Getting Started with Python and Jupyter Notebooks

The purpose of this Jupyter Notebook (https://jupyter.org) is to get you started using Python and Jupyter Notebooks for routine engineering calculations. This introduction assumes this is your first exposure to Python or Jupyter notebooks.

This notebook composes information available here (https://jckantor.github.io/CBE30338/01.01-Getting-Started-with-Python-and-Jupyter-Notebooks.html) and here (https://numpy.org/doc/stable/user/quickstart.html)

The easiest way to use Jupyter notebooks is to use a cloud-based service such as Google Colaboratory (https://colab.research.google.com). You will need continuous internet connectivity to access your work, but the advantages are there is no software to install or maintain.

Installing Jupyter/Python on your Laptop

For regular off-line use you should consider installing a Jupyter Notebook/Python environment directly on your laptop. This will provide you with reliable off-line access to a computational environment. This will also allow you to install additional code libraries to meet particular needs.

Choosing this option will require an initial software installation and routine updates. For this course the recommended package is Anaconda available from Continuum Analytics. Downloading and installing the software is well documented and easy to follow. Allow about 10-30 minutes for the installation depending on your connection speed.

After installing be sure to check for updates before proceeding further. With the Anaconda package this is done by executing the following two commands in a terminal window:

```
> conda update conda
> conda update anaconda
```

Anaconda includes an 'Anaconda Navigator' application that simplifies startup of the notebook environment and manage the update process.

Installing the Course Environment

Instrutions for the Anaconda Navigator are slightly different and are not covered in this notebook. We assume you use a terminal from hereon.

If you decide to use Anaconda, the first thing to do is to create a virtual environment for the course. This makes sure that you do not pollute your main OS python envinronment.

You can do this with:

```
conda create --name feedback-control python=3.10
```

You only need to run the previous command once.

You then just need to activate your environment:

```
conda activate feedback-control
```

create a folder you want to use for this course. You can use your OS GUI or run:

```
mkdir feedback-control
```

This creates the folder `feedback-control` in your current directory. Make sure you are in the correct folder before executing the previous commands.

To run the notebooks you need to install the following packages: `fastcore pandas matplotlib control sympy numpy ffmpeg-python`

you can do this running:

```
python -m pip install fastcore pandas matplotlib control sympy numpy ffmpeg-python notebook
```

You are ready to go!

Run:

```
jupyter notebook
```

to start your notebook session.

1. Start a Jupyter Notebook Session

If you are using a cloud-based service a Jupyter session will be started when you log on.

If you have installed a Jupyter/Python distribution on your laptop then you can open a Jupyter session in one of two different ways:

- Use the Anaconda Navigator App, or
- Open a terminal window on your laptop and execute the following statement at the command line:

```
> jupyter notebook
```

Either way, once you have opened a session you should see a browser window.

At this point the browser displays a list of directories and files. You can navigate amoung the directories in the usual way by clicking on directory names or on the 'breadcrumbs' located just about the listing.

Jupyter notebooks are simply files in a directory with a .ipynb suffix.

2. Simple Calculations with Python

Python is an elegant and modern language for programming and problem solving that has found increasing use by engineers and scientists. In the next few cells we'll demonstrate some basic Python functionality.

In [ ]:

```
a = 12
b = 2

print(a + b)
print(a**b)
print(a/b)
```

```
14
144
6.0
```

Python Libraries

The Python language has only very basic operations. Most math functions are in various math libraries. The numpy library is convenient library. This next cell shows how to import numpy with the prefix np, then use it to call a common mathematical function

In [ ]:

```
import numpy as np

# mathematical constants
print(np.pi)
print(np.e)

# trignometric functions
angle = np.pi/4
print(np.sin(angle))
print(np.cos(angle))
print(np.tan(angle))
```

```
3.141592653589793
2.718281828459045
0.7071067811865475
0.7071067811865476
0.9999999999999999
```

Working with Lists

Lists are a versatile way of organizing your data in Python.

In [ ]:
```
xList = [1, 2, 3, 4]
xList
```

Out[ ]:
```
[1, 2, 3, 4]
```

You can join one list to another or **concatentate** them

In [ ]:
```
# Concatenation
x = [1, 2, 3, 4];
y = [5, 6, 7, 8];

x + y
```

Out[ ]:
```
[1, 2, 3, 4, 5, 6, 7, 8]
```

In [ ]:
```
np.sum(x)
```

Out[ ]:
```
10
```

Element by element operation

In [ ]:
```
print(np.add(x,y))
print(np.multiply(x,y))
print(np.dot(x,y))
```

```
[ 6  8 10 12]
[ 5 12 21 32]
70
```

A for loop is a means for iterating over the elements of a list. The colon marks the start of code that will be executed for each element of a list. Indenting has meaning in Python. In this case, everything in the indented block will be executed on each iteration of the for loop. This example also demonstrates string formatting.

In [ ]:
```
for x in xList:
    print("sin({0}) = {1:8.5f}".format(x,np.sin(x)))
```

```
sin(1) =  0.84147
sin(2) =  0.90930
sin(3) =  0.14112
sin(4) = -0.75680
```

NumPy arrays

Note that while you can do calculations on the lists, NumPy has a special object to represent math vectors or matrices called [array (https://numpy.org/doc/stable/reference/generated/numpy.array.html)](https://numpy.org/doc/stable/reference/generated/numpy.array.html).

This is NumPy's main object and it is a homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of non-negative integers. In NumPy dimensions are called axes.

NumPy arrays are much more powerful.

Creating an array:

In [ ]:
```
a = np.array([2, 3, 4])
```

array transforms sequences of sequences into two-dimensional arrays, sequences of sequences of sequences into three-dimensional arrays, and so on.

In [ ]:
```
b = np.array([(1.5, 2, 3), (4, 5, 6)])
print(b)
```

```
[[1.5 2.  3. ]
 [4.  5.  6. ]]
```

The type of the array can also be explicitly specified at creation time:

In [ ]:
```
c = np.array([[1, 2], [3, 4]], dtype=complex)
print(c)
```

```
[[1.+0.j 2.+0.j]
 [3.+0.j 4.+0.j]]
```

Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with initial placeholder content. These minimize the necessity of growing arrays, an expensive operation.

In [ ]:
```
print(np.zeros((3, 4)))
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

In [ ]:
```
np.ones((2, 3, 4), dtype=np.int16)
```

Out[ ]:
```
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]], dtype=int16)
```

Arithmetic operators on arrays apply elementwise. A new array is created and filled with the result.

In [ ]:
```
a = np.array([20, 30, 40, 50])
b = np.arange(4)
print(a)
print(b)
```

```
[20 30 40 50]
[0 1 2 3]
```

In [ ]:
```
c = a - b
print(c)
```

```
[20 29 38 47]
```

In [ ]:
```
b**2
```

Out[ ]:
```
array([0, 1, 4, 9])
```

In [ ]:
```
10 * np.sin(a)
```

Out[ ]:
```
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
```

In [ ]:
```
a < 35
```

Out[ ]:
```
array([ True,  True, False, False])
```

Important Unlike in many matrix languages, the product operator `*` operates elementwise in NumPy arrays. The matrix product can be performed using the `@` operator (in python >=3.5) or the `dot` function or method:

In [ ]:
```
A = np.array([[1, 1],
              [0, 1]])

B = np.array([[2, 0],
              [3, 4]])
```

In [ ]:
```
A * B     # elementwise product
```

Out[ ]:
```
array([[2, 0],
       [0, 4]])
```

In [ ]:
```
A @ B     # matrix product
```

Out[ ]:
```
array([[5, 4],
       [3, 4]])
```

In [ ]:
```
A.dot(B)  # another matrix product
```

Out[ ]:
```
array([[5, 4],
       [3, 4]])
```

Working with Dictionaries

Dictionaries are useful for storing and retrieving data as key-value pairs.

In [ ]:
```
mw = {'CH4': 16.04, 'H2O': 18.02, 'O2':32.00, 'CO2': 44.01}
mw
```

Out[ ]:
```
{'CH4': 16.04, 'H2O': 18.02, 'O2': 32.0, 'CO2': 44.01}
```

We can retrieve a value from a dictionary:

In [ ]:
```
mw['CH4']
```

Out[ ]:
```
16.04
```

A for loop is a useful means of interating over all key-value pairs of a dictionary.

In [ ]:
```
for values in mw.keys():
    print("Value {:<s} is {}".format(values, mw[values]))
```

```
Value CH4 is 16.04
Value H2O is 18.02
Value O2 is 32.0
Value CO2 is 44.01
```

Dictionaries can be sorted by key or by value

In [ ]:
```
for values in sorted(mw):
    print(" {:<8s}  {}".format(values, mw[values]))
```

```
CH4       16.04
CO2       44.01
H2O       18.02
O2        32.0
```

In [ ]:
```
for values in sorted(mw, key = mw.get):
    print(" {:<8s}  {}".format(values, mw[values]))
```

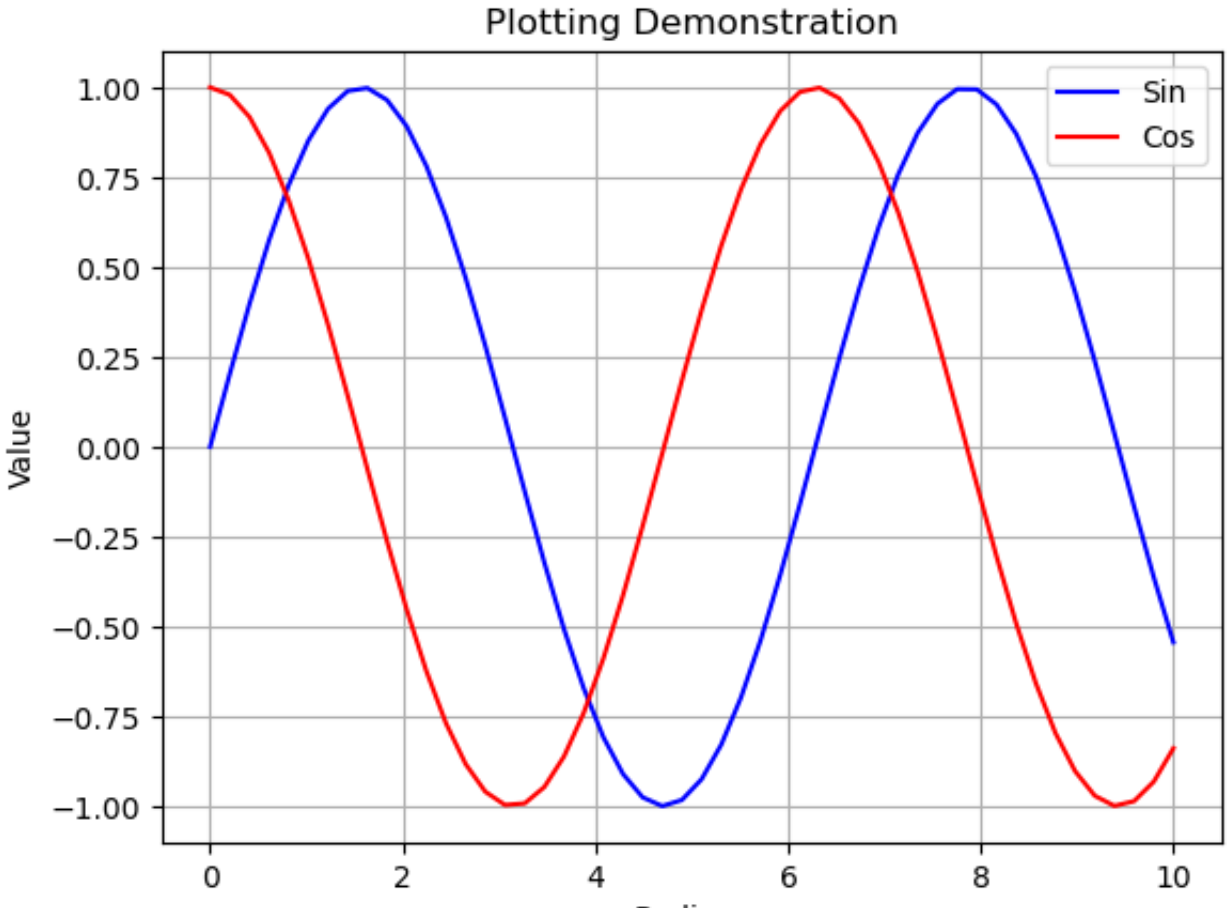
```
CH4       16.04
H2O       18.02
O2        32.0
CO2       44.01
```

Plotting with Matplotlib

Importing the matplotlib.pyplot library gives IPython notebooks plotting functionality very similar to Matlab's. Here are some examples using functions from the

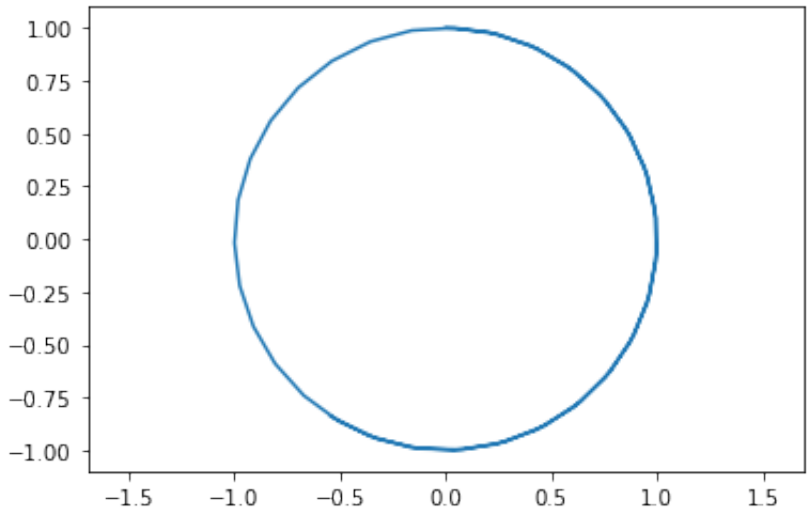
In [ ]:

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,10)
y = np.sin(x)
z = np.cos(x)

plt.plot(x,y,'b',x,z,'r')
plt.xlabel('Radians');
plt.ylabel('Value');
plt.title('Plotting Demonstration')
plt.legend(['Sin','Cos'])
plt.grid()
```



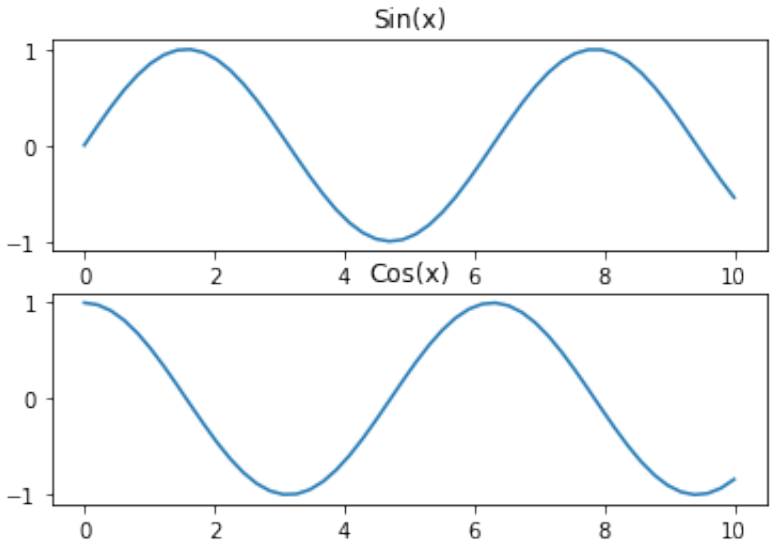
In [ ]:
```
plt.plot(y,z)
plt.axis('equal')
```

Out[ ]:
```
(-1.09972447591003,
 1.0979832896606587,
 -1.0992804688576738,
 1.0999657366122702)
```



In [ ]:
```
plt.subplot(2,1,1)
plt.plot(x,y)
plt.title('Sin(x)')

plt.subplot(2,1,2)
plt.plot(x,z)
plt.title('Cos(x)')
```

Out[ ]: Text(0.5, 1.0, 'Cos(x)')



Where to Learn More

Python offers a full range of programming language features, and there is a seemingly endless range of packages for scientific and engineering computations. Here are some suggestions on places you can go for more information on programming for engineering applications in Python.

Introduction to Python for Science

This excellent introduction to python is aimed at undergraduates in science with no programming experience. It is free and available at the following link.

- Introduction to Python for Science (https://physics.nyu.edu/pine/pymanual/html/pymanMaster.html)

Tutorial Introduction to Python for Science and Engineering

The following text is available on Amazon. Resources for this book are available on github.

- A Primer on Scientific Programming with Python (Fourth Edition) (https://link.springer.com/book/10.1007/978-3-662-49887-3?token=M3aff43&utm_campaign=3_fjp8312_springer_katte_M3aff43&countryChanged=true&gclid=CjwKCAjwiY6MBhBqEiwARFSCPvf-XpMOTr8-ilpV5PSmz8-V62APimY0mv0G2gbU6DiXOk0mDSxSfRoCyNwQAvD_BwE) by Hans Petter Langtangen. Resources for this book are available on github (http://hplgit.github.io/scipro-primer/).

pycse is a package of python functions, examples, and document prepared by John Kitchin at Carnegie Mellon University.

- pycse (https://github.com/jkitchin/pycse/blob/master/pycse.pdf) - Python Computations in Science and Engineering by John Kitchin at Carnegie Mellon. This (https://github.com/jkitchin/pycse) is a link into the the github repository for pycse, click on the Raw button to download the .pdf file.

And there is plenty more! Google it!

Python Basics

This second part of the notebook is to describe some more Python concepts that will be used during the class.

Variables

In [ ]:

```python
#A variable stores a piece of data and gives it a name
answer = 42

#answer contained an integer because we gave it an integer!

is_it_thursday = True
is_it_wednesday = False

#these both are 'booleans' or true/false values

pi_approx = 3.1415

#This will be a floating point number, or a number containing digits after the decimal point

my_name = "Andrea"
#This is a string datatype, the name coming from a string of characters

#Data doesn't have to be a singular unit

#p.s., we can print all of these with a print command. For Example:
print(answer)
print(pi_approx)
```

```
42
3.1415
```

More Complicated Data Types

In [ ]:

```python
#What if we want to store many integers? We need a list!
prices = [10, 20, 30, 40, 50]

#This is a way to define a list in place. We can also make an empty list and add to it.
colors = []

colors.append("Green")
colors.append("Blue")
colors.append("Red")
```

```
print(colors)

#We can also add unlike data to a list
prices.append("Sixty")

#As an exercise, look up lists in python and find out how to add in the middle of a list!

print(prices)
#We can access a specific element of a list too:

print(colors[0])
print(colors[2])

#Notice here how the first element of the list is index 0, not 1!
#Languages like MATLAB are 1 indexed, be careful!

#In addition to lists, there are tuples
#Tuples behave very similarly to lists except that you can't change them
# after you make them

#An empty Tuple isn't very useful:
empty_tuple = ()

#Nor is a tuple with just one value:
one_tuple = ("first",)

#But tuples with many values are useful:
rosa_parks_info = ("Rosa", "Parks", 1913, "February", 4)

#You can access tuples just like lists
print(rosa_parks_info[0] + " " + rosa_parks_info[1])

# You cannot modify existing tuples, but you can make new tuples that extend
# the information.
# I expect Tuples to come up less than lists. So we'll just leave it at that.
```

```
['Green', 'Blue', 'Red']
[10, 20, 30, 40, 50, 'Sixty']
```

```
Green
Red
Rosa Parks
```

Using variables

In [ ]:

```python
float1 = 5.75
float2 = 2.25
#Addition, subtraction, multiplication, division are as you expect

print(float1 + float2)
print(float1 - float2)
print(float1 * float2)
print(float1 / float2)

#Here's an interesting one that showed up in the first homework in 2017. Modulus:
print(5 % 2)
```

```
8.0
3.5
12.9375
2.5555555555555554
1
```

Importing in Python

In [ ]:

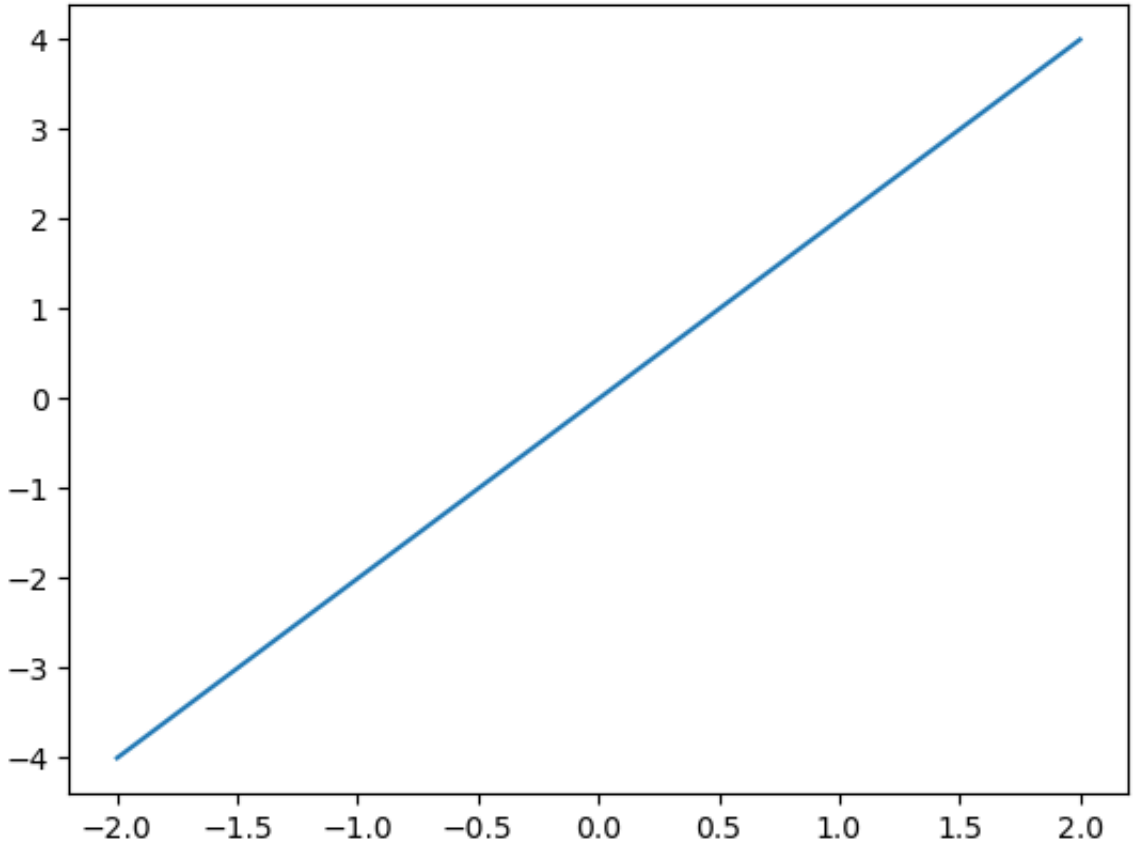
```python
#Just about every standard math function on a calculator has a python equivalent pre made.
#however, they are from the 'math' package in python. Let's add that package!
import math
print(math.log(float1))
print(math.exp(float2))
print(math.pow(2,5))
# There is a quicker way to write exponents if you want:
print(2.0**5.0)

#Like in MATLAB, you can expand the math to entire lists
list3 = [1, 2, 3, 4, 5]
print(2 * list3)
```

```
1.749199854809259
9.487735836358526
32.0
32.0
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

In [ ]:

```
# We can plot easily in Python like in matlab, just import the relevant package!
import matplotlib.pyplot as plt

x_vals = [-2, -1, 0, 1, 2]
y_vals = [-4, -2, 0, 2, 4]
plt.plot(x_vals, y_vals)
```

Out[ ]: [<matplotlib.lines.Line2D>]



Loops

In [ ]:

```python
#Repeat code until a conditional statement ends the loop

#Let's try printing a list
fib = [1, 1, 2, 3, 5, 8]

#While loops are the basic type
i = 0
while(i < len(fib)):
    print(fib[i])
    i = i + 1

#In matlab, to do the same thing you would have the conditional as: counter < (length(fib) + 1)
#This is because matlab starts indexing at 1, and python starts at 0.

#The above type of loop is so common that the 'for' loop is the way to write it faster.

print("Let's try that again")
#This is most similar to for loops in matlab
for i in range(0, len(fib)) :
    print(fib[i])

print("One more time:")
#Or you can do so even neater
for e in fib:
    print(e)
```

```
1
1
2
3
5
8
Let's try that again
1
1
2
3
5
8
```

```
One more time:
1
1
2
3
5
8
```

Functions (https://www.w3schools.com/python/python_functions.asp)

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

In [ ]:
```
def my_function():
    print("Hello from a function")
```

To call a function, use the function name followed by parenthesis:

In [ ]:
```
my_function()
```

```
Hello from a function
```

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

In [ ]:
```
def my_function(fname):
    print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

In [ ]:
```
def my_function(food):
    for x in food:
        print(x)
```

In [ ]:
```
fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

```
apple
banana
cherry
```

To let a function return a value, use the return statement:

In [ ]:
```python
def my_function(x):
    return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

Classes (https://www.geeksforgeeks.org/python-classes-and-objects/)

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.

To understand the need for creating a class let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed, age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's why we need classes.

Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

It's not hard to define Python class. To do so, you'll need the `class` keyword:

```
class ClassName:     # Statement-1     .     .     .     # Statement-N
```

For example

In [ ]:
```python
class Example:
    variable = 123
```

If you run the above code in a Python environment, you'll find you can call `Example.variable` to return an integer value.

In [ ]:
```
Example.variable
```

Out[ ]:
```
123
```

This is an example of a class for data-only objects, but it's equally easy to define a class that returns a function object by adding the `def` keyword to your code:

In [ ]:
```
class Example:
    def b(self):
        return "this is an example class"
```

In [ ]:
```
Example.b # we are accessing the function...this is probably not what we want to do..
```

Out[ ]:
```
<function __main__.Example.b(self)>
```

We need a few more concepts:

Some more class concepts

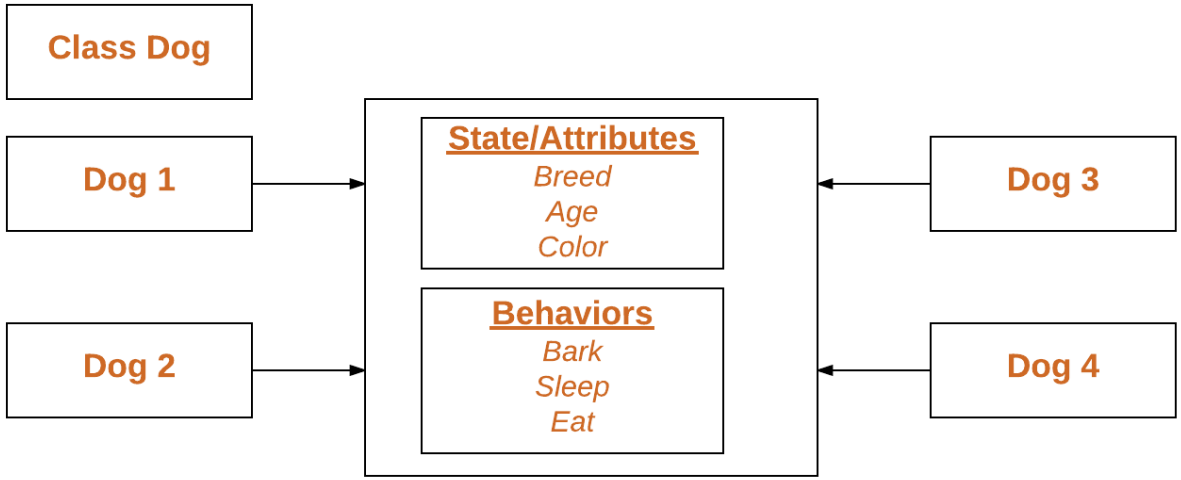
An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values. It's not an idea anymore, it's an actual dog, like a dog of breed pug who's seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required. An object consists of :

- State: It is represented by the attributes of an object. It also reflects the properties of an object.
- Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.
- Identity: It gives a unique name to an object and enables one object to interact with other objects.

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Example:



In [ ]:

```
class Dog:

    # A simple class
    # attribute
    attr1 = "mammal"
    attr2 = "dog"

    # A sample method
    def fun(self):
        print("I'm a", self.attr1)
        print("I'm a", self.attr2)

# Object instantiation
Rodger = Dog()

# Accessing class attributes
# and method through objects
print(Rodger.attr1)
Rodger.fun()
```

```
mammal
I'm a mammal
I'm a dog
```

Self

Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it.

If we have a method that takes no arguments, then we still have to have one argument.

When we call a method of this object as `myobject.method(arg1, arg2)`, this is automatically converted by Python into `MyClass.method(myobject, arg1, arg2)`.

Note that this means that inside the function `method` (in our example) we now have access to the instance of the class! so we can access its variables, etc.

__init__ method

The **init** method is similar to constructors in C++, it constructs the object and can be used to initialise the object's state.

Like methods, a constructor also contains a collection of statements (i.e. instructions) that are executed when the object is created.

The `__init__` method runs as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

In [ ]:

```python
# A Sample class with init method
class Person:

    # init method or constructor
    def __init__(self, name):
        self.name = name

    # Sample Method
    def say_hi(self):
        print('Hello, my name is', self.name)

p = Person('Nikhil') # as soon as we do this, the __init__ method is called.
p.say_hi()
```

```
Hello, my name is Nikhil
```

Class and Instance Variables

- Instance variables are used to store data that is unique to each instance of the class. Instance variables are variables whose value is assigned inside the `__init__` method or inside a class method (a method with the argument `self`)
- Class variables are for attributes and methods shared by all instances of the class. Class variables are variables whose value is assigned directly in the class.

In [ ]:

```python
# Class for Dog
class Dog:

    # Class Variable
    animal = 'dog'

    # The init method or constructor
    def __init__(self, breed, color):

        # Instance Variable
        self.breed = breed
        self.color = color

# Objects of Dog class
Rodger = Dog("Pug", "brown")
Buzo = Dog("Bulldog", "black")

print('Rodger details:')
print('Rodger is a', Rodger.animal)
print('Breed: ', Rodger.breed)
print('Color: ', Rodger.color)

print('\nBuzo details:')
print('Buzo is a', Buzo.animal)
print('Breed: ', Buzo.breed)
print('Color: ', Buzo.color)

# Class variables can be accessed using class
# name also
print("\nAccessing class variable using class name")
print(Dog.animal)
```

```
Rodger details:
Rodger is a dog
Breed:  Pug
Color:  brown

Buzo details:
Buzo is a dog
```

```
Breed:  Bulldog
Color:  black

Accessing class variable using class name
dog
```

Additional Resources

- Code Academy (https://www.codecademy.com/catalog)
- Official Python Reference (https://docs.python.org/3/reference/index.html)
- Real Python (https://realpython.com)

Google Colab

- An Introduction to Google Colab (https://mcgrawect.princeton.edu/guides/Google-Colab-Introduction.pdf), McGraw Center for Teaching and Learning
- Getting Started with Google Colab (https://towardsdatascience.com/getting-started-with-google-colab-f2fff97f594c)
- Colab Walkthrough (https://cs230.stanford.edu/section/2/colab.pdf), Stanford University
- Google Colab Tutorial for Data Scientists (https://www.datacamp.com/tutorial/tutorial-google-colab-for-data-scientists#!), Datacamp.com