Python Basic Syntax

Prepared by Andy Shahidehpour (ashahide@hawk.iit.edu (mailto:ashahide@hawk.iit.edu)) and Essa Almutar (ealmutar@hawk.iit.edu (mailto:ealmutar@hawk.iit.edu))

Introduction

Python is a popular programming language for science and engineering applications. It's free and open source, which makes it ideal for students to use throughout their school career. The goal of this workshop is to give you a basic understanding of Python and some sample code to use later.

Part 1: Libraries and How to Use Them

Libraries are collections of code built to add funcitonality to the base language. For example, chemists can build libraries that contain code specifically for chemistry calculations, or researchers can build libraries to implement equations that they have created. Most of the libraries you will need are included with Anaconda so you just need to import them to each program.

To import a a library, you write "import {library} as {nickname}" and use the function in the libraries as something like nickname.{commmand}. Look at the examples below:

NumPy (Num-Pie) is a library made for using matrices and arrays.

Once numpy is imported as np, we can use the functions in the library like np._____

In [1]:
```
import numpy as np
```

In [2]:
```
A_array = np.array([1,2,3]) # This creates a numpy array
print(A_array)
print(type(A_array))
```

```
[1 2 3]
<class 'numpy.ndarray'>
```

In [3]:
```
A_list = [1,2,3]  # This is a list of numbers instead of an array. Lists work
print(A_list)
type(A_list)
```

```
[1, 2, 3]
```

Out[3]:
```
list
```

SciPy (Sigh-Pie) is a scientific library that includes function for everything from curve fitting and optimization to differential equation solvers

In [4]:
```
import scipy as sp

print(type(sp))
```

```
<class 'module'>
```

Pandas is used for data manipulation.

In [5]:
```
import pandas as pd

print(type(pd))
```

```
<class 'module'>
```

Part 2: Mathematical Operations

In [6]:
```
1 + 2 #addition
```

Out[6]:
```
3
```

In [7]:
```
1 - 2 # subtraction
```

Out[7]:
```
-1
```

In [8]:
```
1 / 2 # division
```

Out[8]:
```
0.5
```

In [9]:
```
1 * 2 # multiplication
```

Out[9]:
```
2
```

In [10]:
```
P = 1
print(P)
```

```
1
```

In [11]:
```
P  = 1
P += 1 # add one to the value of P. This is the same as P = P + 1
print(P)
```

```
2
```

In [12]:
```
P  = 1
P -= 1 # subtract one from the value of P. This is the same as P = P - 1
print(P)
```
```
0
```

In [13]:
```
P  = 5
P /= 2 # Divide the value of P by two. This is the same as P = P/2
print(P)
```
```
2.5
```

In [14]:
```
P  = 5
P *= 2 # Multiply the value of P by two. This is the same as P = P*2
print(P)
```
```
10
```

Operations can be done on whole arrays of numbers

In [15]:
```
A = np.array([1,2,3,4,5])
print(A)
```
```
[1 2 3 4 5]
```

In [16]:
```
A / 5 # This divides each number in the array by 5
```
Out[16]:
```
array([0.2, 0.4, 0.6, 0.8, 1. ])
```

In [17]:
```
A * 5 # This multiplies each number in the array by 5
```
Out[17]:
```
array([ 5, 10, 15, 20, 25])
```

In [18]:
```
A -= 1 # This subtracts one number from each in A, and then sets A equal to t
print(A)
```
```
[0 1 2 3 4]
```

In [19]:
```
import numpy as np

A = np.array([1,2,3])
B = np.array([2,3,4])
```

In [20]:
```
A + B
```
Out[20]:
```
array([3, 5, 7])
```

In [21]:
```
A - B
```
Out[21]:
```
array([-1, -1, -1])
```

In [22]:
```
A * B # this multiples the values element by element
```
Out[22]:
```
array([ 2,  6, 12])
```

In [23]:
```
np.dot(A,B)
```
Out[23]:
```
20
```

In [24]:
```
A / B
```
Out[24]:
```
array([0.5       , 0.66666667, 0.75      ])
```

Printing

In [25]:
```
A =1
print("A = {} with .format".format(A)) # this will print whatever the value c

print(f"A = {A} with an f-string") # this will print whatever the value of A
```
```
A = 1 with .format
A = 1 with an f-string
```

Indexing/Slicing

NumPy arrays are functionally very similar to matrices. You can also use matrices, but arrays are more general.

In [26]:
```
A = np.array([1,2,3,4,5])
print(A[0], A[1], A[2], A[3], A[4])
```
```
1 2 3 4 5
```

In [27]:
```
print(A[-5], A[-4], A[-3], A[-2], A[-1])
```
```
1 2 3 4 5
```

In [28]:
```
print(A[:])
```
```
[1 2 3 4 5]
```

In [29]:
```
print(A[1:])
```
```
[2 3 4 5]
```

In [30]:
```
print(A[:-1])
```
```
[1 2 3 4]
```

In [31]:
```
print(A[-1])
```

```
5
```

In [32]:
```
A = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print(A)
print("Shape = {}".format(A.shape))
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
Shape = (2, 5)
```

In [33]:
```
print(A[0,0])
```

```
1
```

In [34]:
```
print(A[0,1])
```

```
2
```

In [35]:
```
print(A[::])
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
```

In [36]:
```
print(A[:,0])
```

```
[1 6]
```

In [37]:
```
print(A[0,:])
```

```
[1 2 3 4 5]
```

Matrix operations

In [38]:
```
import numpy as np

A = np.matrix('1, 2, 3; 4, 5, 6; 7, 8, 9')
B = np.matrix('2, 3, 4;1, 2, 3;2, 3, 4')
print(A)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

In [39]:

```
A = np.matrix('1, 2, 3; 4, 5, 6; 7, 8, 9')
B = np.matrix('2, 3, 4;1, 2, 3;2, 3, 4')

Add = A + B
print(f'Add = {Add}')
```

```
Add = [[ 3  5  7]
 [ 5  7  9]
 [ 9 11 13]]
```

In [40]:

```
A = np.matrix('1, 2, 3; 4, 5, 6; 7, 8, 9')
B = np.matrix('2, 3, 4;1, 2, 3;2, 3, 4')

Sub = A - B
print(f'Sub = {Sub}')
```

```
Sub = [[-1 -1 -1]
 [ 3  3  3]
 [ 5  5  5]]
```

In [41]:

```
A = np.matrix('1, 2, 3; 4, 5, 6; 7, 8, 9')
B = np.matrix('2, 3, 4;1, 2, 3;2, 3, 4')

Mult = A * B
print(f'Mult = {Mult}')
```

```
Mult = [[10 16 22]
 [25 40 55]
 [40 64 88]]
```

In [42]:

```
A = np.matrix('1, 2, 3; 4, 5, 6; 7, 8, 9')
B = np.matrix('2, 3, 4;1, 2, 3;2, 3, 4')

inner_product = np.inner(A, B)
print(f"Inner product = {inner_product}")
```

```
Inner product = [[20 14 20]
 [47 32 47]
 [74 50 74]]
```

Linear systems

In [43]:
```
from scipy.optimize import fsolve

def fun(x):
   f1 = x[0] + x[2] - 6
   f2 = -3 * x[1] + x[2] - 7
   f3 = 2 * x[0] + x[1] + 3 * x[2] - 15
   return [f1, f2, f3]

x = fsolve(fun, [1, 1, 1])
print(f'x1 = {round(x[0], 3)}, x2 = {round(x[1], 3)}, x3 = {round(x[2], 3)}')
```

```
x1 = 2.0, x2 = -1.0, x3 = 4.0
```

$$AX = b$$
$$X = A^{-1}b$$

In [44]:
```
A = np.matrix('1 , 0, 1; 0, -3, 1; 2, 1, 3')
b = np.matrix('6 ; 7 ; 15')
A_1 = np.linalg.inv(A)
X = A_1 * b
print(f'x1 = {round(x[0], 3)}, x2 = {round(x[1], 3)}, x3 = {round(x[2], 3)}')
```

```
x1 = 2.0, x2 = -1.0, x3 = 4.0
```

Miscellaneous

In [45]:
```
import scipy.linalg as sci

print(f"\n2**2 = {2**2}")
print(f"\nsqrt(4) = {np.sqrt(4)}")
print(f"\nnp.exp(1) = {np.exp(1)}")
print(f"\nnp.log10(10) = {np.log10(10)}")
print(f"\nsci.expm(A) = {sci.expm(A)}")
print(f"\nnp.exp(A) = {np.exp(A)}")
print(f"\nnp.sin(np.pi) = {np.sin(np.pi)}")
print(f"\nnp.log(2.71) = {np.log(2.71)}")
```

```
2**2 = 4

sqrt(4) = 2.0

np.exp(1) = 2.718281828459045

np.log10(10) = 1.0

sci.expm(A) = [[10.23451671  1.78608529 12.61560863]
 [ 3.57217057  0.85204376  5.47126749]
 [25.23121726  5.47126749 37.25181925]]

np.exp(A) = [[ 2.71828183  1.          2.71828183]
 [ 1.          0.04978707  2.71828183]
 [ 7.3890561   2.71828183 20.08553692]]

np.sin(np.pi) = 1.2246467991473532e-16

np.log(2.71) = 0.9969486348916096
```

Part 3: Loops

There are two main ways to do loops. First is range(k) which loops through values from zero up to but not including k

In [46]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

In [47]:
```
for i in range(1,5):
    print(i)
```

```
1
2
3
4
```

In [48]:
```
A = np.array([1,2,3,4,5])
for i in range(len(A)):
    print(A[i])
```

```
1
2
3
4
5
```

The second main loop method is enumerate. This generates a value and an index for that value.

In [49]:
```
List = ['Apple', 'Orange', 'Banana', 'Grape']

for index, value in enumerate(List):
    print(f"Index = {index}")
    print(f"Value = {value}")
```

```
Index = 0
Value = Apple
Index = 1
Value = Orange
Index = 2
Value = Banana
Index = 3
Value = Grape
```

Part 4: Functions

Functions in Python use indentations. After you define the function use "def", anything indented once will be within the function. The function ends when you remove the indent or when you use the command "return"

In [50]:
```
def Addition(a,b):
    return a + b

c = Addition(1,1)

print(c)
```

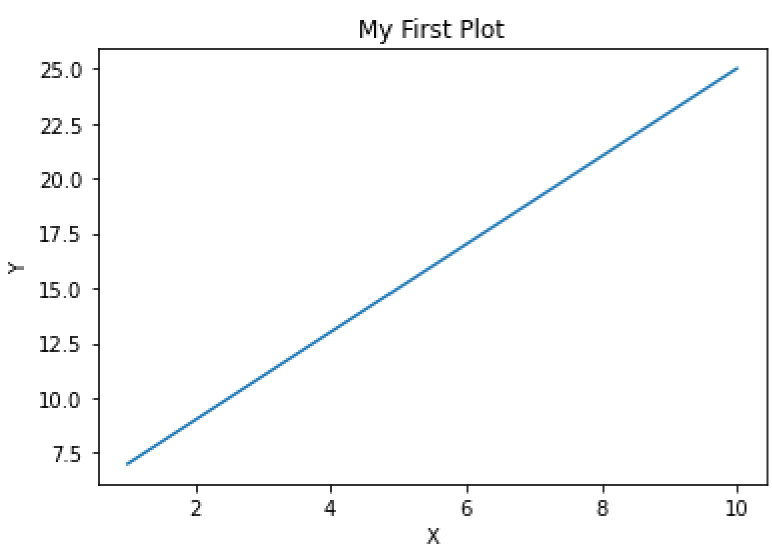
```
2
```

In [51]:
```
def IsThisNumberSeven(a):
    if a == 7: # "==" compares a to 7 instead of assigning a as 7

        k = 'Yes'
    else:
        k = 'No'

    return k

k = IsThisNumberSeven(5)
print(k)

k = IsThisNumberSeven(7)
print(k)
```

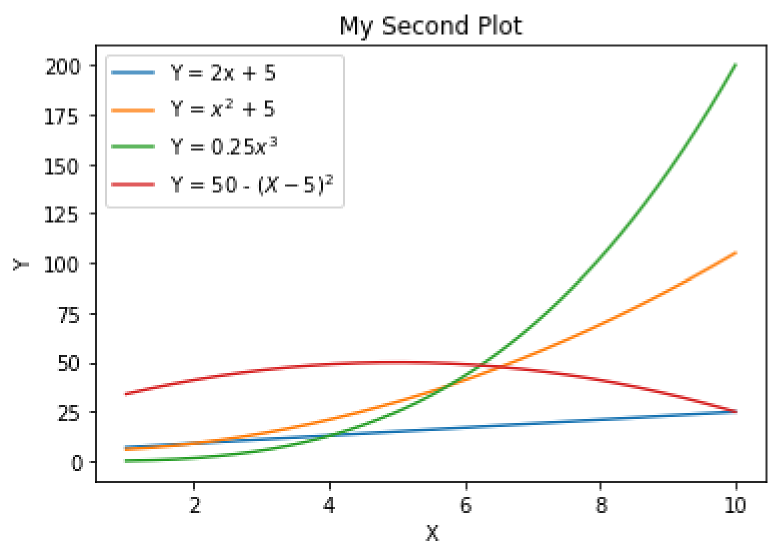
```
No
Yes
```

Part 5: Plotting

In [52]:
```
import matplotlib.pyplot as plt
import numpy as np

X = np.linspace(1, 10, 100)

Y = 2*X + 5

plt.figure()

plt.plot(X,Y)
plt.title('My First Plot')
plt.xlabel('X')
plt.ylabel('Y')

plt.show()
```



In [53]:
```
import matplotlib.pyplot as plt
import numpy as np

X = np.linspace(1, 10, 500)

Y1 = 2*X + 5
Y2 = X**2 + 5
Y3 = (1/5)*X**3
Y4 = -(X - 5)**2 + 50

plt.figure()

plt.plot(X,Y1, label = r'Y = 2x + 5') # Similar to f-strings, r-strings let y
plt.plot(X,Y2, label = r'Y = $x^{2}$ + 5')
plt.plot(X,Y3, label = r'Y = 0.25$x^{3}$')
plt.plot(X,Y4, label = r'Y = 50 - $(X - 5)^{2}$')

plt.title('My Second Plot')
plt.xlabel('X')
plt.ylabel('Y')

plt.legend()

plt.show()
```

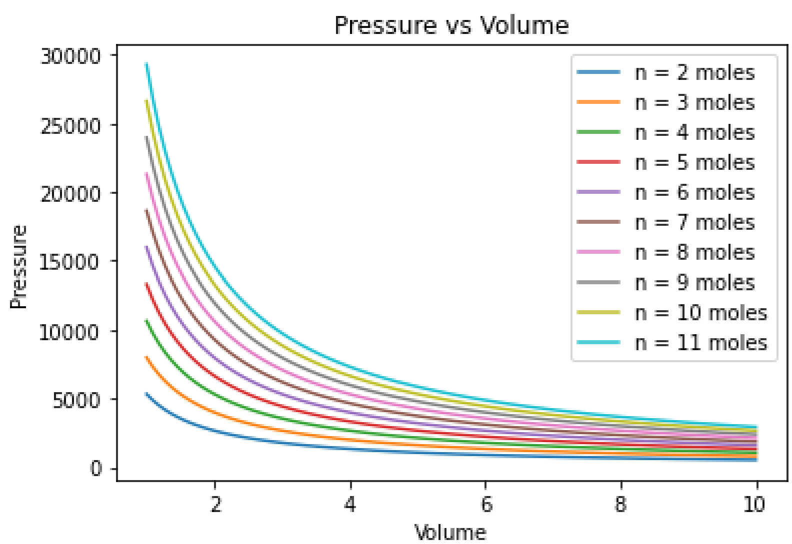


PV = nRT is the ideal gas law. It can be written as P = nRT / V

In [54]:
```
def Pressure(n,R,V,T):
    return n*R*T / V
```

In [55]:
```
for i in range(10):
    n = 2 + i
    R = 8.32
    V = np.linspace(1,10,100)
    T = 320
    P = Pressure(n, R, V, T)

    plt.plot(V, P, label = f'n = {i+2} moles')

plt.xlabel('Volume')
plt.ylabel('Pressure')

plt.legend()

plt.title('Pressure vs Volume')
```

Out[55]:
```
Text(0.5, 1.0, 'Pressure vs Volume')
```



Solving ODEs

Consider a batch reactor with reaction $A \rightarrow B$

$$\frac{dC_A}{dt} = -kC_A \ ; C_A(0) = 10$$

$$\frac{dC_B}{dt} = kC_A \ ; C_B(0) = 0$$

In [56]:
```
from scipy.integrate import odeint # This is basically the same as ode45 in M
import numpy as np
```

In [57]:
```
def equations(x,t):

    k = 1

    Ca = x[0]
    Cb = x[1]

    dCAdt = -k*Ca

    dCBdt = k*Ca

    return dCAdt, dCBdt
```

In [58]:
```
t = np.linspace(0, 10, 100)

initial = [10, 0]

Solutions = odeint(equations,initial,t)
```

In [59]:
```
Solutions
```

Out[59]:
```
array([[1.00000000e+01, 0.00000000e+00],
       [9.03923901e+00, 9.60760990e-01],
       [8.17078420e+00, 1.82921580e+00],
       [7.38576715e+00, 2.61423285e+00],
       [6.67617145e+00, 3.32382855e+00],
       [6.03475094e+00, 3.96524906e+00],
       [5.45495562e+00, 4.54504438e+00],
       [4.93086477e+00, 5.06913523e+00],
       [4.45712652e+00, 5.54287348e+00],
       [4.02890318e+00, 5.97109682e+00],
       [3.64182187e+00, 6.35817813e+00],
       [3.29192982e+00, 6.70807018e+00],
       [2.97565403e+00, 7.02434597e+00],
       [2.68976480e+00, 7.31023520e+00],
       [2.43134268e+00, 7.56865732e+00],
       [2.19774876e+00, 7.80225124e+00],
       [1.98659763e+00, 8.01340237e+00],
       [1.79573307e+00, 8.20426693e+00],
       [1.62320606e+00, 8.37679394e+00],
       [1.46725476e+00, 8.53274524e+00],
       [1.32628666e+00, 8.67371334e+00],
       [1.19886221e+00, 8.80113779e+00],
       [1.08368021e+00, 8.91631979e+00],
       [9.79564445e-01, 9.02043555e+00],
       [8.85451716e-01, 9.11454828e+00],
       [8.00380972e-01, 9.19961903e+00],
       [7.23483492e-01, 9.27651651e+00],
       [6.53974022e-01, 9.34602598e+00],
       [5.91142754e-01, 9.40885725e+00],
       [5.34348066e-01, 9.46565193e+00],
       [4.83009990e-01, 9.51699001e+00],
       [4.36604277e-01, 9.56339572e+00],
       [3.94657043e-01, 9.60534296e+00],
       [3.56739935e-01, 9.64326006e+00],
       [3.22465755e-01, 9.67753424e+00],
       [2.91484504e-01, 9.70851550e+00],
       [2.63479812e-01, 9.73652019e+00],
       [2.38165700e-01, 9.76183430e+00],
       [2.15283670e-01, 9.78471633e+00],
       [1.94600056e-01, 9.80539994e+00],
       [1.75903643e-01, 9.82409636e+00],
       [1.59003509e-01, 9.84099649e+00],
       [1.43727074e-01, 9.85627293e+00],
       [1.29918338e-01, 9.87008166e+00],
       [1.17436292e-01, 9.88256371e+00],
       [1.06153472e-01, 9.89384653e+00],
       [9.59546615e-02, 9.90404534e+00],
       [8.67357122e-02, 9.91326429e+00],
       [7.84024840e-02, 9.92159752e+00],
       [7.08698798e-02, 9.92913012e+00],
       [6.40609787e-02, 9.93593902e+00],
       [5.79062504e-02, 9.94209375e+00],
       [5.23428439e-02, 9.94765716e+00],
       [4.73139483e-02, 9.95268605e+00],
       [4.27682091e-02, 9.95723179e+00],
```

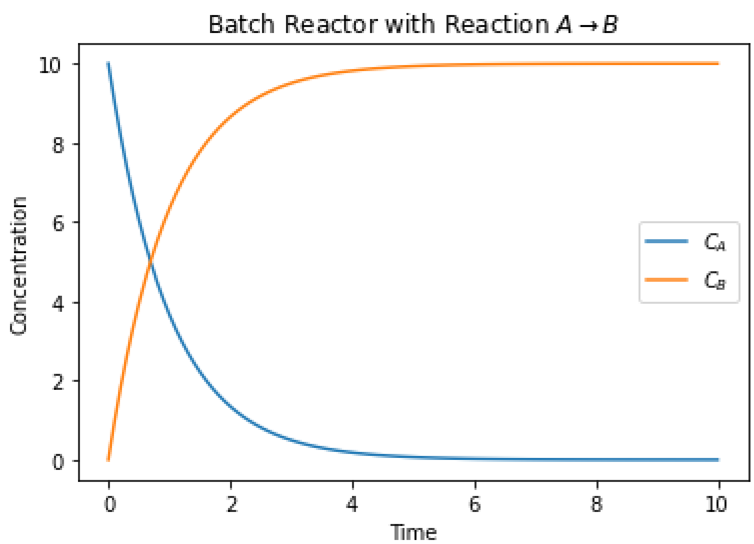
```
       [3.86592080e-02, 9.96134079e+00],
       [3.49449822e-02, 9.96505502e+00],
       [3.15876051e-02, 9.96841239e+00],
       [2.85527913e-02, 9.97144721e+00],
       [2.58095512e-02, 9.97419045e+00],
       [2.33298703e-02, 9.97667013e+00],
       [2.10884280e-02, 9.97891157e+00],
       [1.90623340e-02, 9.98093767e+00],
       [1.72308996e-02, 9.98276910e+00],
       [1.55754221e-02, 9.98442458e+00],
       [1.40789968e-02, 9.98592100e+00],
       [1.27263417e-02, 9.98727366e+00],
       [1.15036446e-02, 9.98849636e+00],
       [1.03984194e-02, 9.98960158e+00],
       [9.39938000e-03, 9.99060062e+00],
       [8.49632429e-03, 9.99150368e+00],
       [7.68003080e-03, 9.99231997e+00],
       [6.94216341e-03, 9.99305784e+00],
       [6.27518753e-03, 9.99372481e+00],
       [5.67229203e-03, 9.99432771e+00],
       [5.12732047e-03, 9.99487268e+00],
       [4.63470755e-03, 9.99536529e+00],
       [4.18942300e-03, 9.99581058e+00],
       [3.78691960e-03, 9.99621308e+00],
       [3.42308720e-03, 9.99657691e+00],
       [3.09421037e-03, 9.99690579e+00],
       [2.79693077e-03, 9.99720307e+00],
       [2.52821258e-03, 9.99747179e+00],
       [2.28531181e-03, 9.99771469e+00],
       [2.06574799e-03, 9.99793425e+00],
       [1.86727902e-03, 9.99813272e+00],
       [1.68787831e-03, 9.99831212e+00],
       [1.52571390e-03, 9.99847429e+00],
       [1.37912927e-03, 9.99862087e+00],
       [1.24662785e-03, 9.99875337e+00],
       [1.12685659e-03, 9.99887314e+00],
       [1.01859219e-03, 9.99898141e+00],
       [9.20729354e-04, 9.99907927e+00],
       [8.32269177e-04, 9.99916773e+00],
       [7.52307678e-04, 9.99924769e+00],
       [6.80028902e-04, 9.99931997e+00],
       [6.14694215e-04, 9.99938531e+00],
       [5.55636681e-04, 9.99944436e+00],
       [5.02253242e-04, 9.99949775e+00],
       [4.53998519e-04, 9.99954600e+00]])
```

In [60]:

```
Ca = Solutions[:,0] # Each column holds the solution for a specific variable.
Cb = Solutions[:,1]
```

In [61]:
```
import matplotlib.pyplot as plt

plt.figure()

plt.plot(t, Ca, label = '$C_{A}$')
plt.plot(t, Cb, label = '$C_{B}$')

plt.xlabel('Time')
plt.ylabel('Concentration')
plt.title(r'Batch Reactor with Reaction $A \rightarrow B$')

plt.legend()
```

Out[61]: <matplotlib.legend.Legend at 0x235fdb511f0>

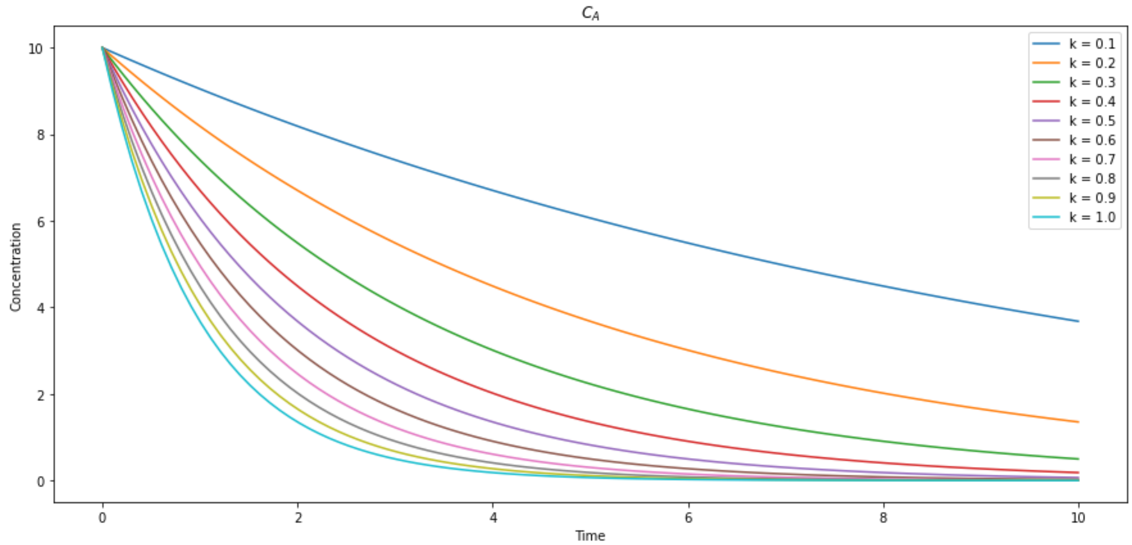


We can also make the rate constant an input and plot different values

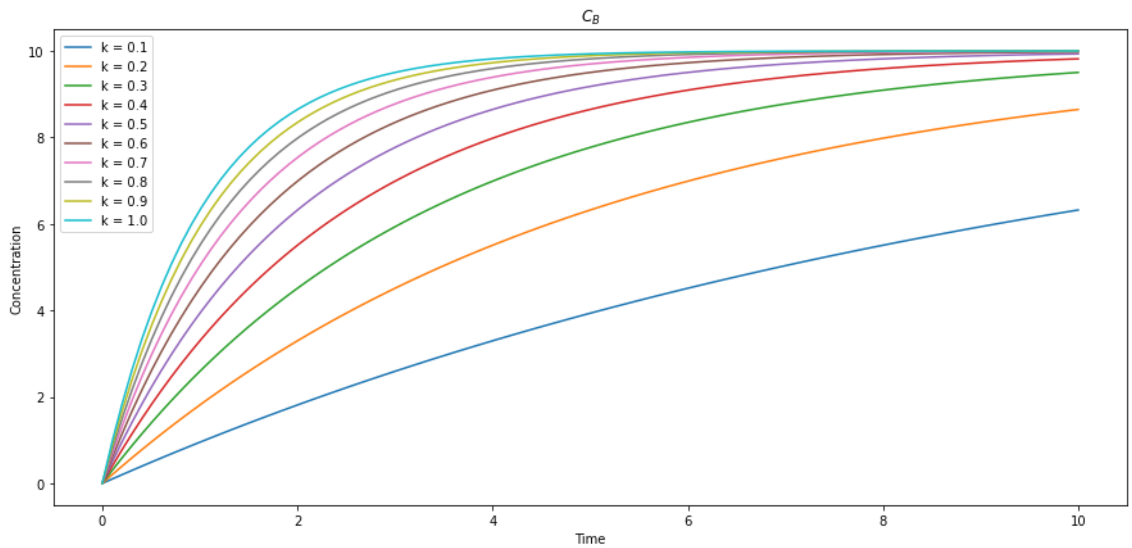
In [62]:
```
def equations(x,t, k):

    Ca = x[0]
    Cb = x[1]

    dCAdt = -k*Ca

    dCBdt = k*Ca

    return dCAdt, dCBdt
```

In [63]:
```
t = np.linspace(0, 10, 100)

initial = [10, 0]

k = np.arange(1, 11)/10 # integers from 1 to 11 NOT including 11 (so 1 to 10)

# make matrices of zeros to hold the solutions. Each column will correspond t
Ca_Solutions = np.zeros((len(t), 10))
Cb_Solutions = np.zeros((len(t), 10))

for i in range(len(k)):
    Solutions = odeint(equations,initial,t, args = (k[i],))

    Ca_Solutions[:,i] = Solutions[:,0]
    Cb_Solutions[:,i] = Solutions[:,1]
```

In [64]:
```
import matplotlib.pyplot as plt

plt.figure(figsize = (15,15))

plt.subplot(2,1,1)

for i in range(len(k)):
    plt.plot(t, Ca_Solutions[:,i], label = f'k = {k[i]}')

plt.xlabel('Time')
plt.ylabel('Concentration')
plt.title(r'$C_{A}$')

plt.legend()

plt.subplot(2,1,2)

for i in range(len(k)):
    plt.plot(t, Cb_Solutions[:,i], label = f'k = {k[i]}')

plt.xlabel('Time')
plt.ylabel('Concentration')
plt.title(r'$C_{B}$')

plt.legend()

plt.show()
```



C_A
Concentration
Time
k = 0.1
k = 0.2
k = 0.3
k = 0.4
k = 0.5
k = 0.6
k = 0.7
k = 0.8
k = 0.9
k = 1.0



C_B
Concentration
Time
k = 0.1
k = 0.2
k = 0.3
k = 0.4
k = 0.5
k = 0.6
k = 0.7
k = 0.8
k = 0.9
k = 1.0