```
#### Introduction to Python - Day 3/3 ####
# Welcome to Day 3 of the Introduction to Python workshop.  Today we will
learn how to execute Python scripts and how to make them use objects and
functions defined in modules.  We will also introduce the last type of
loops: while loops, and learn how to read and write data to files. By the
end of this day you shold be able to write and execute simple Python
scripts that read and process data stored a text file.

#%% Recap from yesterday: lists, dictionaries, loops

# strings ( additional methods: .strip() .replace(), .split() )

# simple loop

#for x in mylist:
#    print(x)

# index loops

#for x in seq[0:5]:
#    print(x)

#%% ------ WHILE LOOPS ------
# In certain scenarios you may want to run a loop as long as a particular
condition is held.  This is what a while loop does:

# while <logical expression>:
#    code here is repeatedly executed
#    as long as <logical expression>  evaluates to True

# Let's look at some simple examples of while loops.

#%% In this example we will first define acondition variable that
represents our condition.

             True

# Then we will  make while loop contingent on the value of acondition.
Inside of our while loop, we will redefine the value of acodition
variable to be False, and provide a print command stating print("This
will only print once")

while                True
    print "in loop"
                 False

print "done"
```

```
print       # prints empty line (and a new line)


# what about printing all the words of just the first sentence ?

            'This is a sentence. This is another sentence.'



       False

        0
while not
    print
    if '.' in
                True
    else
                        1


#%% ------ SCRIPTS ------
# We're now going to begin working with scripts, and running them from
either your command line (windows) or terminal (macs).  First, let's open
the command line/terminal to quickly familiarize ourselves.  From the
command line/terminal, we can open an interactive session of Python by
typing 'python' (for windows) or 'python3' (for macs).  When doing this
you should see that the blinking cursor is now preceded by three greater
than symbols.  This mean that you are in an interactive python session.
That is similar to what we are doing now, where our code is executed
within the console window of the Spyder program.  Note, all of the online
tutorial material is designed such that you are running python through
the command line/terminal.

# Now, we are going to exit the command line/terminal python session by
executing 'exit()' (you can also press ctrl-D), and will instead run a
python script from the normal command line/terminal environment.

# Next, we are going to write our first script by converting our
'reverseComplementIndex' function into a separate script.  So, please
open a new file within Spyder, copy there the function definition and
call the function passing it, as an argument, a variable initialized with
some test sequence.  Once the script is ready,save it as
'reverseComplement.py' file.

# Once the script isready we must save it to a convenient location (your
desktop ? ) using some informative name (say, reverseComplement.py).
After the file is saved, open the terminal or command line (depending on
```

```
your OS), and run the script by typing:  python reverseComplement.py

#%% Running scripts within Spyder


#%%  ------ MODULES ------
# These are Python (.py) files that contain one or more functions which
we can import into our program or the interactive environment.  We can
import modules in several ways.
#   (a)  We can import the module using only its name, but this requires
we precede the call to every function in the module with the name of the
module.
#   (b)  We can import only one function from the module.
#   (c)  We can import every function from the module.
#   Options (b) and (c) mean that we no longer need to precede the
function names with the of the module from which they came.  However,
option (c) could be dangerous if the module is very large.  Examples are
below.

# We will try importing from the math module.
# (a)

#import math

#%% Now we can use math functions like
# math.sqrt()

#%% math.exp()


#%% math.log()

#%% math.pi

#%% math.isclose
# this function addresses comparision two foating point numbers I have
mentioned on two days ago.

#from math import isclose
#math.isclose( 1.0,1.0+1e-15)

#%% Accessing command line arguments: sys module
# use sys.argv

#%% Option (b)  Now instead we will import just the square root function.

#from math import log
```

```
#%% Now we should be able to use the sqrt() function without the 'math.'
preceding it.

#%% Option (c)  Now instead we will import everying from the math module.

#from math import *

#%% Now we should be able to use anything from the math module without
the 'math.' preceding it.


#%% ----- CUSTOM MODULES ------
# There is nothing particularly special about modules. They are just
files with Python code. Frequently they contain definintions of functions
or user defined types (classes). We will work with a simple modules
defining functions. Advance Python workshop will demonstrate how to
define a new class.

# Using your custom module can be tricky if the file is not located in
one of the directories Python searches when looking for modules. You can
see (and modify) the list of directories Python searches through by
examining the sys.path variable  defined in the 'sys' module.

#import mymodule

#%% I want to add my desktop to the list of commonly searched paths.


#%% Now we can load our function defined in mymodule.py module.


#%% And now we should be able to call any function defined there .


#%% File input and output.  A very common practice in programming is the
need to open and examine the contents of a file and/or write our own
output to a new file (or append new output to an existing file).  We will
go over a few quick examples of doing this for text files (.txt).

# To read from a file we have first open it with the open() function,
passing "r" parameter to let Python know that we want to read from it. We
can now inspect contents of this file by using the read methods
associated with the file object.

#file.read() will read entire (remaining) file. It will return the
contents of the opened files as a list of lines. Be careful when using it
```

```
- the file you are trying to read might be very large!!!
#To read a file one line at a time we can simply loop over open file
#Once we are  done reading the file should be closed by calling close()
method.

#let's read a sequence stored in mysequence.seq file

#myfile = open("C:/Users/lukasz/Desktop/Intro to Python/mysequence.seq",
"r")    # "w" "a"

#seq = ""

#for ln in myfile:
#    seq = seq + ln.strip()

#myfile.close()

#seq = seq.replace(' ','')

#print(seq)


#%% with ... as: It is easy to forget to close a file. Python offers a
statement that automatically closes the file

#seq = ""

#with open("C:/Users/lukasz/Desktop/BIGSummer/Intro to Python/
mysequence.seq", "r") as myfile:
#    for ln in myfile:
#        seq = seq + ln.strip()

#seq = seq.replace(' ','')

#print(seq)

#%%Frequently, when reading text files it is necessary to modify incoming
data to fit our needs. Let's say we would like to get our sequence a
single string of uppercase characters with no spaces.


#%% Writing to a file
# The open() function with the "w" parameter tells Python to make a new
text file with the name "filename.txt".

#seq="ATGCATCCGTACGT"
```

```
#with open("C:/Users/lukasz/Desktop/Intro to Python/mysequence2.txt",
"a") as myfile2:
#    myfile2.write(seq + '\t' + seq +'\n')

# Let's write our cleaned up sequence to a file. Todo it we can just
replace the print command(s) with file.write() commands. We also have to
rememnber to close the file (or use with ... as... statement)

#%%Appending to a file. Suppose you don't want to overwrite an existing
file, but only add new contents to it.  This works just like the write
function, only we change the "w" argument to "a", for append.


#%% Let's try to read another file. "codon-table.txt" lists codons
together with corresponding aminoacids. Let's read it into a dictionary.
split() method can be used to split lines into a list of individual non-
white space strings.

#aa_single = {}
#aa_long = {}

#with open("C:/Users/lukasz/Desktop/Intro to Python/codon-table.txt",
"r") as myfile:
#    for ln in myfile:
#        ln = ln.strip()
#        cols = ln.split()
#        if len(cols) == 3:
#           aa_single[cols[0]] = cols[1]
#           aa_long[cols[0]] = cols[2]
#
#print(aa_long)

#%% ------ FINAL PROJECT ------
# For the final activity: Use what we have learned during last three days
to read a file (SARS-N.seq) with a DNA sequence corresponsind to a
fragment of SARS virus genome (NOTE: SARS genome is encodede by RNA) and
find out what are the seqwuences of proteins it might encode. You will
need to add some code to remove  spaces and new line ('\n') characters
that appear in the file. You will also have to skip the first line
(header) of the file. Once the DNA sequence is ready to process you will
have to iterate over 3 letter substrings (codons) and use translation
table (after reading it from codon-table.txt file) to infer which
aminoacid or stop codon each substring correspond to. Note, that
translation might start in 3 diffrent 'reading frames'  each shifted by
one position. It is also possible that the complementary DNA strand is
translated. How will you handle these cases ?
```

NOTE: This is, in principle, the core functionality provided by the https://web.expasy.org/translate/ Web page. You can use it to verify the output of your program.