```
#### Introduction to Python - Day 1/3 ####

# Today we will go over the format of the workshop, Spyder, IPython and
Python, variables and assignments, mathematical and logical operations,
variable reassignments, floats and integers, object types and coercion,
text (string) operations and, methods, if/elif/else statements and
defining functions.

# This workshop aims to teach you some of the basics of coding in Python.
Note that QCB Collaboratory offers other workshops focusing on different
aspects of Python (graphing and data analysics, digitial image proessing,
etc.) - you might find current list of workshops on the QCB Workshops
page: https://qcb.ucla.edu/collaboratory-2/workshops/

# Each day the workshop is divided into two parts.  The first part will
introduce different aspects of Python.  During this part you will be
following instructions in the .py files (or the instructor) and typing
code along with the instructor.  In the second part will let you use the
new techniques.  Each day I will e-mail you copies of the tutorials with
my own code written into the blanks.  But, don't let that demotivate you
from typing anything.  Most of programming is troubleshooting, so I
encourage you to type along, and we will address any questions and errors
as they arise.  My own coding results in errors every single day, it is a
natural part of programming.


# So, what is Spyder and how does it communicate with Python?  The Spyder
Integrated Development Environment (IDE) is a computer program that has
both IPython and a text editor.  You are reading from the text editor.
# IPython is an interpreter for Python.  It is a computer application
that allows you to execute Python commands and programs.
# Python itself is a programming language.  It is the language that you
write computer programs in.
# You can think of Python as the gramar and style with which you write
code, IPython then reads, interprets, and executes the code (hence being
called an interpreter), and Spyder is software that allows to do all of
these things at once in a structured environment.
# Within Spyder, we have the text editor (the window in which you are
currently reading text) the variable/file expolorer and help menu on the
upper right, and the IPython console in the bottom right.
# We can execute code both in the console directly, or by running chunks
of code (cells) from the text editor file, or even the entire text editor
file itself.  Spyder does not run code line by line.


#%% '#%%' Defines a cell.  With each cell we can write a block of code
that can be executed independently of the other cells using hot keys (or
by clicking on the right pointing green arror button).  This gives us a
natural way to set up a tutorial, where every block of code that we want
to run is entered into a separate cell.  This may seem arduous at first
(it is not how Python is inteded to be run), but hopefully it will pay
off by providing the tactile experience of programming.  As we learn more
we will transition to more standard way of running Python programs that
are stored as files ('Python scripts')

# Separately, a single pound symbol will define a 'comment', or
'commented code'.  This is code that the interpreter will not read, and
hence will not be executed, allowing you to narrate and describe what you
code does.  It is very important that you develop a healthy habbit of
ALWAYS COMMENTING YOUR CODE.  We do this so that when sharing our code
with others, they can quickly understand what the code does.

#%% To 'execute' a cell in Spyder, press Ctrl+Enter.  Try it in this
cell.  Nothing should happen because this cell is commented.  To execute
a cell and advance to the next cell, press Ctrl+Shift+Enter

#%% Excellent.  Now we will execute our first line of actual Python code.
Execute this cell of code, which contains the following 'print()'
function, and the argument 'hello world'.

#print('hello world')

# In the console you should see the input to 'print('hello world')', and
the output 'hello world'.

#%% ------ ASSIGNMENT OF SIMPLE VARIABLES ------
# The basic variables that are common across many languages are strings,
integers, floats, and booleans.  Strings are collections of letters,
numbers, and punctuation.  Integers are the numerical integers (including
positives and negatives).  Floats are integers with decimal values.
Booleans are the values 'True' or 'False'.  The use of 'True' and 'False'
in programming is fundamental.  Other variable types do exist, and we
will explore them, but they are particular to Python, where as those we
have just mentioned are common to all interpreted languages (e.g. R and
Matlab).

#%% Assign the string "this our 1^st string" to the variable named
"sometext"

#sometext  = 'this our 1^st string'

#%% Now tell Python to print the value of the variable "sometext" using
the print() function.
```

```
#print( sometext )
#print( 'sometext')


#%% Now assign the value of 'True' to the variable name "aboolean"

#aboolean = False

#%% Now tell Python to print the value of "aboolean".
#print( aboolean )

#%% Assign the integer 42 to the variable name "someinteger"

#someinteger = -42

#%% Now tell Python to print the value of the variable "someinteger"
#print( someinteger )

#%% Assign the floating point number 4.2 to the variable name "somefloat"

#somefloat = -4.2e2

# Now tell Python to print the value of the variable "somefloat"

#print( somefloat )


#%% Notice that if you type only the names of variables, and that is the
final piece of code in a cell, or script, or console entry, then the
value of the final variable in that group will be displayed.

#somefloat

#%% Note that only the last value will be printed out

#anotherfloat = -4.2e-8

#somefloat
#anotherfloat


#%% ------ FLOATS VS INTEGERS ------
# Here we provide a few quick comments on differences between floats and
integers.
# As mentioned before, integers are the counting numbers, can be positive
and negative, and never have either decimals or commas. In Python 3 there
is no limit on the magnitude (absolute value) of integers. Note, that
```

```
this is not the case for other programming languages and is related to
how the numbers are represented in the computer memory.

#biginteger = -44444444444444444444444444444


#%%# Floats on the other hand are expressed with decimal point or in
scientific notation.  "Float" is short for floating point number, where
the phrase "floating point" expresses the fact that each floating point
number can be thought of as composed from two integers:  mantissa
represents significant digits of a floting point number and exponent
determines the position of the decimal point that can 'float' along the
mantissa. Note, that, in contrast to integer variables, the magnitudes
of, both, mantissa and exponent are bounded.  This might lead to
truncation and roundoff errors. Thorough discussion of the details of
floating point number representation and of rounding error (aka  "machine
epsilon", a fundamental quantity for numerical computing) is beyond the
scope of this workshop.  To demonstrate the nature of potential problems
we can:

#bigfloat= -4.444444444444444444444444444e999
#bigfloat

#%% (1) Print the results of operations 1 + 1e-1,  1 + 1e-15 and 1 +
1e-16.  What do you expect the answers to be?  What are they really?

    1   1e-30
print

#%% (2) Print the results of operations 1.200000e8 + 2.5000e-7 and
1.200000e8 + 2.5000e-8 and 1.200000e8 + 2.5000e-9.  What do you expect
the answers to be?  What are they really?

print 1 200000e8   2 5000e-7
print 1 200000e8   2 5000e-8
print 1 200000e8   2 5000e-9


#%% ------ EXPRESSIONS: MATHEMATICAL OPERATORS ------
# Here we practice using different mathematical operations.  On top of
the addition we performed above, all other basic mathematical operations
are available in Python.  These are (with their respective commands):
addition +, subtraction -, multiplication *, exponentiation **, division
/,  and the modulo operation (or remainder) %. Let's try them all.

#%% Addition
print 1+2
```

```
#%% Subtraction
print 1-2

#%% Multiplication
print 2 5

#%% Exponentiation
print 2  10

#%% Exponentiation
print 2  0 5
    2  0 5
print    2
print 2   1
print 2e-1


#%% Division
print 5 0 2 0
print 5  2

#%% Remainder (modulo)

print 555551 3

#%% Expressions can be combined and used in assignments

    10
    2 0
    2 8
print

#%% As a reminder,  assignments are NOT mathematical equations - left
side must be a variable

#a + b = b + 7

#%% ------- REASSIGNMENT OF VARIABLES ------
# Here we demonstrate the idea of reassignment.  That is, assigning a
value to an existing variable.  This may seem trivial, but it will prove
to be an essential tool later.
#

#c = 1
#print(c)
```

```
#c = 2
#print(c)

#c = 3
#print(c)

#%% The value of the right side of assignment is evaluated before storing
the result on the left side
#c = c + 1
#print(c)

#%% ------ WORKING WITH NUMBERS, TYPES, AND COERCION ------
# Here we introduce a few functions that can be useful when working with
collections of numbers.
# Functions look like in a math textbooks

#  function_name( argument1, argument2, ...)

# although they are limited to performing math operations. A function
named  'print' just prints the arguments

#c =2.45
#print( 'Value of c=',c)

# You may be interested in the maximum, minimum, and absolute values of
numbers.  These functions exist in Python, and have sensible names.  For
example, consider the collection of numbers 5, 7, 3, 5, 8, 2.  What would
you guess is the name of the Python function to find the largest, or the
smallest number in this colection ?

#%% Maximum
#print(  max(5, 7, 3, 5, 8, 2)  )

#%% Minimum
#print(  min(5, 7, 3, 5, 8, 2)  )

#%% Absolute value:  How about the absolute value of a negative number,
like -10?

#print( abs(11.1) )


#%% Types:  We can also ask Python to tell us the type of number we are
working with.

# Integer type
```

```
#a = 23
#type(a)


#%% Float type

#b = 2.45
#type(b)

#%% String type

#s = 'mystring'
#type(s)

#%% Coercion:  There can be instances where we need to "coerce" (another
word is "cast") a type from what Python might first assume to what we
actually want.  An example of this is that Python assumes the number 10
is an integer.  But maybe we need to treat 10 as a string (it could be
part of a filename, directory, date, etc.).  To coerce a number to a
string, we use the str() function.

#a = int("123")
#print(a)
#b ="123"
#print(b)

# Try coercing the integer 10 into a string using the str() function.
How can you tell it worked?
#s = str(a)
#print(s)

#%% On the other hand, we may need to coerce a string into a float.  Try
coercing the string '10' into a float using the float() function. How can
you tell it worked?
#f = float(a)
#print(f)

#%% What about strings to integers?



#%% Floats to integers?


#%% Letters and punctuation to floats or integers?
#%% ------ EXPRESSIONS: COMPARISIONS AND LOGICAL OPERATIONS ------

# Frequently a program needs to decide what action to take depending on
the data it is given. For example, it might process numbers only if they
are positive or it might process strings only if they start with 'ATG'.
Python offers a large number operators that take two arguments and return
True or False depending on their relationship. These might test if two
things are the same ==, are not the same !=, but also if one thing is
smaller < , smaller or equal <=, greater >,greater or equal >=  than the
other one. What is the meaning of the result depends on the type of the
compared objects. Let's find out.

# Ask Python if 2 is exactly equal to 2.

#2==2

#a=3
#a=2
#a==3

#print( a == 2 )

#a = 2
#print( a == 2 )

#%% Ask Python if 2 is not equal to 3.
#print( a != 2 )

#%% Ask Python if 2.0 is greater than 5.0.
#print(45.0 <= 45.0)

#%% Apparently, for numbers, comparision operators match our intuition.


#%% Now, what about comparing strings ? Let's ask Python if "cat" is
bigger than "mouse"

#print( "cat" > "mouse")

#%% What about 'cat' and 'Mouse' ?
#print( "cat" > "Mouse")


#%% Apparently, for strings, comparision operators correspond to
alphabetical order, however, all capital letters precede (are smaller)
than lower case letters.
```

```
#%% Now, what about '5' and'cat' ? Can you guess what's the order of
numbers and letters ?

#print( "0" > "1")
#print( "1" > "2")
#print( "A" > "0")

#print( "1" < "2")
#print( "0" < "1")

#%% Each character does have a number associated with it that is used
when two characters are to be compared.

#ord('0')
#ord('A')

#%% in, not in, startswith:  We can also ask Python if a particular
string is or is not a part of another string or check if a string starts
with a given string. Note that in the last case we use a method to run a
test.
#print( 'AT' in 'cat')


#%% and, or, not: Frequently several conditions need to be tested at the
same time. For example, we might have to check if a number falls within
(0.0,1.0) interval or of an integer is divisible by 3 and greater than
150 or if a DNA sequence starts with 'ATG' and contains one of the stop
codons. To do this we can use boolean (or logical) operators. Lets try
them.

#a = 123.5
#print( a >= 0 and a <= 1 )


#%% logical AND
# Let's try to test if some integer number stored in anumber variable is
positive and odd

           153

# What about divisible by 3 and greater than 150 ?

print             3    0  and            150
```

```
#%% logical OR
# What about testing if a DNA sequence stored in a sequence variable
contains one of the stop codons('TAA','TGA','TAG' )
             'ATGAAAATATGAGCGAGG'

print 'TAA' in           or 'TGA' in           or 'TAG' in


#%% logical NOT

print  not 2 3   5 2


#%% ------ MAKING DECISIONS: CONDITIONAL STATEMENTS
#IF-ELSE
# Logical expressions are most often used within conditional statements,
such as "if-else".  These are statements that tell Python what to do "if"
a given condition is True and what "else" to do  if the condition is
False. Let's try writing an if else statement that prints the larger of
two numbers.  Fist, lets initialize two variables to represent the two
numbers.

    0
    0

#%%  Now we'll write the if else statement.
# Note the editor automatically formats the statement for us.  This is
done to streamline how much we have to type (colons to separate the 'if'
and 'else' conditions from the actions, where as other languages require
numerous parentheses) as well as to make the code easier to read in terms
of an executing hierarchy.

if
    print
else
    print


#%% Do you know a simpler way to print the larger of two numbers ?


#%% ------ MAKING DECISIONS: CONDITIONAL STATEMENTS
# IF-ELIF-ELSE

# Sometimes we need to decide between more than two cases with  'elif'
command (shorthand for 'else if').  You may find that you have multiple
conditional scenarios for which you want Python to do different things
```

```
(for example, think about how many possibilities we had in our Truth
table, or maybe when comparing numbers we are interested in exact
equality as well as greater than or lesser than.).  For these scenarios,
we use the 'elif' statement.  Let's try using the elif statement to tell
Python to check if one number is greater than, exactly equal, or lesser
than another number, and for each instance to print "greater than',
'exactly equal' or 'lesser than'.

    0

if     0
    print 1
elif      0
    print "value invalid"
else
    print  1


#%% ------ WORKING WITH TEXT ------
# Here we will begin learning some more Python and programming
principles, but strictly in the context of sequencing analysis.  This
type of data may not be your thing, but we invite you to come along for
this part of the workshop anyways as we will still learn vital
programming concepts.  However, for examples of analyzing other types of
data (e.g. spreadsheets of data, images) we encourage you to consider
taking Advanced Python Workshop and any other QCB Collaboratory workshops
that may focus on some specific Python  application (Machine Learning
with Python, Digital Image Processing with Python, etc).

#%% ------ QUOTATIONS MARKS ------

# First off, when defining a segment of text, or a string, how many
quotes do we need to use?

# Try defining a string with single quotes.


#%% Try defining a string with double quotes.


#%% What if we mix single and double quotes?


#%% What if our string is possessive (it contains an apostrophe)?


# Try using double quotes, or placing a slash in front of the apostrophe
```

```
#%% ------ STRING OPERATIONS: LOGICAL IN/EXCLUSION, CONCATENATION,
MULTIPLICATION ------

# in:  We can use the "in" operator and the "not in" operator to search
for the precense of one string (or motif) in another.  Try asking Python
if the string "ATA" is contained in the larger string "TATATATA"



#%% not in:  Try asking Python if the string "ATG" is not contained in
the larger string "AAATGAAACGATTTT"


#%% +:  The addition operation can be used to merge two strings together.
This is called concatenation.  Try concatenating the strings "ATG" and
"CAT"



#%% *:  The multiplication symbol can be used to repeat strings.  Try
repeating the string "CAT" 5 times.


#%% ------ BASIC FUNCTIONS ON STRINGS ------

# Python has several basic functions that can be used to easily access
information about your string or to modify it one way or another. To see
a list of these, use the "dir()" function with the name of your string as
the argument. You can also use help(str) to see a description of str (ie
string) type or help(str.<function>) to see description of individual
function.


# To use these functions (also called "methods") on your string, type the
name of your string, followed by a period, followed by the name of the
function, followed by open and close parentheses.


#%% upper:  We can set all of the alphabet values in the string to be
upper case values.


#%% lower:  We can set all of the alphabet values in the string to be
lower case values.
```

```python
#%% count:  We can ask Python for a count of all instances of a
particular element in a string.  With this function, we must place the
element of interest within the parentheses.  Try asking Python for all
instances of the element 'e' in your string.


#%%  len:  We can ask Python for the length of a string using the
function len().  Unlike our previous functions, this function is used by
placing the string (or its variable name inside of the parentheses.  Try
asking Python for the length of your string.


#%% ------- FUNCTIONS ------

# Like the functions that are inherent in Python, we can write our own
functions.  They often correspond to a self-contained block of code that
parforms some specific operation on some data.  Functions tend to be used
multiple times withinaprogram and might be even shared between different
programs.  We will show how to do it tomorrow but, first, we have to
write our first function.  Function definiton typically looks like this:

#   def function_name( parameter1, parameter2,... ):
#       enter
#       here
#       any
#       code
#       needed
#       and comments
#   return result_of_function

# For our first function, we will write a function that takes two
parameters, multiplies them and returns the result.


#%% Now let's run (or 'call' ) our function with different inputs.


#%% Obviously, we can assign the result of the function call to
somevariabe .


#%% Now let's make a more 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.


#%%  ------- Final function of the day -----

# Instructions for writing this function.  This new function should take
a sequence as an input
#parameter (a string composed of A, T, G, C).  It will then search that
sequence and, if the
#sequence contains 'ATG', it should print 'Start codon present at
position='. It should then
#check if the sequence contains one of the stop codons, and, if present,
it should print
#'Stop codon TAA present at position=' and the stop codon(s).
```