Table of Contents

Installation of Anaconda

Anaconda is a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment.

Please refer to this webpage https://docs.anaconda.com/anaconda/install/index.html for the step-by-step instructions.

Running Python programs

There are two suggested ways to write and run Python programs: one is through Jupyter notebook, and the other one is through Spyder.

Open Anaconda navigator

Read this webpage for how to open the navigator for your operating system (Windows, MacOS, or Linux): https://docs.anaconda.com/anaconda/user-guide/getting-started/#your-first-python-program-hello-anaconda

On the Navigator's Home tab, you can install Jupyter notebook or Spyder.

Jupyter notebook

Read this https://docs.anaconda.com/anaconda/user-guide/getting-started/#run-python-in-a-jupyter-notebook for how to run Python in Jupyter notebook.

Spyder

Spyder is an open-source cross-platform integrated development environment for scientific programming in the Python language. Read this https://docs.anaconda.com/anaconda/user-guide/getting-started/#run-python-in-spyder-ide-integrated-development-environment for how to run Python in Spyder IDE.

Some Python operations

Read this cheat sheet https://www.pythoncheatsheet.org/ for more.

Python basics

https://www.pythoncheatsheet.org/#Python-Basics

$$(1+2)\times 5^2 - 10/3$$

In [1]:
```
(1 + 2) * 5 ** 2 - 10/3
```

Out[1]:
```
71.66666666666667
```

Numpy

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

$$e^{0.1} - 1 \text{ or } \ln(e^{0.2})$$

In [2]:
```
import numpy as np

print(np.exp(0.1) - 1) # You need to call the print function to show the result

np.log(np.exp(0.2))
```

```
0.10517091807564771
```

Out[2]:
```
0.2
```

Create an array $[0.1, 0.2, \ldots, 1.0]$ and access its elements.

In [3]:
```
a = np.arange(0.1, 1.1, 0.1) # create an array, 0.1 is the spacing between value
print(a)
```

```
print(np.linspace(0.1, 1.0, 10)) # another way

print(a[0]) # the first element

print(a[-1]) # the last element
```

```
[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
0.1
1.0
```

We can print strings along with numbers.

In [4]:

```
print('The last element of ', a, ' is ', a[-1])
```

```
The last element of  [0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]  is  1.0
```

We can extract elements from the array.

In [5]:

```
print(a[0:5]) # The result includes the start index, but excludes the end index.

print(a[0:-1]) # the array excluding the last element
print(a[:-1]) # another way
```

```
[0.1 0.2 0.3 0.4 0.5]
[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
[0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

In [6]:

```
a[1:] # the array excluding the first element
```

Out[6]:

```
array([0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

It is handy to work with arrays.

In [7]:

```
a * 2
```

Out[7]:

```
array([0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, 1.8, 2. ])
```

If you work with a list of ten numbers, the above multiplication is equivalent to replication.

In [8]:

```
b = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
print('The type of object ', b, ' is ', type(b))

b * 2 # replicate the list three times, but it's not what we want.
```

```
The type of object  [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]  is  <cla
ss 'list'>
```

Out[8]:

```
[0.1,
 0.2,
 0.3,
 0.4,
 0.5,
 0.6,
 0.7,
 0.8,
 0.9,
```

```
 1.0,
 0.1,
 0.2,
 0.3,
 0.4,
 0.5,
 0.6,
 0.7,
 0.8,
 0.9,
 1.0]
```

Two ways to do element-wise multiplication.

In [9]:
```
print(np.array(b) * 3) # np.array() converts the list b into an array

[element * 2 for element in b] # we use the for loop
```

```
[0.3 0.6 0.9 1.2 1.5 1.8 2.1 2.4 2.7 3. ]
```

Out[9]:
```
[0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0]
```

Loops

for loop

References:

- https://docs.python.org/3/tutorial/controlflow.html#for-statements
- https://docs.python.org/3/tutorial/controlflow.html#the-range-function

In [10]:
```
for ele in b: # note that b is a list
    print(ele * 2)
```

```
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
```

In [11]:
```
for index in range(len(b)): # another way
    print(b[index] * 2)
```

```
0.2
0.4
0.6
0.8
1.0
1.2
1.4
```

```
1.6
1.8
2.0
```

while loop

In [12]:

```
index = 0
while index < len(b):
    print(b[index] * 2)
    index += 1
```

```
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
```

User-defined functions

In [13]:

```
def myFunction(x):
    return np.exp(x) - 1

c = np.arange(0.01, 0.11, 0.01)
print('The new list is ', c)
myFunction(c)
```

```
The new list is  [0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.1 ]
```

Out[13]:

```
array([0.01005017, 0.02020134, 0.03045453, 0.04081077, 0.0512711 ,
       0.06183655, 0.07250818, 0.08328707, 0.09417428, 0.10517092])
```

In [14]:

```
def myFunc2(arg1, arg2 = 2): # multiple arguments
    return arg1 * arg2

print(myFunc2(c)) # the default value of arg2 is 2
print(myFunc2(c, 3))
```

```
[0.02 0.04 0.06 0.08 0.1  0.12 0.14 0.16 0.18 0.2 ]
[0.03 0.06 0.09 0.12 0.15 0.18 0.21 0.24 0.27 0.3 ]
```

In [15]:

```
myFunction_1 = lambda x: np.exp(x) - 1
myFunction_1(c)
```

Out[15]:

```
array([0.01005017, 0.02020134, 0.03045453, 0.04081077, 0.0512711 ,
       0.06183655, 0.07250818, 0.08328707, 0.09417428, 0.10517092])
```

Graph plotting

In what follows we will see that $e^x - 1 \approx x$ for small x.

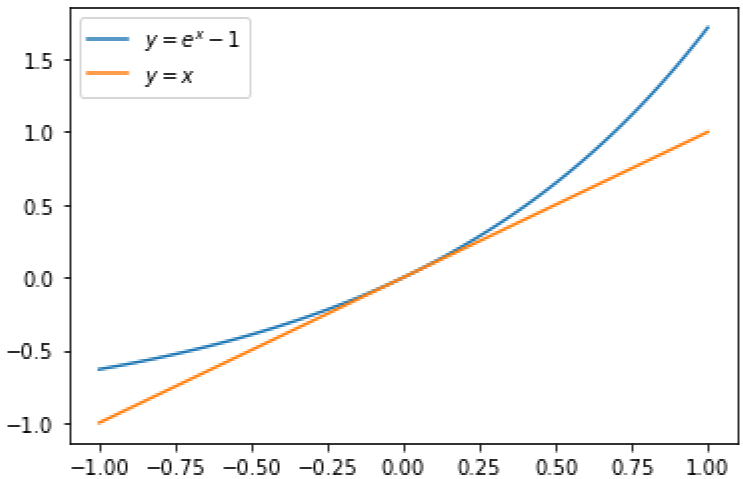
In [16]:

```
# importing the required module
import matplotlib.pyplot as plt

x = np.linspace(-1, 1, 101)
print(x)
y = myFunction(x)

plt.plot(x, y, label = '$y = e^x - 1$')
plt.plot(x, x, label = '$y = x$')
plt.legend()
plt.show()
```

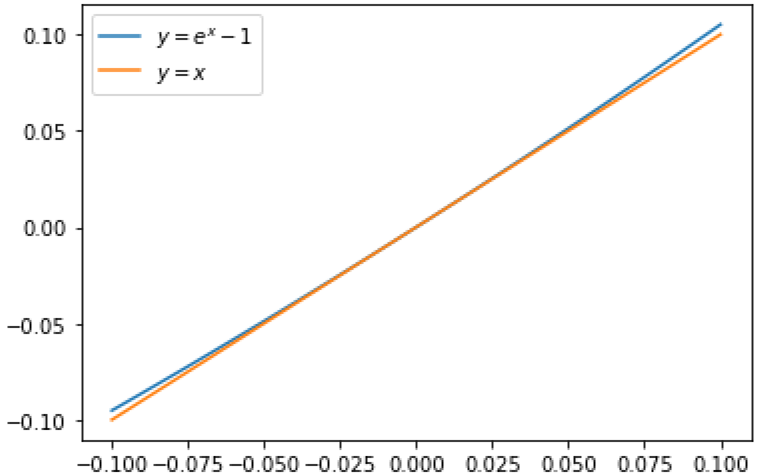
```
[-1.   -0.98 -0.96 -0.94 -0.92 -0.9  -0.88 -0.86 -0.84 -0.82 -0.8  -0.78
 -0.76 -0.74 -0.72 -0.7  -0.68 -0.66 -0.64 -0.62 -0.6  -0.58 -0.56 -0.54
 -0.52 -0.5  -0.48 -0.46 -0.44 -0.42 -0.4  -0.38 -0.36 -0.34 -0.32 -0.3
 -0.28 -0.26 -0.24 -0.22 -0.2  -0.18 -0.16 -0.14 -0.12 -0.1  -0.08 -0.06
 -0.04 -0.02  0.    0.02  0.04  0.06  0.08  0.1   0.12  0.14  0.16  0.18
  0.2   0.22  0.24  0.26  0.28  0.3   0.32  0.34  0.36  0.38  0.4   0.42
  0.44  0.46  0.48  0.5   0.52  0.54  0.56  0.58  0.6   0.62  0.64  0.66
  0.68  0.7   0.72  0.74  0.76  0.78  0.8   0.82  0.84  0.86  0.88  0.9
  0.92  0.94  0.96  0.98  1.  ]
```



In [17]:

```
x = np.linspace(-0.1, 0.1, 101)
print(x)
y = myFunction(x)
plt.plot(x, y, label = '$y = e^x - 1$')
plt.plot(x, x, label = '$y = x$')
plt.legend()
plt.show()
```

```
[-0.1   -0.098 -0.096 -0.094 -0.092 -0.09  -0.088 -0.086 -0.084 -0.082
 -0.08  -0.078 -0.076 -0.074 -0.072 -0.07  -0.068 -0.066 -0.064 -0.062
 -0.06  -0.058 -0.056 -0.054 -0.052 -0.05  -0.048 -0.046 -0.044 -0.042
 -0.04  -0.038 -0.036 -0.034 -0.032 -0.03  -0.028 -0.026 -0.024 -0.022
 -0.02  -0.018 -0.016 -0.014 -0.012 -0.01  -0.008 -0.006 -0.004 -0.002
  0.     0.002  0.004  0.006  0.008  0.01   0.012  0.014  0.016  0.018
  0.02   0.022  0.024  0.026  0.028  0.03   0.032  0.034  0.036  0.038
  0.04   0.042  0.044  0.046  0.048  0.05   0.052  0.054  0.056  0.058
  0.06   0.062  0.064  0.066  0.068  0.07   0.072  0.074  0.076  0.078
  0.08   0.082  0.084  0.086  0.088  0.09   0.092  0.094  0.096  0.098
  0.1  ]
```



Takeaway:

- $e^x - 1 \approx x$ when $x \approx 0$.