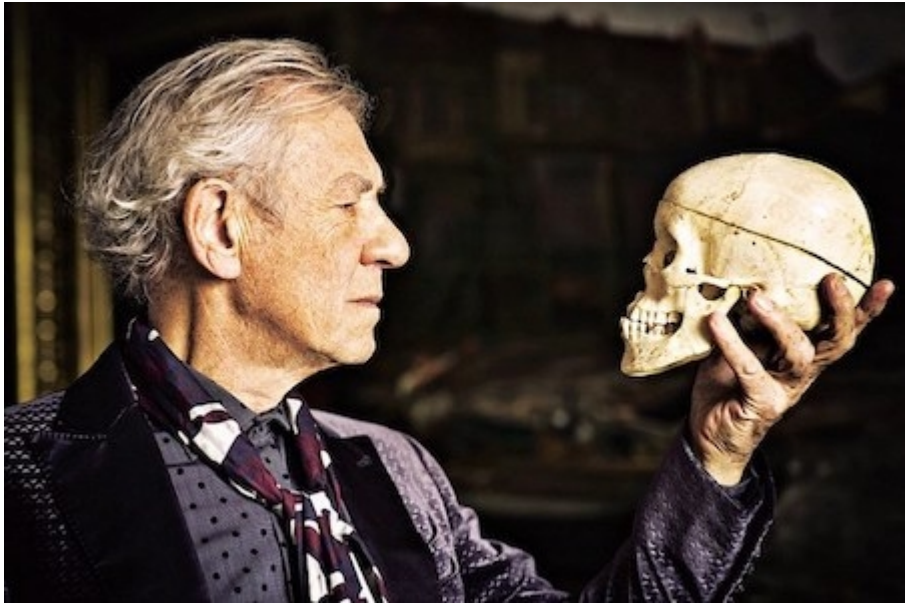
To function or not to the function that is the question?

Organize your code into “paragraphs” - capture a complete thought and “define it”

Don’t repeat yourself - make it work once and then reuse it

If something gets too long or complex, break up into logical parts and “encapsulate” with functions.

Make a library of common functions that you will use over and over in bioinformatics.



IMPORT YOUR OWN FUNCTIONS WHEN YOU NEED TO USE THEM

Read and strip

There are a few ways to open and read lines. This is one of my preferred ways:

```
fh = open(“file.txt")
sequence = ""
for line in fh:
        sequence += line.rstrip('\n')

fh.close
print(sequence)                    #without header, can add if statement (class 6)


with open(‘file.txt') as f:
        sequence = [line.rstrip() for line in f]
        sequence = “”.join(sequence[1:])            #if you have header
```

Docstrings

Docstrings allow you to print out the first string right after the defined function.

```
#Function definition is here
def printme( str ):
    “This prints a passed string into this function”
    print str;
    return;

#Calling the printme function
printme(“First call of user-defined function!")
printme(“Second call of user-defined function")

print(printme.__doc__)
```

Docstrings

Docstrings allow you to print out the first string right after the defined function.

```
#Function definition is here
def printme( str ):
    “”” This prints a passed string into this function
        I wrote this
    ”””
    print str;
    return;

#Calling the printme function
printme(“First call of user-defined function!")
printme(“Second call of user-defined function")

print(printme.__doc__)
```

Testing functions

You can test your functions using assert. If the function returns correctly, it passes, else there is an assertion error.

```
#Function definition is here
def printme( str ):
        “””This prints a passed string into this function”””
        print str;
        return;

#Calling the printme function
printme(“First call of user-defined function!")
printme(“Second call of user-defined function")
```

assert printme(“this string”) == “this string”

assert printme(“this string”) == “wrong string”

Quiz 19

- On canvas now