Introduction to Python
Lab 2

[Objective]

1. Install python and configure environment variable.
2. Learn python syntax and execution of your first python program in command line.
3. Learn how to use various python libraries (Numpy, Matplotlib, and so on).

[Software Installation]

1. Python : is an interpreted, high-level, general-purpose programming language. It is dynamically typed and garbage-collected and supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.
2. Miniconda: is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib and a few others.

Installation:

a) Download:

wget -c https://mirrors.tuna.tsinghua.edu.cn/anaconda/miniconda/Miniconda3-latest-Linux-x86_64.sh

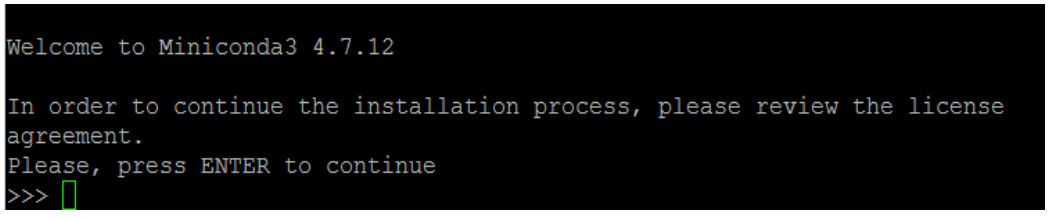
b) Add executable permissions

chmod +x Miniconda3-latest-Linux-x86_64.sh

c) run shell script

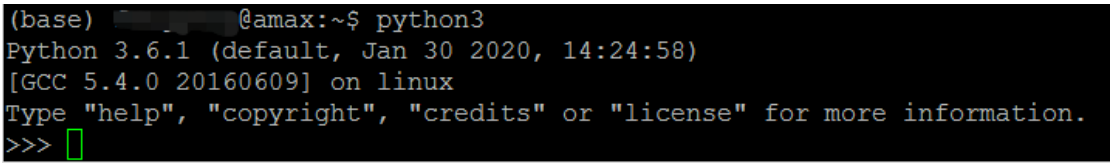
./Miniconda3-latest-Linux-x86_64.sh

d) Press ENTER or yes when prompted



e)Configure environment variable

source ~/.bashrc



Now, you can use python3

[Python Fundamentals]

Common built-in functions :

help(function): Get function help information

such as: help(print)

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

print(xxx): type information

input(): keyboard input

len(object): return the length of object

range(N) : creates a list of integers, typically used in a for loop

If condition statements:

```
if  judgement condition:
        statement
else:
        Statement
```

while loop statement:

```
while judgement condition:
        statement
```

for loop statement:

such as:

```
for i in range(10):
    print(i)
```

```
>>> for i in range(10):
...     print(i)
...
0
1
2
3
4
5
6
7
8
9
```

Common Data Types:

1. Number

Such as:

```
>>> 5+4
9
>>> 4.3-2
2.3
>>> 3*7
21
>>> 2/4
0.5
>>> 2//4
0
>>> 2**3
8
```

2. String

Strings in Python are enclosed in single quotes' or double quotes ", and backslashes \ are used to escape special characters.

The syntax for intercepting strings is as follows:

```
>>> str = "abcd"
>>> print(str[0:-1])    outputs all characters from the first to the penultimate
abc
>>> print(str[0])       outputs the first character of the string
a
>>> print(str[2:])      outputs all characters after the third character
cd
>>> print(str[2:4])     outputs characters starting from the third to the fourth
cd
```

3. List

A list is a comma-separated list of elements written between square brackets [].

Like strings, lists can also be indexed and truncated, returning a new list containing the desired elements.

```
>>> list=['a',2,'c',4]
>>> print(list)
['a', 2, 'c', 4]
>>> print(list)
['a', 2, 'c', 4]
>>> print(list[0])
a
>>> print(list[1:3])
[2, 'c']
>>> print(list[2:])
['c', 4]
```

Common function:(you can use help() function to get help information)

1. list.append(obj)
2. list.insert(index,obj)
3. list.extend(list)
4. list.index(obj,start,end)

5. list.remove(elements）
6. list.pop(index)
7. list.count(obj)
8. list.clear()

```
>>> list=[1,2,3,4]
>>> help(list.clear)
Help on built-in function clear:

clear() method of builtins.list instance
    Remove all items from list.
```

4. Tuple

A tuple is similar to a list except that its elements cannot be modified.

Tuples are written in parentheses () and elements are separated by commas.

```
>>> tuple=(123,'abc','345','efg')
>>> print(tuple)
(123, 'abc', '345', 'efg')
>>> print(tuple[0])
123
>>> print(tuple[1:3])
('abc', '345')
>>> print(tuple[2:])
('345', 'efg')
```

Common function:

1. tuple.index()
2. tuple.count()
3. min(tuple)
4. max(tuple)
5. tuple(list)
6. len(tuple)

5. Set

A set is made up of one or more units of various sizes. The things or objects that make up a set are called elements or members.

The basic functionality is to perform membership testing and remove duplicate elements.

```
>>> stu = { 'Jim', 'Mary', 'Tom', 'Jack', 'Rose'}
>>> if 'Rose' in stu :                    determine if it is in the set
...     print('Rose is in set')
... else :
...     print('Rose is not in set')
...
Rose is in set
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> print(a - b)                          difference set
{'d', 'b', 'r'}
>>> print(a | b)                          union set
{'d', 'a', 'z', 'r', 'b', 'c', 'm', 'l'}
>>> print(a & b)                          intersection
{'a', 'c'}
```

Common function:

1. set.add()
2. set.remove()
3. set.pop()
4. set.clear()
5. set1.union(set2)
6. set1.intersection(set2)
7. set1.difference(set2)

6.Dictionary

A dictionary is another very useful built-in data type in Python.

A list is an ordered collection of objects, and a dictionary is an unordered collection of objects. The difference between the two is that the elements in the dictionary are accessed by keys, not by offsets.

A dictionary is a type of mapping. The dictionary is identified by {} and it is an unordered set of keys: values.

```
>>> dict = {'name': 'lilei','code':1, 'age':24}
>>> print(dict['name'])          output value with key 'name'
lilei
>>> print(dict.keys())           output all keys
dict_keys(['name', 'code', 'age'])
>>> print(dict.values())         output all values
dict_values(['lilei', 1, 24])
>>> for k,v in dict.items():     traverse the dictionary
...     print(k,v)
...
name lilei
code 1
age 24
```

[Python script example]

1. vi first.py

```
list=[1,2,3]                    define a list containing elements 1,2,3


dict={'one':('a','b','c')}      define a dictionary

tuple=tuple(list)               convert list type to tuple type

dict['two']=tuple               add new elements to the dictionary

for i in dict.keys():
    print(i,dict[i],len(dict[i]))
~
~ traverse the dictionary and then output the keys,values and the length of values
~
```

2. Save , exit and add executable permissions

press [esc]

press :wq and enter key

chmod +x first.py

3. Execute python script in shell script

vi run.sh

```
# !/bin/bash

key="python"
if [ $key = "python" ] ; then
        python3 first.py
else
        echo "nothing to do"
fi
~
~
~
~
~
~
~
```

4. Save , exit and add executable permissions

press [esc]

press :wq and enter key

chmod +x run.sh

5 ./run.sh

```
(base)  @amax:~$ ./run.sh
one ('a', 'b', 'c') 3
two (1, 2, 3) 3
```

[Exercise]

a)

1. Write a python file named lab2_a.py. Calculate the area of the trapezoid: input the upper and lower bases and height, and output the area. The area requires two significant digits.
2. If you enter something other than a number, ask for it again

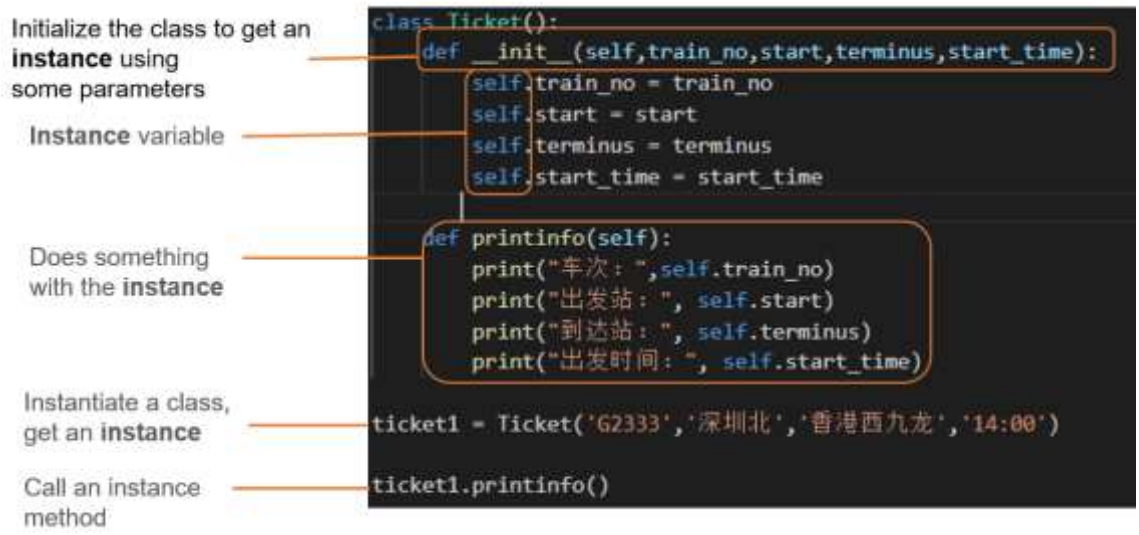
b)

1. Use a dictionary to achieve a record of a transcript(five students)
2. Dictionary keys are student names
3. Dictionary values are student grades that contain Mathematics and Chinese.
4. Enter value 0 or 1 when you run lab2_b.py
5. Use conditional statements to judge, if the input is 1, output all students' names and grades, otherwise output the average grade of Mathematics and Chinese respectively.

[Python Classes Introduction]

To learn how to definite and use Python classes.

Here is an example:



[Library Introduction]

Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

You can link http://www.numpy.org/ or https://www.numpy.org.cn/ to see the tutorial.

You can check if the Numpy library has already been installed in your python by

```
[tom@nerv ~]$ python
Python 2.7.16 (default, Mar 11 2019, 18:59:25)
[GCC 8.2.1 20181127] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>>
```

If no error messages occur, that means the library has already been installed.

If the library has not yet installed, you need to install it by typing
python3 -m pip install numpy in command line.
Please do the installation in your own virtual environment.
(Minicoda has been installed in lab2)

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

You can link https://matplotlib.org/ or https://www.matplotlib.org.cn/ to see the tutorial.

Installation: Type python3 -m pip install matplotlib in command line. (Minicoda has been installed in lab2)

There are other libraries like:

Scipy: a Python-based ecosystem of open-source software for mathematics, science, and engineering.

You can link https://docs.scipy.org/doc/scipy/reference/ to see the tutorial.

Panda: a fast, powerful, flexible and easy to use open source data analysis and manipulation tool.

You can link https://pandas.pydata.org/pandas-docs/stable/ to see the tutorial.

Scikit-learn: a free software machine learning library for the Python programming language

You can link https://scikit-learn.org/stable/ to see the tutorial.

Other libraries: **Seaborn, Gensim, Scrapy**…

These libraries can be installed in the same way.

[NumPy Fundamentals]

First, login up server and get into python3 environment.

(1) Array creation

NumPy's main object is the homogeneous multidimensional array.

You can create an array from a regular Python list or tuple using the array function. Import Numpy first.

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
```

The attributes of an ndarray object are:

ndarray.ndim, ndarray.shape, ndarray.size, ndarray.dtype, ndarray.itemsize

For example:

```
>>> a = np.arange(15).reshape(3, 5)   create an array
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape   the dimensions of the array
(3, 5)
>>> a.ndim   the number of axes
2
>>> a.dtype.name describing the type of the elements in the array
'int64'
>>> a.itemsize the size in bytes of each element of the array
8
>>> a.size  the total number of elements of the array
15
```

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

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

```
>>> c = np.array( [ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

The function zeros creates an array full of zeros, the function ones creates an array full of ones, and the function empty creates an array whose initial content is random and depends on the state of the memory. By default, the dtype of the created array is float64.

```
>>> np.zeros( (3,4) )
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
>>> np.ones( (2,3,4), dtype=np.int16 )                # dtype can also be specified
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)
>>> np.empty( (2,3) )                                 # uninitialized
array([[  3.73603959e-262,   6.02658058e-154,   6.55490914e-260],  # may vary
       [  5.30498948e-313,   3.14673309e-307,   1.00000000e+000]])
```

Also, we can use reshape(m,n) to modify the size

```
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

You can also learn how to use np.zeros_like,(ones_like, empty, empty_like, arange, linspace) with the tutorial.

(2) Basic operations:

```
>>> a = np.array( [20,30,40,50] )
>>> b = np.arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True,  True, False, False])
```

Specifying the axis parameter you can apply an operation along the specified axis of an array:

```
>>> b = np.arange(12).reshape(3, 4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)                          # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)                          # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)                       # cumulative sum along each row
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

Matrix product can be performed like:

```
>>> A = np.array( [[1,1],
...                [0,1]] )
>>> B = np.array( [[2,0],
...                [3,4]] )
>>> A * B                           # elementwise product
array([[2, 0],
       [0, 4]])
>>> A @ B                           # matrix product
array([[5, 4],
       [3, 4]])
>>> A.dot(B)                        # another matrix product
array([[5, 4],
       [3, 4]])
```

You can also learn how to use np. all(any, sum, sort, argmax, argmin, argsort, average, bincount, ceil, clip…) with the tutorial.

PS: For some tasks, like adding a scalar, dot product, concatenation, numpy is around 5 to 100 times faster than standard python list.

(3) Indexing, Slicing and Iterating

One-dimensional arrays can be indexed, sliced and iterated over, much like lists and other Python sequences.

```
>>> a = np.arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
# equivalent to a[0:6:2] = 1000;
# from start to position 6, exclusive, set every 2nd element to 1000
>>> a[:6:2] = 1000
>>> a
array([1000,    1, 1000,   27, 1000,  125,  216,  343,  512,  729])
>>> a[ : :-1]                                    # reversed a
array([  729,  512,  343,  216,  125, 1000,   27, 1000,    1, 1000])
>>> for i in a:
...     print(i**(1/3.))
```

Multidimensional arrays can have one index per axis. These indices are given in a tuple separated by commas:

```
>>> def f(x,y):
...     return 10*x+y
...
>>> b = np.fromfunction(f,(5,4),dtype=int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2,3]
23
>>> b[0:5, 1]                       # each row in the second column of b
array([ 1, 11, 21, 31, 41])
>>> b[ : ,1]                        # equivalent to the previous example
array([ 1, 11, 21, 31, 41])
>>> b[1:3, : ]                      # each column in the second and third row of b
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])
```

(4) Broadcasting rules

Broadcasting：In math, operations like dot products and matrix addition require the same dimensions. In numpy, this is not the case. Up until now, we have used 1d and 2d ndarrays, representing vectors and matrices, and numpy acts as we would expect. However, the operations we have described work even when the two inputs do not have standard shapes, given by a set of very specific rules.
General broadcasting rules:

- Write out the shapes of each ndarray
- Starting from the back, that dimension has compatible values if either:
  - They are the same value, or
  - One of them is a 1
- The size of the resulting array is the maximum along each dimension
- Note: the two ndarrays do not need to have the same number of dimensions

Here are some examples:
In this case, we add a scalar to an ndarray. Numpy automatically adds the scalar to every single element.

```
>>> x = np.array([1,  10,  100, 200])
>>> x - 200
array([-199, -190, -100,    0])
```

When 2 arrays have diffirent shapes but one dimension is same, such as (3,2) and (2,)

```
array([-199, -190, -100,    0])
>>> A = np.array([[1,2], [3,4], [5,6]])
>>> B = np.array([1,1])
>>> A + B
array([[2, 3],
       [4, 5],
       [6, 7]])
```

From the np.matmul() documentation, If either argument is N-D, N > 2, it is treated as a stack of matrices residing in the last two indexes and broadcast accordingly. If 2 arrays have shapes (3, 4, 5, 6) and (4, 6, 7):

- According to the documentation, the last two dimensions represent matrices, so we take those out and broadcast the rest: (3, 4) and (4,)
- Using our broadcasting rules, the result of broadcasting these shapes will be (3, 4)
- Matrix multiplication results in a matrix of shape (5, 7)
- Our output will have shape (3, 4, 5, 7)

(5) Stacking together different arrays

Operations contain: np.vstack(), np.hstack(), np.newaxis(), np.concatenate().

Try to understand the meanings of these operations.

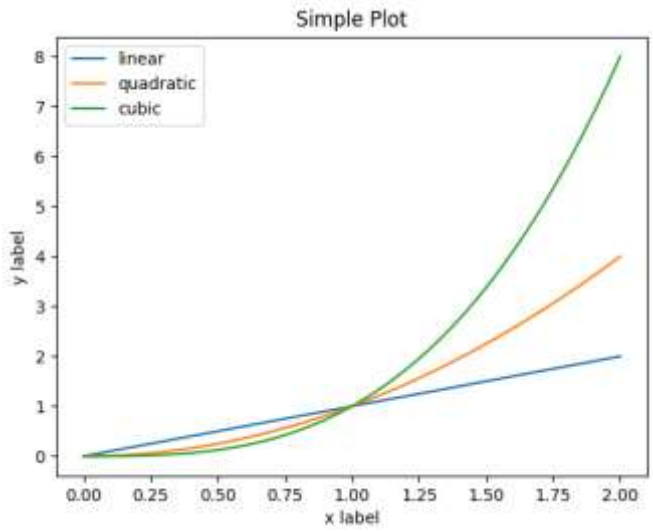
```
>>> a = np.array([4.,2.])
>>> b = np.array([5.,3.])
>>> np.vstack((a,b))
array([[4., 2.],
       [5., 3.]])
>>> np.hstack((a,b))
array([4., 2., 5., 3.])
>>> a[:,np.newaxis]
array([[4.],
       [2.]])
>>> np.hstack((a[:,np.newaxis],b[:,np.newaxis]))
array([[4., 5.],
       [2., 3.]])
>>> np.concatenate((a,b),axis=0)
array([4., 2., 5., 3.])
```

[Matplotlib Fundamentals]

Learn to plot a simple example, run the codes and you can get the result in the right.

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

x = np.linspace(0, 2, 100)
 # Use plt.plot() plot(x,y)
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')
 # Set the label of axis
plt.xlabel('x label')
plt.ylabel('y label')
 # Set the title of figure
plt.title("Simple Plot")
 # Show legend
plt.legend()
 # Display
plt.show()
```



Try to run the following codes and see what will be plotted. After two examples, learn how to plot a simple figure.

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g', alpha=0.75)


plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```

[Exercise]

a)

Randomly create 2 numpy arrays: A, B with shape(5,4) and elements integer from 0-9 , swap the first row of two arrays, use function concatenate() to reshape a C with shape(10,4), print the shape, maximum, minimum, mean and sum of C.

b)

Plot the function $f(x) = sin^2(x-2)e^{-x^2}$ over the interval [0; 2]. Add proper axis labels, a title, etc.