Scientific Programming Practical 6

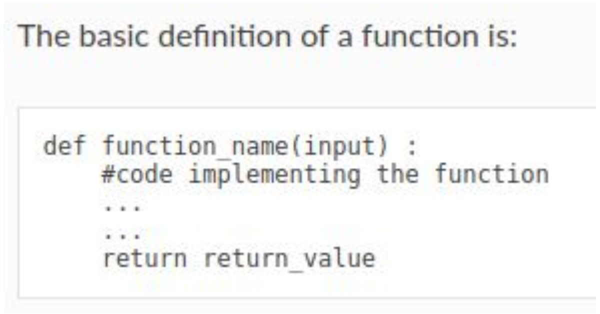
Introduction

Luca Bianco - Academic Year 2019-20
luca.bianco@fmach.it

Functions

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:

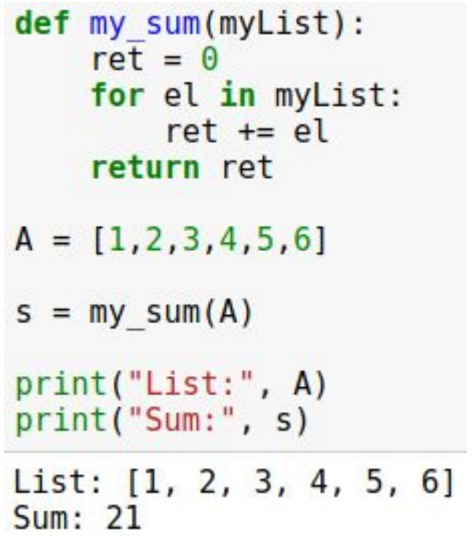
```
def function_name(input) :
    #code implementing the function
    ...
    ...
    return return_value
```

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

Functions

Example: define a function that gets a list of integers and returns its sum.



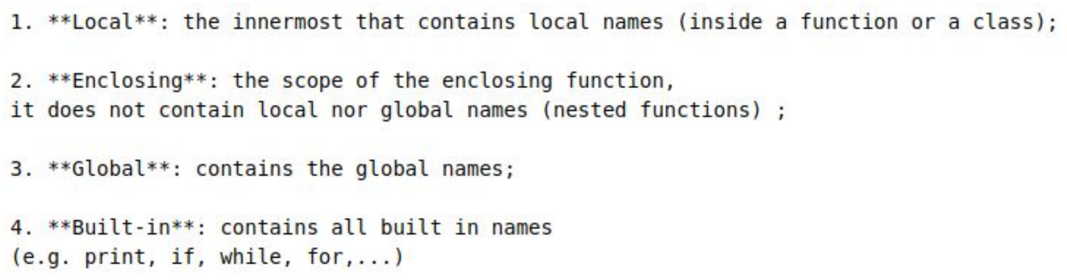
```
def my_sum(myList):
    ret = 0
    for el in myList:
        ret += el
    return ret

A = [1,2,3,4,5,6]

s = my_sum(A)

print("List:", A)
print("Sum:", s)
```

```
List: [1, 2, 3, 4, 5, 6]
Sum: 21
```

Namespace and scope

Namespaces are mappings from *names* to objects, or in other words places where names are associated to objects.

Namespaces can be considered as **the context**. According to Python's reference a **scope** is a *textual region of a Python program, where a namespace is directly accessible*



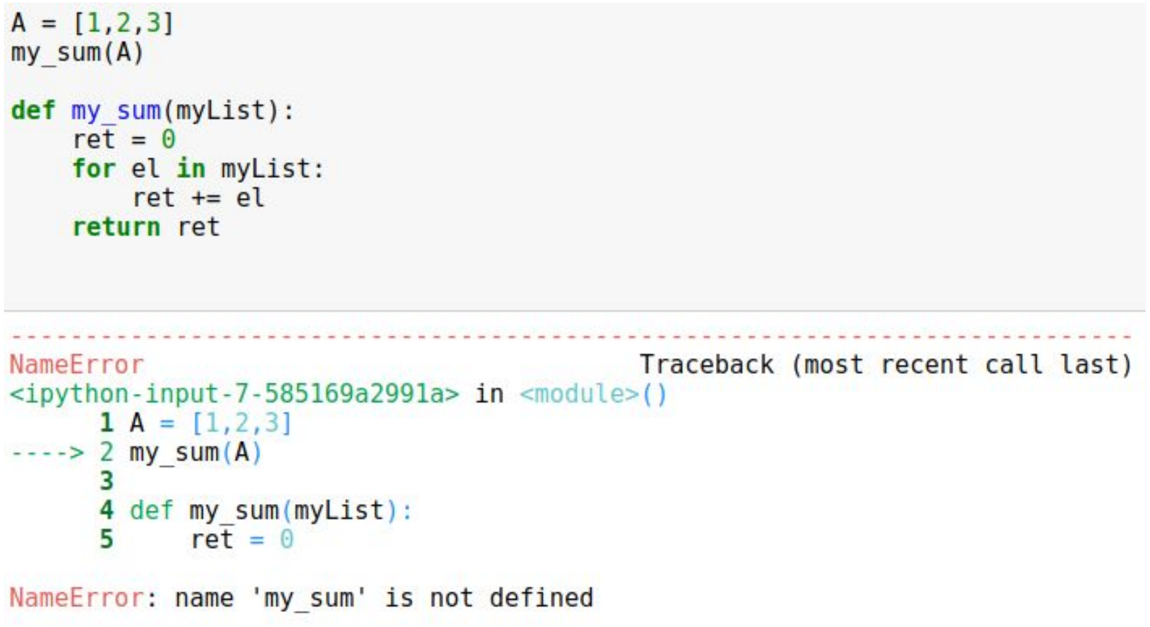
```
1. **Local**: the innermost that contains local names (inside a function or a class);

2. **Enclosing**: the scope of the enclosing function,
it does not contain local nor global names (nested functions) ;

3. **Global**: contains the global names;

4. **Built-in**: contains all built in names
(e.g. print, if, while, for,...)
```

LEGB order for finding variable

Functions

Example: define a function that gets a list of integers and returns its sum.

Importantly enough, **a function needs to be defined** (i.e. its code has to be written) **BEFORE** it can actually be used.

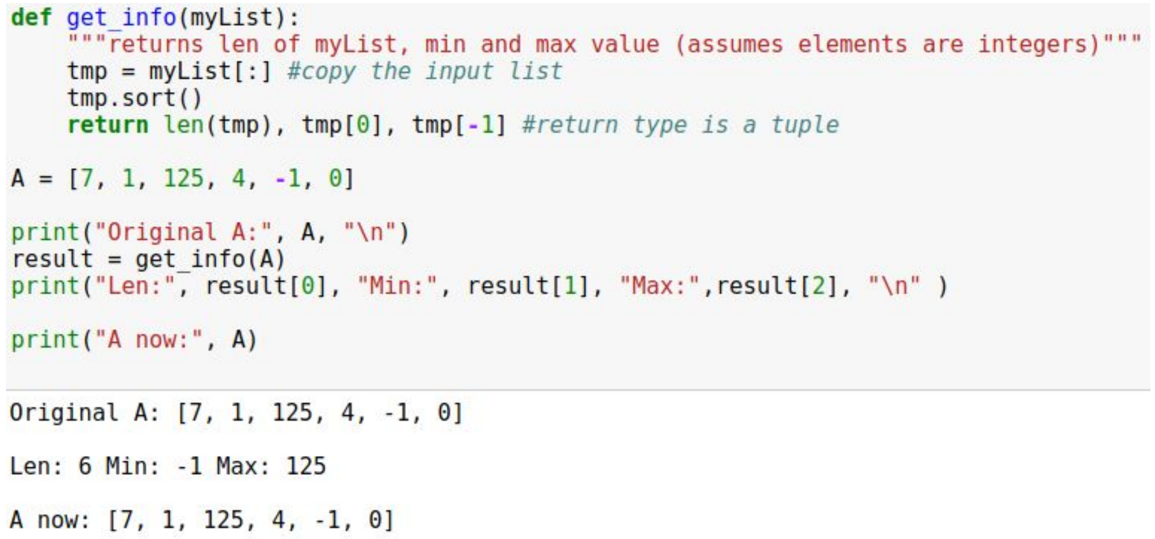


```
A = [1,2,3]
my_sum(A)

def my_sum(myList):
    ret = 0
    for el in myList:
        ret += el
    return ret
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-585169a2991a> in <module>()
      1 A = [1,2,3]
----> 2 my_sum(A)
      3 
      4 def my_sum(myList):
      5     ret = 0

NameError: name 'my_sum' is not defined
```

Functions

Example: Let's write a function that, given a list of integers, returns the number of elements, the maximum and minimum.

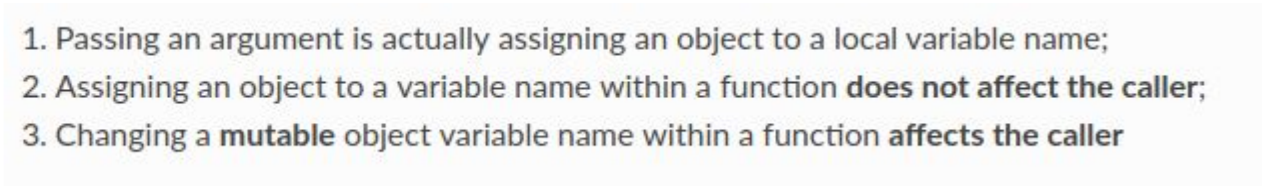


```python
def get_info(myList):
    """returns len of myList, min and max value (assumes elements are integers)"""
    tmp = myList[:] #copy the input list
    tmp.sort()
    return len(tmp), tmp[0], tmp[-1] #return type is a tuple

A = [7, 1, 125, 4, -1, 0]

print("Original A:", A, "\n")
result = get_info(A)
print("Len:", result[0], "Min:", result[1], "Max:",result[2], "\n" )

print("A now:", A)
```

```
Original A: [7, 1, 125, 4, -1, 0]

Len: 6 Min: -1 Max: 125

A now: [7, 1, 125, 4, -1, 0]
```

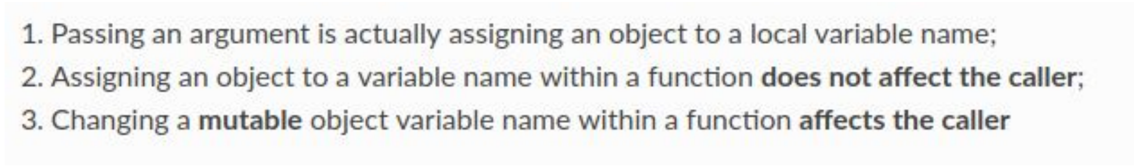
Argument passing

Things to remember

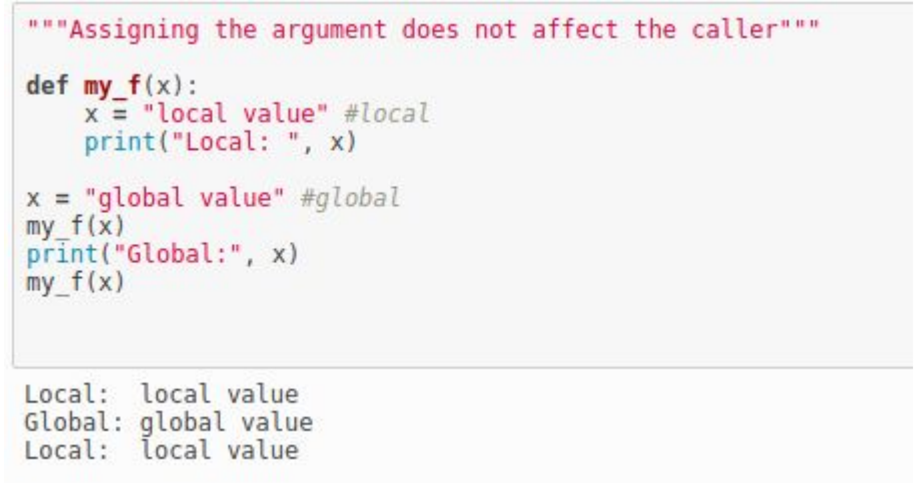


1. Passing an argument is actually assigning an object to a local variable name;
2. Assigning an object to a variable name within a function **does not affect the caller**;
3. Changing a **mutable** object variable name within a function **affects the caller**

Argument passing



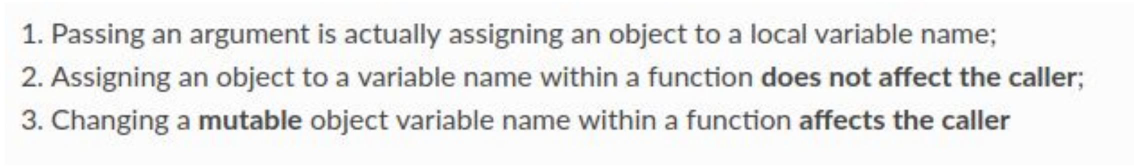
1. Passing an argument is actually assigning an object to a local variable name;
2. Assigning an object to a variable name within a function **does not affect the caller**;
3. Changing a **mutable** object variable name within a function **affects the caller**

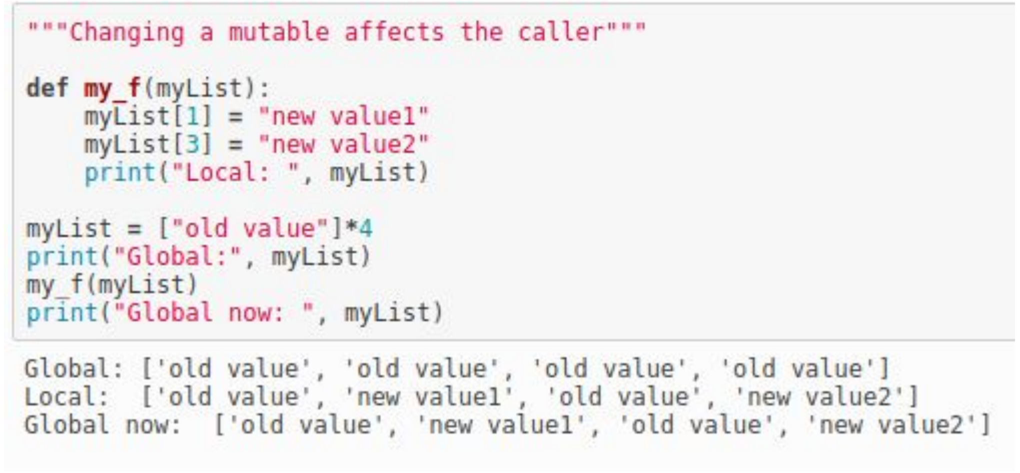


```
"""Assigning the argument does not affect the caller"""

def my_f(x):
    x = "local value" #local
    print("Local: ", x)

x = "global value" #global
my_f(x)
print("Global:", x)
my_f(x)
```

```
Local:  local value
Global: global value
Local:  local value
```

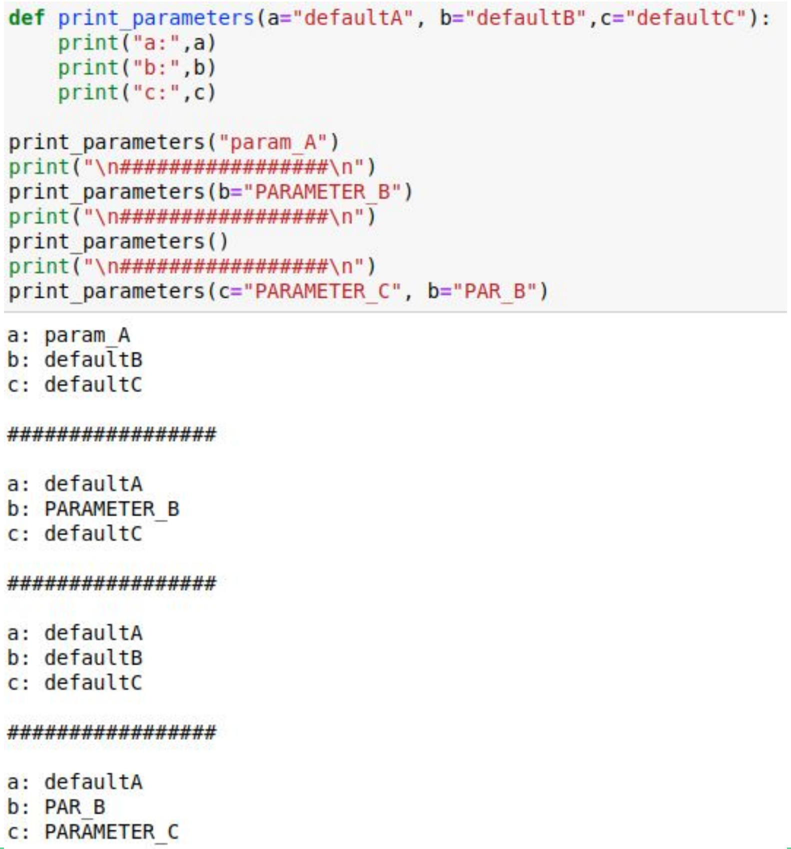
Argument passing


1. Passing an argument is actually assigning an object to a local variable name;
2. Assigning an object to a variable name within a function **does not affect the caller**;
3. Changing a **mutable** object variable name within a function **affects the caller**


```python
"""Changing a mutable affects the caller"""

def my_f(myList):
    myList[1] = "new value1"
    myList[3] = "new value2"
    print("Local: ", myList)

myList = ["old value"]*4
print("Global:", myList)
my_f(myList)
print("Global now: ", myList)
```

```
Global: ['old value', 'old value', 'old value', 'old value']
Local:  ['old value', 'new value1', 'old value', 'new value2']
Global now:  ['old value', 'new value1', 'old value', 'new value2']
```

Argument passing by keyword and defaults



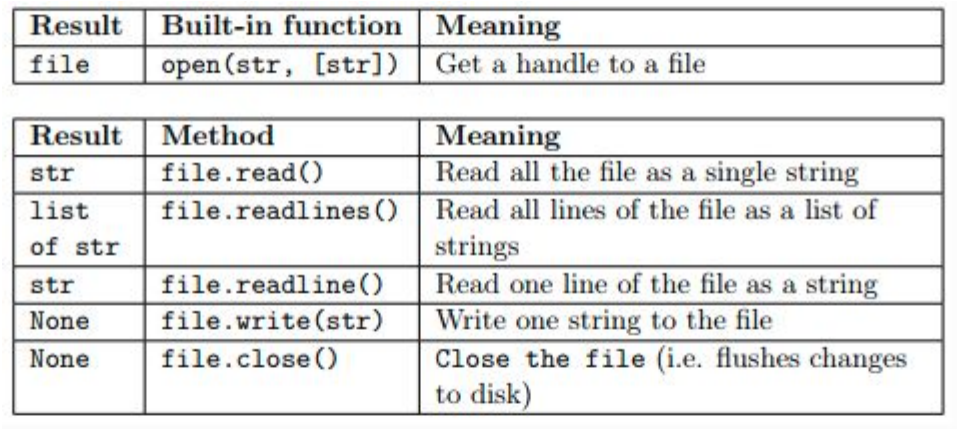
```python
def print_parameters(a="defaultA", b="defaultB",c="defaultC"):
    print("a:",a)
    print("b:",b)
    print("c:",c)

print_parameters("param_A")
print("\n#################\n")
print_parameters(b="PARAMETER_B")
print("\n#################\n")
print_parameters()
print("\n#################\n")
print_parameters(c="PARAMETER_C", b="PAR_B")
```

```
a: param_A
b: defaultB
c: defaultC

#################

a: defaultA
b: PARAMETER_B
c: defaultC

#################

a: defaultA
b: defaultB
c: defaultC

#################

a: defaultA
b: PAR_B
c: PARAMETER_C
```

File Input/Output

With files you need to perform 3 steps:

Open the file**, read/write, close**



Result	Built-in function	Meaning
file	open(str, [str])	Get a handle to a file

Result	Method	Meaning
str	file.read()	Read all the file as a single string
list of str	file.readlines()	Read all lines of the file as a list of strings
str	file.readline()	Read one line of the file as a string
None	file.write(str)	Write one string to the file
None	file.close()	Close the file (i.e. flushes changes to disk)

File Input/Output

```
file_handle = open("file_name", "file_mode")
```

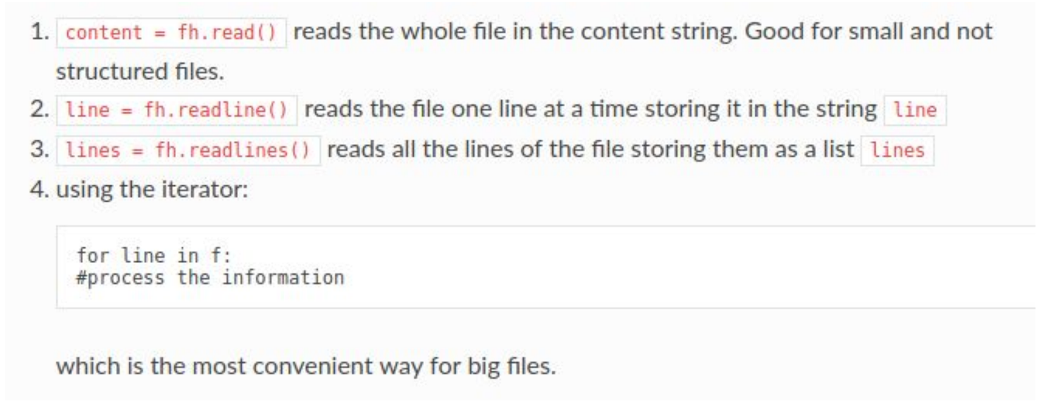
With files you need to:

Open, read/write, close

Opening mode: “r”, “w”, “a”,”b”,...

overwrites!

Read



1. `content = fh.read()` reads the whole file in the content string. Good for small and not structured files.
2. `line = fh.readline()` reads the file one line at a time storing it in the string `line`
3. `lines = fh.readlines()` reads all the lines of the file storing them as a list `lines`
4. using the iterator:

```
for line in f:
#process the information
```

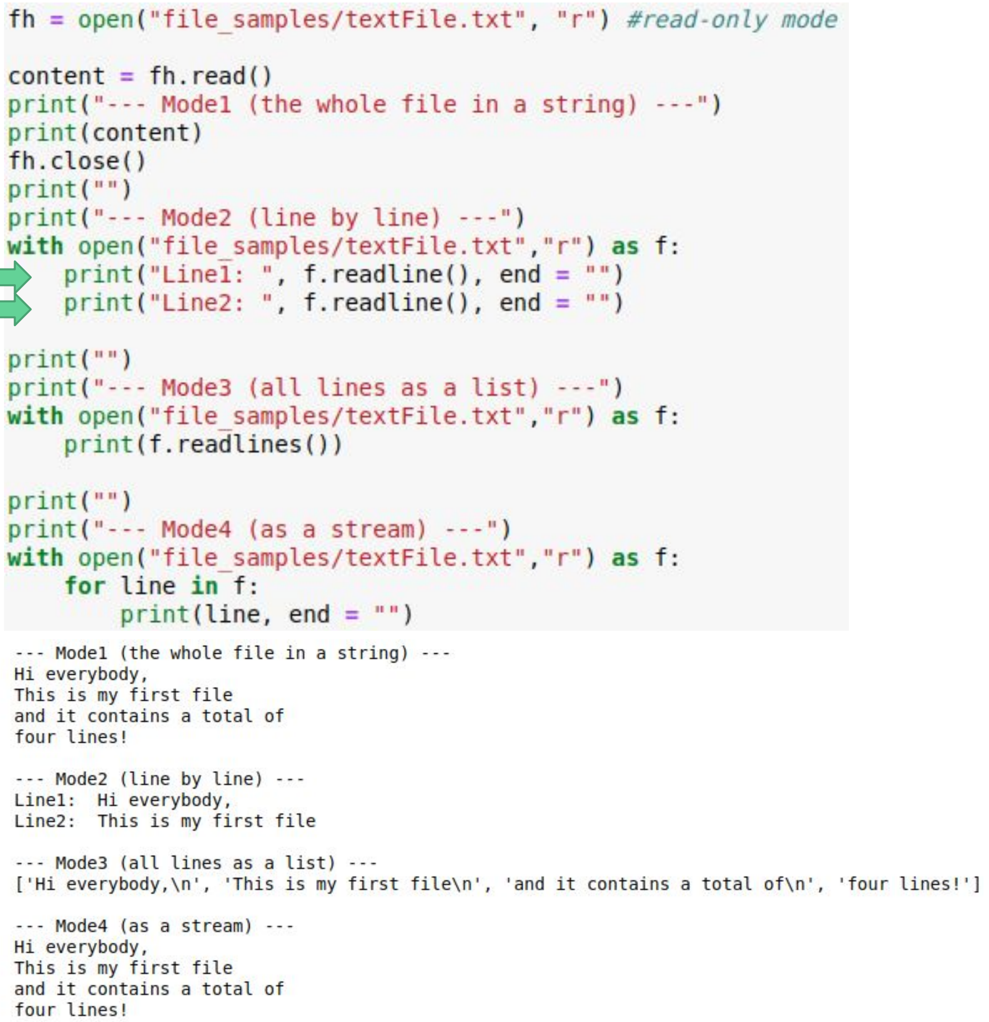
which is the most convenient way for big files.

Write

```
file_handle.write(data_to_be_written)
```

```
file_handle.close()
```

File Input/Output

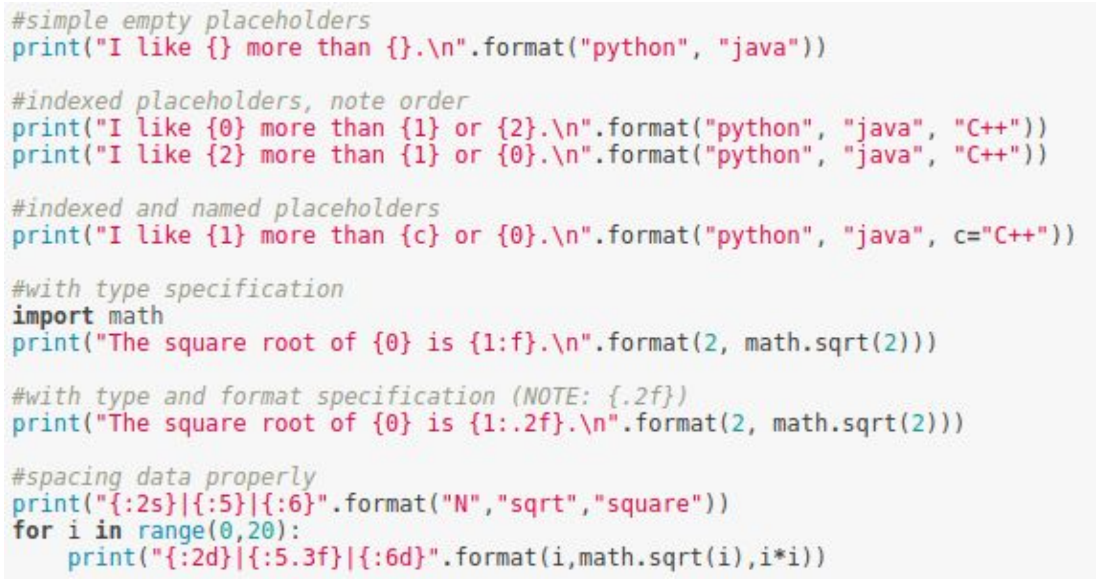


```python
fh = open("file_samples/textFile.txt", "r") #read-only mode

content = fh.read()
print("--- Mode1 (the whole file in a string) ---")
print(content)
fh.close()
print("")
print("--- Mode2 (line by line) ---")
with open("file_samples/textFile.txt","r") as f:
    print("Line1: ", f.readline(), end = "")
    print("Line2: ", f.readline(), end = "")

print("")
print("--- Mode3 (all lines as a list) ---")
with open("file_samples/textFile.txt","r") as f:
    print(f.readlines())

print("")
print("--- Mode4 (as a stream) ---")
with open("file_samples/textFile.txt","r") as f:
    for line in f:
        print(line, end = "")
```

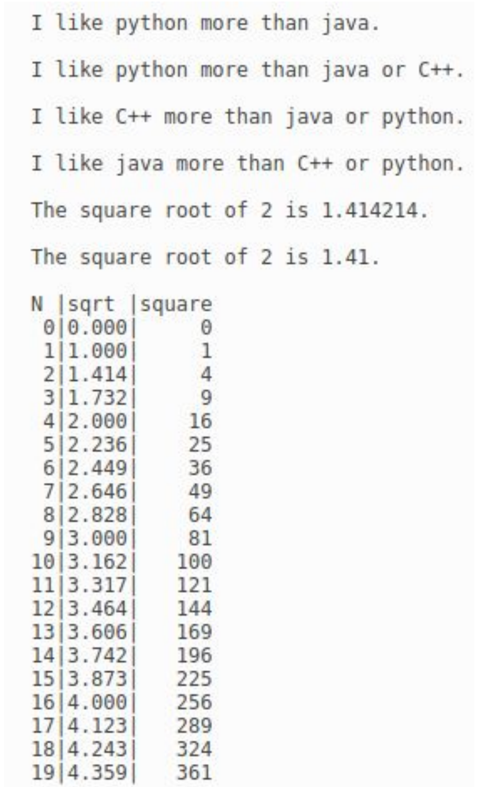
```
--- Mode1 (the whole file in a string) ---
Hi everybody,
This is my first file
and it contains a total of
four lines!

--- Mode2 (line by line) ---
Line1:  Hi everybody,
Line2:  This is my first file

--- Mode3 (all lines as a list) ---
['Hi everybody,\n', 'This is my first file\n', 'and it contains a total of\n', 'four lines!']

--- Mode4 (as a stream) ---
Hi everybody,
This is my first file
and it contains a total of
four lines!
```

more info in the
Practical6 notes...

String formatting



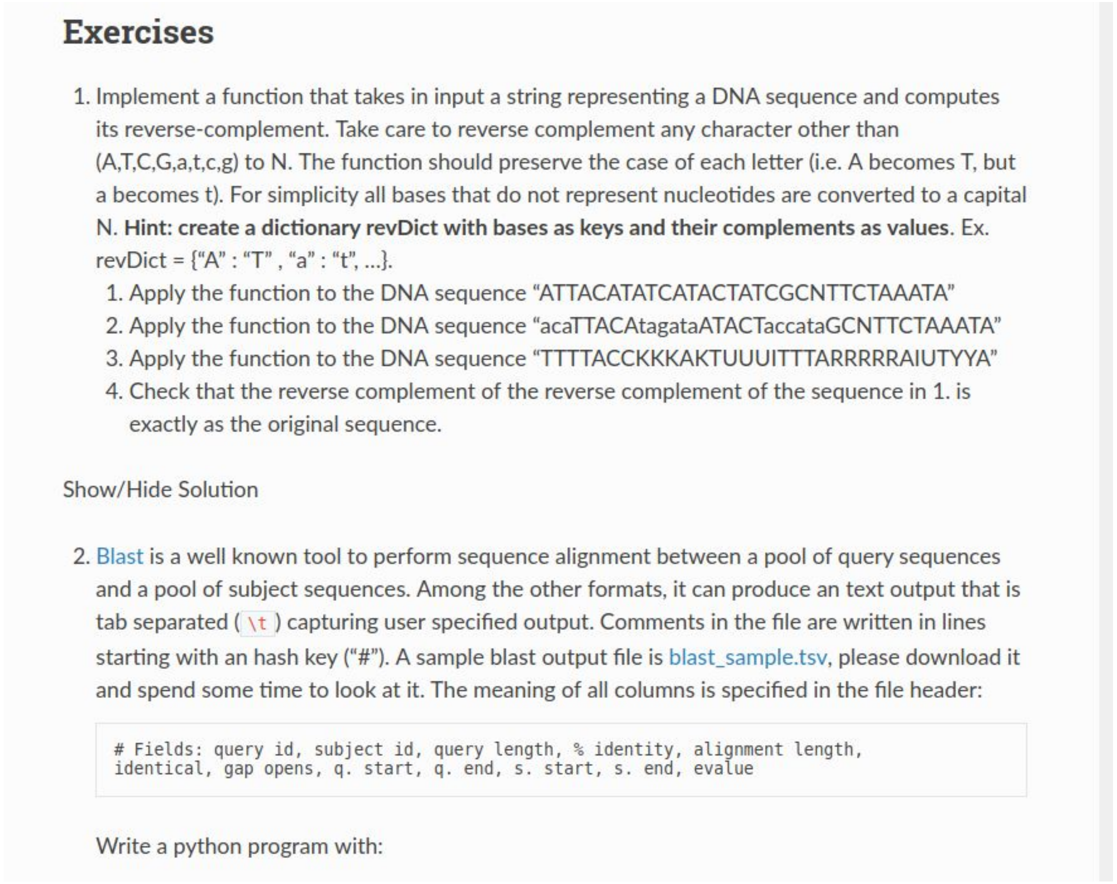
```python
#simple empty placeholders
print("I like {} more than {}.\n".format("python", "java"))

#indexed placeholders, note order
print("I like {0} more than {1} or {2}.\n".format("python", "java", "C++"))
print("I like {2} more than {1} or {0}.\n".format("python", "java", "C++"))

#indexed and named placeholders
print("I like {1} more than {c} or {0}.\n".format("python", "java", c="C++"))

#with type specification
import math
print("The square root of {0} is {1:f}.\n".format(2, math.sqrt(2)))

#with type and format specification (NOTE: {.2f})
print("The square root of {0} is {1:.2f}.\n".format(2, math.sqrt(2)))

#spacing data properly
print("{:2s}|{:5}|{:6}".format("N","sqrt","square"))
for i in range(0,20):
    print("{:2d}|{:5.3f}|{:6d}".format(i,math.sqrt(i),i*i))
```



```
I like python more than java.

I like python more than java or C++.

I like C++ more than java or python.

I like java more than C++ or python.

The square root of 2 is 1.414214.

The square root of 2 is 1.41.

N |sqrt |square
 0|0.000|     0
 1|1.000|     1
 2|1.414|     4
 3|1.732|     9
 4|2.000|    16
 5|2.236|    25
 6|2.449|    36
 7|2.646|    49
 8|2.828|    64
 9|3.000|    81
10|3.162|   100
11|3.317|   121
12|3.464|   144
13|3.606|   169
14|3.742|   196
15|3.873|   225
16|4.000|   256
17|4.123|   289
18|4.243|   324
19|4.359|   361
```

Format can be used to add values to a string in specific placeholders (normally defined with the syntax **{}**) or to format values according to the user specifications (e.g. number of decimal places for floating point numbers).

http://qcbsciprolab.readthedocs.io/en/latest/practical6.html



Exercises

1. Implement a function that takes in input a string representing a DNA sequence and computes its reverse-complement. Take care to reverse complement any character other than (A,T,C,G,a,t,c,g) to N. The function should preserve the case of each letter (i.e. A becomes T, but a becomes t). For simplicity all bases that do not represent nucleotides are converted to a capital N. **Hint: create a dictionary revDict with bases as keys and their complements as values**. Ex. revDict = {"A" : "T" , "a" : "t", ...}.
   1. Apply the function to the DNA sequence "ATTACATATCATACTATCGCNTTCTAAATA"
   2. Apply the function to the DNA sequence "acaTTACAtagataATACTaccataGCNTTCTAAATA"
   3. Apply the function to the DNA sequence "TTTTACCKKKAKTUUUITTTARRRRRAIUTYYA"
   4. Check that the reverse complement of the reverse complement of the sequence in 1. is exactly as the original sequence.

Show/Hide Solution

2. Blast is a well known tool to perform sequence alignment between a pool of query sequences and a pool of subject sequences. Among the other formats, it can produce an text output that is tab separated (`\t`) capturing user specified output. Comments in the file are written in lines starting with an hash key ("#"). A sample blast output file is blast_sample.tsv, please download it and spend some time to look at it. The meaning of all columns is specified in the file header:

```
# Fields: query id, subject id, query length, % identity, alignment length,
identical, gap opens, q. start, q. end, s. start, s. end, evalue
```

   Write a python program with: