```
#### Introduction to Python - Day 2/3 ####
# Welcome to Day 2 of the Introduction to Python workshop. Today we will
continue with strings and then introduce lists and dictionaries.

#%% Strings recap

#%% ------  INDEXING  ------
# Very often we are interested in accessing individual character within a
string.  In order to do this, we need to specify its position (or
"index").  Let's define a string so we can practice indexing its
elements.


#%% To access a single character, type the name of the string followed by
a pair of square brackets, and a numerical value that represents the
position of the the character along the length of the string.

# Try indexing for the third character in your string.  Which letter do
you expect Python to return?


# Did you get what you expected?

#%% Python starts its counts at 0, not at 1.  You must always keep this
in mind when indexing in Python.  And whenever you start working with
another language, you should always ask "Is this language 0-indexed?"

# Let's try some more indexing.  Ask Python for the 0th element in your
string.


#%% find:  We can ask Python for the first index corresponding to the
location of particular elements in a string.  Again, wih this function we
must place the element of interest within the parentheses.  Try asking
Python for the location of the first instance of the element 'A' in your
string.

#mystring = 'abcd'
#mystring.find('c')

#%% Ask Python for an index that is larger than the length of the string
your defined.


#%% Negative indexing:  You can also index by negative values.  In
Python, indexing by a negative value means you are starting from the end
of the string.  The counting is awkward though.  An index of "-1"
```

```
represents the first element from the end of the string (we can't negate
0).

# Try negative indexing your string by -1


#%% Try negative indexing your string by -3


#%% Indexing by Ranges:  You can also ask for characters corresponding to
a continuous range of indices (so called 'slice').  To do this, we use
the colon as an operator (":"). The operation is sometimes called
'slicing'.
# Try indexing your string from the second element to the fourth.  Which
letters do you expect to get?

# Which letters did you get back?  What does this mean about the second
value in the range?


#%% What happens if you skip one or both of the numbers in your slice ?


#%% Indexing by range with step:  We can also index by a continuous range
of values that skips every other, every 2, every 3, etc. values.  We do
this by using a second colon and specifying interval.  So, the index
1:8:2 would represent, "index from the second element to the ninth, but
return every second entry" or more specifically "return the second,
fourth, sixth and eighth elements".  Try this command out on your string.


#%% What happens if the interval is negative ?


#%% Now let's make a somewhat useful function.  This new function will
calculate the GC content of a string sequence.  Specifically, it should
count the occurences of 'G' in the sequence, and add that number to the
occurences of 'C' in the sequence.  The function should finally divide
the sum by the total sequence length, and return the result.


#%% Now let's use our function.


#%% ------ SIMPLE LOOPS ------
# Sometimes we need to perform the same task on every character of a
string. In these instances, we use loops.
```

```
# The general structure of a for loop is the following
# for current_character in my_string:
#       code that is executed with loop

# This loop will run as many times as there are characters in my_string,
each time setting the value of current_character to the consecutive
character of the string.  Let's try to write a loop that prints, one at a
time, all the characters in a string.

#%% Let's print a string, one character at a time


#%% String practice: reverse complement function
# Let's try to write a function that takes as an argument DNA sequence
and returns its reverse complement. For example, 'ATGACCAGG' input should
result in 'CCTGGTCAT'

#def reverseComplement( sequence ):
#   revcom = ""
#     ...
#   return(revcom)

#%% Anyone got it right ?


#%% ------ COMPOSITE DATA TYPES: LISTS ------

# String can be thought of as special case of a generic 'list' object
that is composed of a series of simple(r) objects. In case of strings
these are just single characters. In case of a generic list each element
can be of a different type, even itself another list.  To define a list,
we use the square brackets.


#%% We can also coerce a string into a list


#%%  As with strings, we can access elements of a list by indexing. This
is because strings are essentially  lists, just with all elements of the
same type, and with some Python interpretation happening 'under the
hood'.  Let's make a new list of nucleotides and practice indexing.


#%%  Now index list
#%% Note that we can change elements of the list using assignment.

# it is not possible for strings - strings are 'immutable'

#%% ------ LIST METHODS ------
# There are many functions (called methods) that are designed for
manipulating lists.  To see a list of them, run 'dir()', with the name of
a list as the argument.


#%% append:  The append method can be used to add elements to the end of
the list.


#%% index:  The index method can be used to return the list index
corresponding to the element provided.  Again, this method only
identifies the first instance of the matching element.


#%% sort:  The sort method can be used to sort the elements of the list.
Some assumptions are made regarding the correct ordering - numbers are
sorted as... numbers, strings are sorted alphabetically, other types
might have their own specific rules.


#%% Sort method may sometimes yield somewhat surprising results. How is a
list of integers [1, 50, 111, 2, 5, 7] sorted ?

#%% And what about the same numbers but listed as their string
representation  ["1", "50", "111", "2","5", "7"]  ?

#%% Sorting lists of elements of mixed types, in general, will not work.


#%% ... vs sorted:  Sometimes we might want to keep the original list....

#%% remove:  The remove method can be used to remove elements from the
list, as identified by their value (not by their index).  Again, for
repeated elements, this only removes the first instance of the element.


#%% pop:  The pop method can be used to remove elements from the list, as
identified by their index.


#%% len:  As before, len is not a method (it is not called by
listname.len()) but instead is an inherent function in Python called by
```

```
len(listname).  len will tell us the number of elements in the list.


#%% ------ INDEX LOOPS ------
# An alternative to simple loop that we introduced above, we can
explicitely use index to retrieve individual elements of a list. To do so
we,first, need to use range() function that generates a sequence of
idices. When used as 'range(start_integer, stop_integer)' it defines a
sequence of integers from start_integer to stop_integer - 1 so for a
string of length len(my_string) we have to use 'range(0,len(my_string))'.
Let's try it...

#To loop over all idices of my_string

#for current_index in  range(0, len(my_list)):
#   current_element = my_list[current_index]


#%% Now, if we use a third entry in the range() function, then the
returned list skips integers by that value.


#%% In this next example, we'll define a counting variable and make the
while loop test its value.

# How can we get the final number to print?
# What happens if we accidentally generate an infinite loop?
# Can we have while, for, if, else, elif all in one statement/function?


#%% --- Reverse complement: index version ---
#Let's modify our reverseComplement function so that it uses explicit
indices

#def reverseComplementIndex(sequence):
#...


#%% ------ COMPOSITE DATA TYPES: DICTIONARIES ------
# Dictionary (sometimes referred to as 'hash table') is another object
type in Python.  They appear in many (but not all) modern programming
languages. Dictionaries are similar to lists, but instead of having
indeces and values, they have 'keys' and values.  Keys can be strings or
numbers, and serve as unique identifiers for their paired value. Like
lists, the values can be anything (e.g. strings, lists, or even another
dictionary).
```

```
# Let's define a short dictionary and practice indexing it by its keys.
To define a dictionary, we use curly brackets instead of square brackets
or parentheses.


#%% Call the name of the dictionary to view its elements


#%% Index the dictionary by any one of the keys.


#%% What happens if key is not present in the dictionary ? How to test if
key is there?


#%% Add a new key:value pair to the dictionary


#%% Call the name of the dictionary again to view its elements.


#%%--- Final function of the day ---
# ReverseComplement function so that it uses a dictionary

#def reverseComplementDictionary(sequence):
#    complement = {"A":"T","T":"A","C":"G","G":"C"}
#...
```