Practical Session 3

1. Aim of the Session

In this session, you will learn how you can make your programs understand and manipulate sets of numbers; you will explore more **Numpy**'s Ufuncs including Trigonometric functions, Exponents and logarithms; you will also learn how to define a function using a Lambda, which is one of the most useful and important features to know about in Python. Furthermore, you will learn how to find the limit of functions and the derivative of functions using Python.

2. Preparation

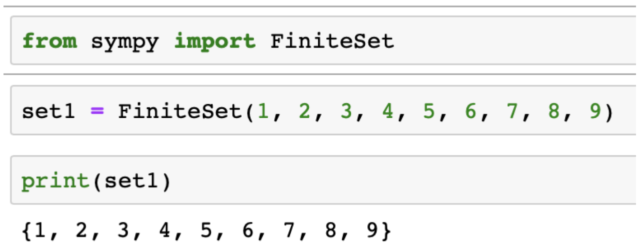
- Click the **window icon** on the bottom left corner:
- Search **Anaconda Navigator** by typing **Anaconda** from the search box under All Programs.
- Click on **Anaconda 3.6 Navigator**
- Click **Launch** under Jupyter from Anaconda Navigator;
- Once the Jupyter notebook is open, what you can see are files and/or folders under C:\Docs
- Click **New** from Jupyter, then select **Python 3** from the pop-up list. You should see a new page with a blank cell, where you can type in Python code.
- Click File and select Rename from the pop-up list, you may give the file a name you prefer, for example, practical3.

3. Sets

Let's go through some basic properties of sets by learning how to work with sets in Python using SymPy. SymPy (https://www.sympy.org/en/index.html) is a Python library for symbolic mathematics.

- Set construction

  To create a set in Python, you can use the **FiniteSet** class from the **sympy** package. For example, to create a set named *set1,* you can type the following:



```
from sympy import FiniteSet
```

```
set1 = FiniteSet(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
print(set1)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

You first import the **FiniteSet** class from SymPy and then create an object of this class by passing in the set members as arguments. You assign the variable *set1* to the set you have just created. You can store different types of numbers including integers, floating point numbers and fractions in the same set. You can also create an empty set without passing any arguments.

Task1:

Create two sets: s1 = {2/5, 4, 10.6} and s2={}

To generate a fraction, you can import the **Fraction** class from **fractions**, then use **Fraction** with two arguments, which are numerator and denominator.

The cardinality of a set is the number of members in the set, which you can find by using the **len()** function. For example:

```
len(set1)
```

```
9
```

Task2:

What is the cardinality of s1 and s2 separately? Answer this question by yourself first, then check your answer by doing it using Python.

You can also create a set by passing in a list of set members as an argument to FiniteSet as shown as follows:

```
'''create sets from lists'''
L = [0, 2, 4, 6]
set2 = FiniteSet(*L)
set2
```

```
{0, 2, 4, 6}
```

Sets in Python are like mathematical sets ignore any repeats of a member. For example,

```
L = [0, 2, 4, 6, 2]
set3 = FiniteSet(*L)
set3
```

```
{0, 2, 4, 6}
```

To check whether a number is in a set, you can use the *in* operator as follows:

```
4 in set1
```

```
True
```

Task3:

Check whether -1 and 4 is a member of s1 you have created previously.

Two sets are equal when they have the same elements. For example:

```
set4 = FiniteSet(2, 3, 7)
set5 = FiniteSet(3, 2, 7)
set4 == set5
```

```
True
```

- Subsets and Power sets

Recall that a set, A, is a subset of another set, B, if all members of A are also members of B. You can use the **is_subset()** method to check whether a set is a subset of another set in Python. For example,

```
s = FiniteSet(2)
s.is_subset(set4)
```

```
True
```

Task4:

1) What is the difference between subset and proper subset?
2) You can use the **is_proper_subset()** method to check whether a set is a proper subset of another set in python. Using the **is_proper_subset()** method to check whether *s* is a proper subset of *set4*.

The power set of a set, A, is the set of all possible subsets of A. You can use the **powerset()** method to obtain the powerset in Python.

Task5: *s1* and *s2* are two sets you have created earlier.

1) True or False: by writing down your answers on a paper first:

   $s1 \subset s2$

   $s2 \subset s1$
2) Using the **is_proper_subset()** method to answer the above questions
3) How many subsets does set *s1* have? What are they? Write down your answers in a paper.
4) Using the **powerset()** method to obtain the powerset of set *s1*. Compare the answer with your answer in 3).
5) Repeat 3) and 4) for set *s2.*

- Set operations

Set operations such as union, intersection and the Cartesian product allow you to combine sets in certain methodical ways. These set operations are extremely useful in real-world problem-solving situations when you have to consider multiple sets together. Later in this module, you'll see how to use these operations to calculate the probabilities of random events.

In SymPy, the union of two sets can be created using the **union()** method. For example, considering two sets, set2={0,2,4,6} and set4={2,3,7}, which you have seen previously:

```
set2.union(set4)
```

```
{0, 2, 3, 4, 6, 7}
```

In SymPy, the intersection of two sets can be created using the **intersect()** method. For example:

```
set2.intersect(set4)
```

```
{2}
```

Both of these operations can also be applied to more than two sets: For example: considering set2, set4 and set1={1,2,3,4,5,6,7,8,9} together:

```
'''how to find the intersection of three sets'''
set2.intersect(set4).intersect(set1)
```

```
{2}
```

In SymPy, you can find the Cartesian product of two sets by simply using the multiplication operator. To see each pair in that Cartesian product, you can iterate through and print them out using a *for* loop. For example, considering two sets: set4={2, 3, 7} and set6={1, 3, 5}:

```
'''Cartesian product'''
M = set4*set6
```

```
M
```

```
{2, 3, 7} x {1, 3, 5}
```

```
for elem in M:
    print(elem)
```

```
(2, 1)
(2, 3)
(2, 5)
(3, 1)
(3, 3)
(3, 5)
(7, 1)
(7, 3)
(7, 5)
```

Task6: given two sets $a = \{1, 2, 10, 99\}$ and $b = \{3, 10, 21, 1\}$, answer the following questions on a paper first, then check your answers with those obtained in Python.

1) What is the union of a and b?
2) What is the intersection of a and b?
3) What is the Cartesian product of a and b?

4. More about NumPy's Functions

NumPy provides a large number of useful ufuncs, some of which you have used in Practical 1. In this session, you will learn some of the most useful and common types for data scientists, including the trigonometric functions, the exponentials and their inverse, the logarithms.

- Trigonometric functions

  You can import numpy as np, and define an array of 100 angles in between $[0, 4\pi]$. For example,

```
import numpy as np

theta = np.linspace(0, 4*np.pi, 100)
```

Task7:

1) Generate corresponding trigonometric function values for 100 angles using **np.sin()**, **np.cos()** and **np.tan()**, separately.
2) produce two curves using different colours and different line styles on the same plot: that is, one for the angles against their sine values and the other one for the angles against their cosine values.

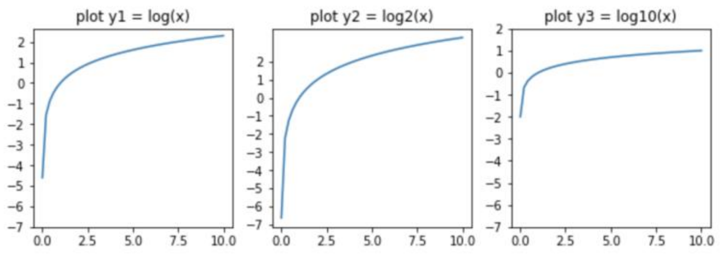
Inverse trigonometric functions are also available: **np.arcsin()**, **np.arccos()** and **np.arctan()**.

- Exponents and logarithms
  If you want to compute values for functions of $e^x, 2^x$ and 3^x using NumPy, you can use **np.exp(x)**, **np.exp2(x)** and **np.power(3,x)**, separately.

  You will learn 3 functions for computing logarithms through Task8.

  Task8:

  Download practical_3_logarithms.ipynb from Canvas. Click on Home from jupyter notebook. Click on **Upload** and select the file you have just downloaded. The file should be opened in a new tab. To make the program running, you need to finish the code by filling the correct code in the places with ????. Once you correct the code, you should see 3 plots similar to the following:

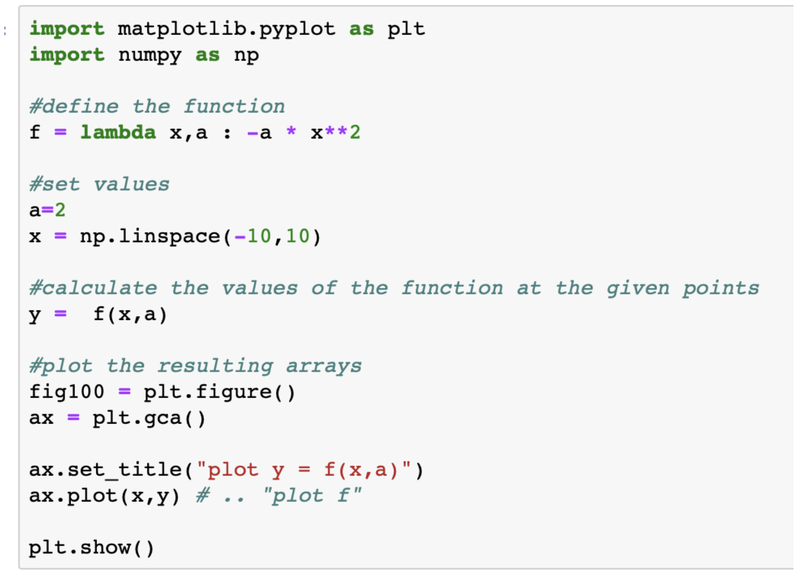


5. Lambda Functions

A lambda function looks like the following:

Target_values = lambda parameters: expression

where the keyword lambda is used to indicate that this is a lambda function. Any words following the **lambda** keyword are created as parameters. The colon is used to separate the parameters and the expression. The following shows an example. The aim of this program is to show a graph of function of $f(x) = -a * x^2$, where a =2. There are two parameters for this lambda function, x and a, where x is in the range of [-10 10] and a has a value of 2. The expression of this lambda function is $-a * x^2$.



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

#define the function
f = lambda x,a : -a * x**2

#set values
a=2
x = np.linspace(-10,10)

#calculate the values of the function at the given points
y =  f(x,a)

#plot the resulting arrays
fig100 = plt.figure()
ax = plt.gca()

ax.set_title("plot y = f(x,a)")
ax.plot(x,y) # .. "plot f"

plt.show()
```

Task9:

1) Define the sigmoid function ($f = \frac{1}{1+e^{-x}}$) using a Lambda function and plot the curve in the range of $x = [-10, 10]$.
2) Define the same function using the key words def and return. Evaluate the function value when $x = 0$ by calling the function you have defined.

6. Solving Calculus Problems

In this section, you will learn how to solve basic calculus problems using SymPy. We assume that all functions in this session you are calculating the derivative of are differentiable in their respective domains.

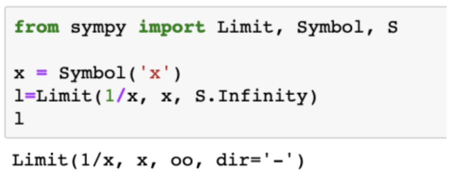
- Finding the limit of functions

  Consider a function $f(x) = 1/x$.

  Task10:

  Generate a graph of $f(x)$ when x is in the range of [10, 1000] with xlabel and ylabel.

  You can find limits of functions in SymPy by creating objects of the **Limit** class as follows:



```python
from sympy import Limit, Symbol, S

x = Symbol('x')
l=Limit(1/x, x, S.Infinity)
l
```

```
Limit(1/x, x, oo, dir='-')
```

In the above code, the first line imports the **Limit**, **Symbol**, and **S** classes. **S** contains the definition of some special values, such as positive infinity, negative infinity, and so on. In the next line, a symbol object, x, is created. The Limit object, l, is created with three arguments: the function expression, the variable and the value at which you want to calculate the function's limit. The result is returned with the oo symbol denoting positive infinity and the dir='-' symbol denoting that the limit is approached from the negative side.

To find the value of the limit, you can use the **doit()** method:

```
l=Limit(1/x, x, S.Infinity)
l.doit()
```

```
0
```

Task11:

Calculate $\lim_{x \to 0} \frac{1}{x}$

- Finding the derivations of functions

  The derivative of a function $y = f(x)$ is the rate of change in the dependent variable y with respect to the independent variable x. It is denoted as either $f'(x)$ or $\frac{dy}{dx}$. Consider a function $f(x) = 100x^2 + 4x + 9$. You can find the derivative of this function by creating an object of the **Derivative** class as follows:

```
from sympy import Symbol, Derivative
```

```
x = Symbol('x')
f =100*x**2+4*x+9
Derivative(f, x)
```

```
Derivative(100*x**2 + 4*x + 9, x)
```

First import the Symbol class and the Derivative class. Then define the independent variable and the function expression. An object of the Derivative class is created by passing with two arguments: the function and the variable. You can obtain the derivative by calling the **doit()** method:

```
df = Derivative(f, x).doit()
```

```
df
```

```
200*x + 4
```

You may calculate the value of the derivative at a particular value of x, say $x = 1$. You can use the **subs()** method:

```
df.subs({x:1})
```

```
204
```

Task12:

Consider a function $f(x) = x^3 + 2x^2 + \ln x$

1) Find the derivative of $f(x)$ by yourself and write down your answer on a paper.
2) Find the derivative of $f(x)$ in Python using SymPy and compare the result with your answer. Note that to call the **ln()** function, **import sympy** first, then use **sympy.ln()** function.
3) Calculate the value of the derivative at $x = 2$ using the **subs()** method.

7. Advanced: Affine transformation

In this section, you can further study how to rotate an image using what you have learnt about affine transformation by reading an interesting post from the following link:

https://stackabuse.com/affine-image-transformations-in-python-with-numpy-pillow-and-opencv/

You can do further experiments using your own selected images.

References

1. A. Saha: Doing math with Python: use programming to explore algebra, statistics, calculus and more! Publisher: No Starch Press. 2015
2. A Beginner’s guide to Understanding Python Lambda Functions https://www.makeuseof.com/tag/python-lambda-functions/