1928

K. N. Toosi University of Technology

Department of electrical engineering

Course: Linear Algebra

Professor: Dr. Mehsan Tavakoli

Lab Instructions - session 0

Introduction to Python

Deadline: -

Spring 2024

Prepared by

Ramin Tavakoli

Open an interactive Python environment (python shell, ipython shell, or Jupyter notebook), run the following commands, and observe the output:

Data structures are a way of organizing and storing data so that they can be accessed and worked with efficiently. They define the relationship between the data, and the operations that can be performed on the data. There are many various kinds of data structures defined that make it easier for data scientists and computer engineers alike to concentrate on the main picture of solving larger problems rather than getting lost in the details of data description and access.
In this session, you'll learn about the various Python data structures and see how they are implemented:

- Abstract Data Type and Data Structures
- Primitive Data Structures
    - Integers
    - Float
    - Strings
    - Boolean
- Non-Primitive Data Structures
    - Arrays
    - Lists
    - Tuples
    - Dictionary
    - Sets

1. Values and types

```
f = 1.2         # float
i = 23          # int
s = "Hi!"       # string
b = True        # boolean
type(i)
type(f)
type(b)
type(s)
```

2. Conversion between types

```
str(i)
str(f)
int(f)
int('123')
float(i)
float('1.1')
int(t)
int(False)
```

3. Operators and Operand

```
op1 = 4
op2 = 5
op1 * op2
op1 + op2
op2 ** op1
op1,op2 = op2,op1
c,d = 1,op1
op1,op2 = op1 - op2 , op1 + op2
5 // 2.0
5 / 2.0
5 / 2
5 % 2
5 // 2
k = op1**3 + op2**2
print(op1)
op1 *= 10
print(op1)
op1 -= 2
print(op1)
op1 = 20
op2 = 30
op1 == op2
op1 < op2
op1 < -5 or op2 > 5
op1 >= op2 - 5
op1 < op2 and op1 == 20
not (op1 < op2 and op1 == 20)
```

4. String operations

```
s2 = "salam"
s = 'salam'
print(s)
print(s2)
# s3 = 'it's a nice day'
s3 = "it's a nice day"
s == s2
len(s)
r = '123'
s+r
s + ' ' + r
s*8
```

```python
s = 'my_name_is Ramin'
print(s[0])
s[1]
s[2]
s[-1]
s[-3]
s[1:4]
s[2:]
s[:4]
s[:-1]
s[1:7]
s[1:7:2]
i = 1
d = 2.2
f = 1.1
s = "Salam"
print("num = %d"%i)
s1 = "%.2f %i %s"%(f,i,s)
print(s1)
print("float=%f, integer=%i, and string=%s."%(d,i,s))
"float={}, integer={}, and string={}.".format(d,i,s)
f"float={d}, integer={i}, and string={s}."
f"float={d*2}, integer={20-4*i}, and string={s+s}."
```

5. Tuples

Tuples are used to store multiple items in a single variable. Tuple is one of 4 built-in data types in Python used to store collections of data. A tuple is a collection which is ordered and **unchangeable**.
Tuples are written with round brackets.

```python
t1 = ('a', 'b', 'c', 'd', 'e')
type(t1)
t1 = ('a',)
type(t1)
t2 = ('a')
type(t2)

t = (23,12,14)
p = (2, 'abc', 13.2)
print(p)
p[0]
p[1]
p[-1]
```

```
len(p)
p + (10,20,30)
t+p
p*3
# p[1] = 100   # TypeError: 'tuple' object does not support item
assignment
a,b,c = p
print(a,b,c)
t = tuple('Ramin')
print(t)
```

6. Lists

The list is a sequence data type which is used to store the collection of data.
Python Lists are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). In simple language, a list is a collection of things, enclosed in [ ] and separated by commas.

```
l = [238.11, 237.81, 238.91]
type(l)
l
i = 12
t = 2
l = [i,t,'hello',2]
l # list is  ...
l = l + [True]
l
l = l + [1,23,5]
l
len(l)
l.append('nice!')
l
l.insert(0,10)
l
l.insert(2,11)
l
11 in l
'hello' in l
100 not in l
l.extend(['a', 'b', 'c'])
l
'''
```

```
append() takes a single element and adds it as is, while extend() takes
an iterable (like a list) and appends its
elements individually to the list.
'''
l.pop()
l
k = l.pop(2)
l
k
l=[100,101,102,103,104,105,106,107,108,109,110,111,112]
l[0]
l[1]
l[5]
l[-1]
l[-2]
print(l[1:5])
l[9:1]
l[1:8]
l[1:10:2]
l[::3]
l[10:1]
l[10:1:-1]
l[::-1]
104 in l
130 in l
l = [14,9,7,23,1,13]
l[4] = 100
print(l)
l[1] = 100
l
l.reverse()
l
l.sort()
l
l = [4,1,7,2,0]
t = l
s = l.copy()    #  .copy() is a method in OOP
l[1] = 100
print("l =",l)
print("t =",t)
print("s =",s)
''' this is a diffrences between deep and shallow copy, you don't need
to know what it means.'''
l = [4,1,7,2,0]
```

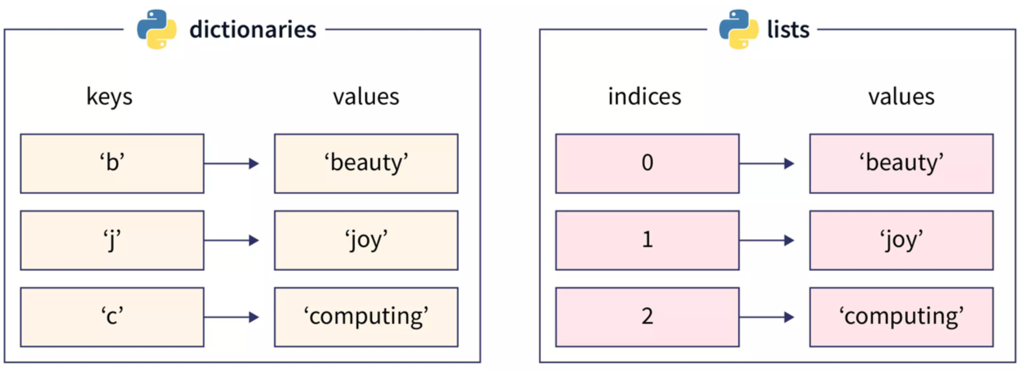
```
t = l[:]
l[1] = 100
print(l)
print(t)
e = enumerate(l)
print(e)
list(e)
z = zip(l,t)
z
list(z)
```

7. Range function

```
r1 = range(10)
r2 = range(2,10)
r3 = range(2,20,3)
r4 = range(20,2,-1)
r1
print(type(r1))
list(r1)
list(r2)
list(r3)
list(r4)
tuple(r1)
```

8. Dictionaries

A *dictionary* is like a list, but more general. In a list, the index positions have to be integers; in a dictionary, the indices can be (almost) any type.
You can think of a dictionary as a mapping between a set of indices (which are called *keys*) and a set of values. Each key maps to a value. The association of a key and a value is called a *key-value pair* or sometimes an *item*.



```
dictionary_unique = {"a": "alpha", "o": "omega",
                     "g": "gamma", "g": "beta"}
print(dictionary_unique)
#dictionary_immutable = {["a","b","c"]: "alpha", "o": "omega", "g":
"gamma", "g": "beta"}
''' dictionary keys must be immutable types and list is a mutable type.
'''

d = {1: 'Salam', 8: 1.4}
d[1]
d[8]
# d[2]
d[2] = 444
d
d[2]
d['K'] = 99
d['Ali'] = 'passwd'
d['Ali']
print(d)
d.keys()
d.values()
d.items()
d
del d['K']
d
dictionary_nested = {"datacamp":{"Deep Learning": "Python", "Machine
Learning": "Pandas"},"linkedin":"jobs","nvidia":"hardware"}
dictionary_nested
dictionary_nested['datacamp']['Deep Learning']
```

Open a Python IDE (IDLE editor, vs code, spyder, pyCharm, etc.)., run the following commands, and see the output.

9. Conditional execution

```
i = 12
b = 1
if i == 12:
    b = 2
print(b)
```

```python
i = 12
if i == 10:
    print('YES')
    print('i equals 10')
else:
    print('NO')
    print('i is not equal to 10')

i = 11
b = 'salam'
if i == 12:
    print('Twelve')
elif i == 11 and b == 'hi':
    print('Eleven-hi')
elif i > 10 and b == 'salam':
    print('SALAAAMM!!')
else:
    print('None')
```

10. Iteration

```python
i = 4
while i > 0:
    print(i)
    i = i - 1
print('Done!')
```

```python
n = 10
while True:
    print(n, end=' ')
    n = n - 1
print('Done!')
```

```python
while True:
    line = input('> ')
    if line[0] == 'No':
        continue
    if line == 'Yes':
        break
    print(line)
print('Done!')
```

```
list = [10,2,3,40.2, 'Hi']
for k in list:
    print(k)
for j in range(2,20):
    print("iterate",j,":", j*2)
```

```
p = [1,2,3]
q = ['first','second','Third']

for i in range(len(p)):
    print(p[i],q[i])
    list = [10,20,1.2, 'salam']
    for j in list:
        print(j)
    for i, j in enumerate(list):
        print(i,j)
    l1 = [10,20,30, 'salam']
    l2 = ['a','b','c', 'aleik']
    for i,j in zip(l1,l2):
        print(i,j)
    for k in range(2,20):
        print(k, k*k)
```

11. Functions

```
''' Built-in functions '''
max('Hello world')
min('Hello world')
len('Hello world')
int('32')
float(32)

# function with two arguments
def add_numbers(num1, num2):
    sum = num1 + num2
    return sum

# function call with two values
print(add_numbers(5, 4))
print(add_numbers(3,3.1))
print(add_numbers('abc', '123'))

def sum(l):
    s = 0
    for k in l:
        s += k
    return s
l = [1,2,4,8,16]


print(sum(l))
print(sum(range(10))
print(sum(2)) # error
```

- Default argument values

```
def add(a,b=1):
    return a+b
print(add(20,4))
print(add(20))
```

- Call by name

```
def sub(n1,n2):
    return n1-n2
print(sub(20,4))
print(sub(n1=20,n2=4))
print(sub(n2=20,n1=4))
```

- Using python modules

```
import math

num = 4
print(math.sqrt(num))

print(math.cos(0), math.exp(0))
print(math.pi)
print(math.cos(math.pi))

import math as m
print(m.tan(0))

from math import sin, cos, exp, pi
print(exp(1), sin(pi/2))

from math import *
```

```
print(pi)
print(e)
print(log(e))
print(cos(log(1)))
print(tan(pi/4))
```

12. List Comprehensions

```
iterable = [2, 2, 32]
list_variable = [x for x in iterable]

list = [10, 20, 30]
list2 = [2*i for i in list]
print(list2)
t = [i*i for i in range(10)]
print(t)
t = [i*i for i in range(10) if i % 2 == 0]
print(t)
```

Task 1

If we want to display a matrix in Python, one of the ways in front of us is to use a list of rows of a matrix (a list of lists). For example, the matrix

$$\begin{bmatrix} 1 & 2 & 2 \\ -1 & 6 & 4 \\ 0 & 4 & -3 \end{bmatrix}$$

can be represented as:

```
                                    A = [ [1 2 2],
                                          [-1 6 4],
                                          [0 4 -3] ]
```

Your task is to write a function named mul that receives two matrices as arguments and returns their product as a matrix:

```
def mult(A,B):
    """
    returns the product of the matrix A by the matrix B
    returns the empty list [] if the matrix dimensions
    are not consistent for multiplication
    """
    # function body
# test the function

P = [[0, 1, 0],
     [1, 0, 0],
     [0, 0, 1]]

B = [[1, 1, 0],
     [0, 2, 3],
     [9, 0, 3]]

C = [[1, 2],
    [3, 1.5],
    [0, 5/2.0]]

print(mul(P,B))
print(mul(B,P))
print(mul(P,C))
```

The output of the code above is:

```
[[0, 2, 3], [1, 1, 0], [9, 0, 3]]
[[1, 1, 0], [2, 0, 3], [0, 9, 3]]
[[3, 1.5], [1, 2.0], [0, 2.5]]
```

If you look carefully, you will notice that in multiplication of P@B, the rows of matrix B have been swapped and in B@P the column of matrix B have been swapped. We called matrix P **Permutation matrix.**

- The arguments A and B must be of the above format (nested list). Do not use numpy arrays or matrices.
- Assume each matrix is consistent (rows are of the same size). But the dimensions might be inconsistent for multiplication in which case you need to return an empty list [ ].

Reference:

1. Python for everybody
2. docs.python.org
3. realpython.com