Lecture 3 Python Basics

Douglas Chung

National Chengchi University

1 October 2021

1 Object Types

1.1 Integers (int)

Integers are numbers that can be written without a fractional component.

```
[ ]: x = 2
     type(x)
```

If we add two or subtract integers together, Python returns an integer:

```
[ ]: y = x - 1 - 3
     y
```

Multiplication of integers will result in integer:

```
[ ]: 4 * 2
```

```
[ ]: type(y)
```

1.2 Floats (float)

Floats (floating point numbers) are real numbers stored under the computer's binary numeral system.

```
[ ]: z = 2.5
     type(z)
```

Mathematical operations between integer and float will result in float:

```
[ ]: x + z
```

Division of integers will result in float:

```
[ ]: 4 / 2
```

```
[ ]: 5 / 2
```

1.3 Strings (str)

Strings are characters.

```
[ ]: w = 'word'
     type(w)
```

We can combine strings by addition:

```
[ ]: w + w
```

```
[ ]: w + "200"
```

```
[ ]: "1" + "2"
```

```
[ ]: w + 200
```

1.4 Booleans (bool)

Booleans can either be “True” or “False”. Python is case sensitive so “true”, “TRUE”, “false”, and “FALSE” are not booleans.

```
[ ]: type(True)
```

```
[ ]: True == False
```

```
[ ]: TRUE
```

2 Some math operations

2.1 Exponentiation

$ 2ˆ2 = 4 $

```
[ ]: 2 ** 2
```

2.2 Modulus

```
[ ]: 4 % 2
```

```
[ ]: 5 % 2
```

```
[ ]: 5.1 % 2
```

2.3 Floor division

```
[ ]: 18 // 6
```

```
[ ]: 11 // 5
```

```
[ ]: 9 // 5
```

2.4 Relational operators

```
[ ]: a = 1
     b = 2.1
     c = 2.1
     a == b
```

```
[ ]: a != b
```

```
[ ]: a > b
```

```
[ ]: a < b
```

```
[ ]: c >= b
```

```
[ ]: c <= a
```

```
[ ]: ((a < b) and (c >= b))
```

```
[ ]: ((a < b) or (c <= a))
```

```
[ ]: ((a < b) and (c <= a))
```

```
[ ]: if a > b:
         print('yes') # indentation required
     else:
         print('no')
```

```
[ ]: if a > b:
         print('a > b')
     elif b > a:
         print('b > a')
     else:
         print('no')
```

3 Functions

Functions take and process inputs then give outputs. For example:

$ f(x,y) = x + y $

[ ]:
```python
def addition(x,y):
    return x + y
```

[ ]:
```python
addition(1,3)
```

4 Comments

Use # to make comment to your code.

[ ]:
```python
X = 100 # assign 100 to X
# Python doesn't read anything after #
```

5 Data types

5.1 Lists

Lists can store objects of different types and lists are mutable.

[ ]:
```python
MyList = ['a',1,'2', True]
type(MyList)
```

[ ]:
```python
MyList
```

Python counts from 0

[ ]:
```python
MyList[0]
```

[ ]:
```python
MyList[-1]
```

[ ]:
```python
type(MyList[0])
```

[ ]:
```python
type(MyList[2])
```

[ ]:
```python
MyList = ['a',1,'2', True]
MyList.append("abc")
MyList
```

[ ]:
```python
MyList = ['a',1,'2', True]
MyList.insert(0,"abc")
MyList
```

[ ]:
```python
MyList = ['a',1,'2', True]
MyList.append([1])
MyList
```

[ ]:
```
MyList = ['a',1,'2', True]
MyList.extend([1])
MyList
```

[ ]:
```
list1 = [1, 2, 3]
list2 = [2, 4, 6]
list1 + list2
```

5.2 Dictionary

[ ]:
```
Person = {'Name': 'Tom', 'Age': 20}
Person
```

[ ]:
```
type(Person)
```

[ ]:
```
Person['Name']
```

6 Flow controls

6.1 For loop

For loop is used to iterate over elements of a sequence.

[ ]:
```
for i in MyList:
    print(i)
```

If we want to do the same thing N times, we can use a for loop on a range(start, stop[, step]). We can do it forward or backward.

[ ]:
```
for i in range(2):
    print("This is loop no: " + str(i))
```

[ ]:
```
for i in range(1,3):
    print("This is loop no: " + str(i))
```

[ ]:
```
for i in range(1,-2,-1):
    print("This is loop no: " + str(i))
```

6.2 Nested for loops

[ ]:
```
for i in range(2):
    print('Outter loop: ' + str(i))

    for j in range(3):
        print('Inner loop: ' + str(j))
```

[ ]:

```python
for i in range(2):
    print('Outter loop: ' + str(i))

    for j in ['one', 'two']:
        print('Middle loop: ' + str(j))

        for k in ['A','B']:
            print('Inner loop: ' + str(k))
```

6.3 While loop

[ ]:

```python
i = 3

while i > 0:
    i -= 1
    print(i)
```

[ ]:

```python
i = 3

while i > 0:
    i = i - 1
    print(i)
```

[ ]:

```python
i = 0

while i < 5:
    i += 2
    print(i)
```

7 Numpy

7.1 Array

[ ]:

```python
import numpy as np

array1 = np.array(list1)
array2 = np.array(list2)
type(array1)
```

[ ]:

```python
array1 + array2
```

7.2 2D array

$$M = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$$

```
[ ]: mat = np.array([[2, 3], [1, 4]])
     mat
```

```
[ ]: mat2 = np.array([[1, 3, 3], [1, 4, 3]])
     mat2
```

7.3 Diagonal elements of a 2D array (~matrix)

```
[ ]: np.diag(mat)
```

7.4 Matrix multiplication

```
[ ]: mat @ np.diag(mat)
```

```
[ ]: mat @ mat
```

```
[ ]: mat @ mat2
```

```
[ ]: mat2 @ mat
```

7.5 Dot product

This is the same as matrix multiplication for 2D arrays.

```
[ ]: np.dot(mat, np.diag(mat))
```

```
[ ]: np.diag(mat) @ mat
```

7.6 Identity matrix

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
[ ]: np.eye(2)
```

```
[ ]: mat @ np.eye(2)
```

7.7 Inverse of matrix

$$M^{-1}M = MM^{-1} = I$$

```
[ ]: from numpy.linalg import inv
     inv(mat)
```

```
[ ]: mat @ inv(mat)
```

[ ]:
```
inv(mat) @ mat
```

7.8 Linear space

np.linspace(start,stop,steps)

[ ]:
```
np.linspace(0, 10, 3)
```

8 Pandas

8.1 Series

[ ]:
```
import pandas as pd
d = {'a': 1, 'b': 2, 'c': 3}
ser = pd.Series(data=d, index=['a', 'b', 'c'])
ser
```

[ ]:
```
ser['c']
```

8.2 .loc

Select elements based on their index labels.

[ ]:
```
ser.loc['b']
```

8.3 .iloc

Select elements based on their index locations.

[ ]:
```
ser.iloc[0]
```

8.4 DataFrame

[ ]:
```
df = pd.DataFrame({'A':[1,2,3],'B':[3,4,5]})
df
```

[ ]:
```
pd.DataFrame(ser)
```

8.5 Subsetting into series

[ ]:
```
print(df.loc[:,'A'])
type(df.loc[:,'A'])
```

8.6 Subsetting into DataFrames

[ ]:
```
print(df.loc[:,['A']])
type(df.loc[:,['A']])
```

8.7 Using .iloc

Select one element.

[ ]:
```
print(df.iloc[0,1])
type(df.iloc[0,1])
```

Extract series.

[ ]:
```
print(df.iloc[1,0:2])
type(df.iloc[1,0:2])
```

[ ]:
```
type(df.loc[0,['A']])
```

[ ]:
```
df.shape
```

[ ]:
```
df.ndim
```

[ ]:
```
ser.shape
```

[ ]:
```
ser.ndim
```

8.8 Convert pandas series to numpy and reshape

[ ]:
```
ser.to_numpy().reshape(1,3)
```

[ ]:
```
ser.to_numpy().reshape(1,3) @ df
```