PCPS

UNIT-2

CONTENTS

- Data Structures
  (Lists, Tuples, Strings, Dictionary, Set)
- Files I/O System
- Functions

How is data stored?

- It is stored in a specialized form called data structures like lists, tuples, dictionaries etc.,

How do I access this data?

- For that there's 2 types.
  - → Sequence
    - Data can be accessed through index
    - We can use formulas to access them
  - → Non Sequence
    - Can't
    - can't

What are the operations we can do then?

- Store homogeneous AND non-homogeneous data as well.
- in & not in → To check if a particular thing is/is not there
- index operator [] → Operates by assigning a fixed value so it can be accessed
- +ve & -ve indexes → Sorting as it is or in reverse order
- Use relational operators → Comparing
- Slicing → Selecting a certain part
- len(), max(), min(), count(), index() → Standard functions with straight forward meaning

What's list?

→ can change

- Sort mutable data in [] square brackets

```
1 courses = ['Math','Physics','Chemistry','CS','ECE']
2 print(courses[0])
3 print(courses[1])
4 print(len(courses))
5 print('Math' in courses)
6 print('Math'==courses[0])
7 print(courses[-1])
8 print(courses[0:2:1])
```

+ve indexes
Length of list
in operation
relational operator
-ve indexes
Slicing
Start (Final+1) step
Printed only 0 & 1

→

```
Math
Physics
5
True
True
ECE
['Math', 'Physics']
```

- indexes start with 0. Not 1

What functions can be done on list?

```
courses = ['Math','Physics','Chemistry','CS','ECE']
courses.append(input('Enter a course: ')) ⟶ adds to the last
print(courses)
courses.sort() ⟶ sort based on alphabetical order
print(courses)
courses.extend(courses) ⟶ extended entire list again
print(courses)
courses.remove('Math') ⟶ Removed 1st Math
print(courses)
courses.pop() ⟶ Popped the last element
print(courses)
courses.insert(len(courses)-1,'Math') ⟶ Inserted element at specific index
print(courses)
courses.reverse() ⟶ Reversed list
print(courses)
courses.copy() ⟶ assign it to a variable, it will be assigned a copy of that list
print(courses)
courses.clear() ⟶ Cleared the entire list
print(courses)
```

```
Enter a course: EEE
['Math', 'Physics', 'Chemistry', 'CS', 'ECE', 'EEE']
['CS', 'Chemistry', 'ECE', 'EEE', 'Math', 'Physics']
['CS', 'Chemistry', 'ECE', 'EEE', 'Math', 'Physics', 'CS', 'Chemistry', 'ECE', 'EEE', 'Math', 'Physics']
['CS', 'Chemistry', 'ECE', 'EEE', 'Physics', 'CS', 'Chemistry', 'ECE', 'EEE', 'Math', 'Physics']
['CS', 'Chemistry', 'ECE', 'EEE', 'Physics', 'CS', 'Chemistry', 'ECE', 'EEE', 'Math']
['CS', 'Chemistry', 'ECE', 'EEE', 'Physics', 'CS', 'Chemistry', 'ECE', 'EEE', 'Math', 'Math']
['Math', 'Math', 'EEE', 'ECE', 'Chemistry', 'CS', 'Physics', 'EEE', 'ECE', 'Chemistry', 'CS']
['Math', 'Math', 'EEE', 'ECE', 'Chemistry', 'CS', 'Physics', 'EEE', 'ECE', 'Chemistry', 'CS']
[]
```

What are Tuples?

- Immutable data structures that can have 0 or more elements
  - ↳ Once set, can't be changed
- data sorted in ()

```
numbers=(1,2,3,4,5,(6,7,8))
for i in numbers:
    print(i)
print(numbers[1:6])
```

can be nested

```
1
2
3
4
5
(6, 7, 8)
(2, 3, 4, 5, (6, 7, 8))
```

can be sliced

iterable

```
numbers=(1,2,3,4,5,6,7,8)
print(len(numbers)) ⟶ length of tuple
print(sorted(numbers)) ⟶ sorting tuple based on alphabetical order
print(min(numbers)) ⟶ Min of Tuple
print(max(numbers)) ⟶ Max of Tuple
print(sum(numbers)) ⟶ Sum of All values in tuple
a='bring cake'
b=tuple(a) ⟶ converting sequence to a tuple
x=numbers +b ⟶ concatenating 2 tuples
print(x*2) ⟶ Repetition
print(2 in x) ⟶ Membership checking
print(x.count(3)) ⟶ Count of the items
print(x.index(4 )) ⟶ Index of specified item
```

```
8
[1, 2, 3, 4, 5, 6, 7, 8]
1
8
36
(1, 2, 3, 4, 5, 6, 7, 8, 'b', 'r', 'i', 'n', 'g', ' ', 'c', 'a', 'k', 'e', 1, 2, 3, 4, 5, 6, 7, 8, 'b', 'r', 'i', 'n', 'g', ' ', 'c', 'a', 'k', '
e')
True
1
3
```

What's Strings?

- Collection of characters / sequence of characters like

"" → Can have 0 character

"Can't See" (0 1 2 3 4 5 6 7 8)

"""" John Cena """" → Or more (0 1 2 3 4 5 6 7 8)

Technically all are same iykyk

- They are also immutable and slicing, indexing, len, min, max, concatenation, repetition can be used

What's Dictionary ??

- Data structure that organizes data into Key & value pair
- Its mutable, unordered and data can be any type
- It is ordered by a concept of hashing

How do I make a dictionary & what can I do with them?

```
1  marks = {'ram': 90, 'shyam': 80, 'mohan': 70, 'sohan': 60, 'raju':100}
2  print(len(marks)) length of dict
3  print(min(marks.keys())) min (in terms of keys)
4  print(max(marks.values())) max (in terms of values)
5  print(marks.keys()) print all keys
6  print(marks.values()) print all values
7  print(marks.get('ram')) print Value of specific key
8  print(marks.get('raj', 'not found')) print value of specific key, if not print 'not found'
9  print(marks.items()) print both keys & values
10 print(marks.keys()) print all keys
11 marks.pop('ram') remove specific key
12 print(marks)
13 secondmarks = marks.popitem() remove & select specific key & assign to another variable
14 print(secondmarks)
15 print(marks)
16 marks.update(dict(raj=90, ram=90)) #or you can use marks.update({'ram': 90, 'raj': 90}) → adding new values into dictionary
17 print(marks)
18 for key,value in marks.items(): → printing keys & values with for loop
19   print(key,value)
20 for key in marks:
21   print(key,dict[key]) → printing keys
22 name        = list(marks) → assigning variable with keys of dict ONLY
23 i=0
24 while i<len(name): → iterating with while loop
25   print(name[i],';',marks[name[i]]) → key & value of that key
26   i+=1
```

→

```
5
mohan
100
dict_keys(['ram', 'shyam', 'mohan', 'sohan', 'raju'])
dict_values([90, 80, 70, 60, 100])
90
not found
dict_items([('ram', 90), ('shyam', 80), ('mohan', 70), ('sohan', 60), ('raju', 100)])
dict_keys(['ram', 'shyam', 'mohan', 'sohan', 'raju'])
{'shyam': 80, 'mohan': 70, 'sohan': 60, 'raju': 100}
('raju', 100)
{'shyam': 80, 'mohan': 70, 'sohan': 60}
{'shyam': 80, 'mohan': 70, 'sohan': 60, 'raj': 90, 'ram': 90}
shyam 80
mohan 70
sohan 60
raj 90
ram 90
shyam dict['shyam']
mohan dict['mohan']
sohan dict['sohan']
raj dict['raj']
ram dict['ram']
shyam ; 80
mohan ; 70
sohan ; 60
raj ; 90
ram ; 90
```

Then what's a set?

- An unordered unique collection of unique elements in {}
- Mutable, Iterable, Non-indexable & comparing membership is fastest than others
- To make an empty set, x = set() .
- Also, set is not hashable but objects in set must be hashable

Why use sets tho? Theres alr so many!

- Sets remove duplicate elements & help compare 2 iterables for common/diff. elements

Explain this hashing thing pls....

- assigns some id & never changes for lifetime
- int, float, complex, bool, string, tuple, range, frozenset, bytes, decimal are hashable.
- List, dict, set & byte array aren't

So, a = {[1,2,3], 4, 5, {6:7, 8:9, 10:11}} ✗

b = {(1,2,3), 4, 5, 6.0, 7+8j} ✓

Okay, what can we do with sets then?

```
1  set1 = {1,23,4,5,6,7,55,69,5,7,1,3,23}
2  print(set1)
3  print(max(set1))
4  print(min(set1))
5  print(len(set1))
6  print(sum(set1))
7  print(sorted(set1)) → sort in ascending order
8  set1.pop() → pop last item
9  print(set1)
10 set1.remove(5) → remove that element
11 print(set1)
12 set1.discard(55) → remove that element
13 print(set1)
14 set1.add(100) → add element
15 set1.update([44,55,66,77,88,99]) → add an entire list of values, but not as list.
16 print(set1)
17 set2 = {10,3.44}
18 set1|=set2 #set1.union(set2) → union of 2 lists
19 print(set1)
20 print(1 in set1) → Verifying membership
```

max, min, len, sum

```
{1, 3, 4, 5, 6, 7, 69, 23, 55}
69
1
9
173
[1, 3, 4, 5, 6, 7, 23, 55, 69]
{3, 4, 5, 6, 7, 69, 23, 55}
{3, 4, 6, 7, 69, 23, 55}
{3, 4, 6, 7, 69, 23}
{66, 3, 4, 100, 6, 7, 69, 99, 44, 77, 23, 55, 88}
{66, 3, 4, 100, 6, 7, 69, 99, 10, 3.44, 44, 77, 23, 55, 88}
False
```

```
1  set1 = {1,2,3};set3={1,2,3}; set5={1,2,3}
2  set2 = {3,4,5};set4 = {3,4,5}; set6 = {3,4,5}
3  set1.intersection_update(set2) → Intersection
4  print(set1)
5  set3.difference_update(set4) → Difference
6  print(set3)
7  set5.symmetric_difference_update(set6) → Symmetric Difference
8  print(set5)
9  x={'a','b','c'}
10 y={'a','b','c','d','e','f'}
11 print(y.issubset(x)) → Check if subset or not
12 print(x.issubset(y))
13 print(x.issuperset(y)) → Check if superset or not
14 print(y.issuperset(x))
15 print(x.isdisjoint(y)) → Check if disjoint or not
16 x.clear() → Clearing set
```

```
{3}
{1, 2}
{1, 2, 4, 5}
False
True
False
True
False
```

What to do if I want more inputs?

- Files comes into picture
- It stays even when program terminates, larger data, less error

How do I use a file then?

- 3 Steps:
  1) Open
  2) Read | Write
  3) Close

```
open(file,mode='r',buffering=-1,encoding=None,errors=None,newline=None,closefd=True,opener=None)
```

Mode. Now choosen is read

Opening function

File location must be specified. But if the code & text file are in the same folder, not required

Doing this method is exactly the same but if not closed, can cause errors

```
with open('demo.txt','r') as f:
    #content

(OR)

f=open('demo.txt','r')
#content
f.close()
```

This is better beaz it automatically closes the file after the block is over

-
```
with open('demo.txt','r') as f:
    contents = f.read()
    print(contents)
```

Reads all the content in the file. But fine for small files. Larger ones will use lot of memory if done all together

```
1) hi
2) hello
3) hey there pretty
4) how are you?
5) can i get your number?
6) can i call you?
7) let's date!
8) this isn't working
9) lets break up
10) arjith singh op
```

-
```
with open('demo.txt','r') as f:
    contents = f.readlines()
    print(contents)
```

Solves our memory issue, but not desired output

```
['1) hi\n', '2) hello\n', '3) hey there pretty\n', '4) how are you?\n', '5) can i get your number?\n', '6) can i call you?\n', '7) let's date!\n", "8) this isn't working\n", '9) lets break up\n', '10) arjith singh op']
```

-
```
with open('demo.txt','r') as f:
    for line in f:
        print(line,end = '')
```

Thus this is better as it solves both problems

```
1) hi
2) hello
3) hey there pretty
4) how are you?
5) can i get your number?
6) can i call you?
7) let's date!
8) this isn't working
9) lets break up
10) arjith singh op
```

- for reading line by line

```
with open('demo.txt','r') as f:
    contents = f.readline()
    print(contents)
    contents = f.readline()
    print(contents)
    contents = f.readline()
    print(contents)
```

```
1) hi

2) hello

3) hey there pretty
```

- for reading specific no. of characters

```
with open('demo.txt','r') as f:
    size_to_read = 6
    contents = f.read(size_to_read)
    print(contents)

    print(f.tell())
```

```
1) hi
6
```

Reads that many no. of characters specified to read

Tells which character it is reading

```
with open('demo.txt','r') as f:
    size_to_read = 6
    contents = f.read(size_to_read)
    print(contents)

    f.seek(0)
    contents = f.read(size_to_read)
    print(contents)
```

```
1) hi
1) hi
```

Sets to read from that character

- For writing

```
with open('demo2.txt','w') as f:
    f.write('Test')
```

There was no file named 'demo2.txt' so, in that case it will create 1 automatically

If you don't mention from where, it will start from 0. So if there was prev data existing, all of it will be erased

Don't forget to use w and not r

- For appending. (Always from the end)

Use append.

```
with open('demo2.txt','a') as f:
    f.write('. Written from the end')
```

```
Test. Written from the end
```

Always writes from last Now.

Text files are fine, But data is in csv... How do I use them?

```
import csv
with open('demo.csv','r') as f:
    contents = csv.reader(f)
    for line in contents:
        print(line)
```

Importing csv helps to read data w/o some parameter that we mention

Reading all contents in csv file

```
['name', 'class', 'marks']
['will smith', '10', '75']
['obama', '4', '88']
['trump', '1', '34']
['snoop dogg', '8', '56']
['swift', '2', '69']
['pepe', '12', '100']
```

Pretty much all functions of read & write can be used

What are these functions ??

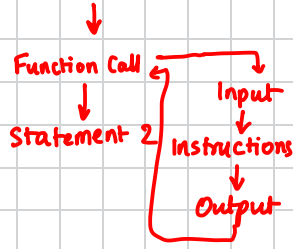
- Self contained block that performs specific task as it takes input, performs operations & returns output
- It helps in reuse of code & easier debugging cuz easy to read

How are they classified ?

- Predefined functions ↳ Standard library & User defined functions ↳ Defined by user

like print(), input(), len(), type(), sorted()

- () is called Function call operator




```
def area_rect(x,y):
    a=x*y
    return a
a= area_rect(5,6)
print('area= ',a)
```

```
area=  30
```

Defining the function name & the input it takes

Instructions

Output

Using that function

Note:

1) Defining multiple functions with same will lead to using only latest one
2) If a function has multiple return values, the function ends after 1st return & rest is unreachable / won't be considered
3) If there are multiple values being returned, they will be stored in a tuple & be returned
4) If there is no return, the output of that function is always None

1)

```
def area_rect(x,y):
    a=x*y
    return a
def area_rect(x,y):
    a=x+y
    return a

a= area_rect(5,6)
print('area= ',a)
```

```
area=  11
```

2)

```
def area_rect(x,y):
    a=x*y
    b= x+y
    return a
    return b

a= area_rect(5,6)
print('area= ',a)
```

```
area=  30
```

3)

```
def area_rect(x,y):
    a=x*y
    b= x+y
    return (a,b)

a= area_rect(5,6)
print('area= ',a)
```

```
area=  (30, 11)
```

4)

```
def area_rect(x,y):
    a=x*y
    b= x+y

a= area_rect(5,6)
print('area= ',a)
```

```
area=  None
```

Can I perform operations directly on arguments instead of a new variable every time?

- There are 2 Types of arguments
  1) Keyword arguments
  2) Positional arguments

Parameters names are used to pass arguments during call

→ Function(x=a, y=b, z=c, ...)

Arguments are in same order

Function (a,b,c ...)

```
def area_rect(x,y):
    a=x*y
    return a
a= area_rect(5,6)
print('area= ',a)
```

```
def minus(a,b):
    return a-b
x=20;y=15
print(minus(x,y))
```

Can we combine both positional & Keyword arguments ??

```
def a(x,y,z,a,b):
    print(a,b,x,y,z)
a(1,2,z={5,6},b=[4,5],a=99)
```

→ Position was switched but since we mentioned variable, it worked

If a variable is repeated, we get TypeError as it is assigning multiple values

Is it possible to give default value to a parameter?

```
def f(a,b=5):
    print(a,b)

print(f(4))
print('\n')
f(5,13)
```

```
4 5
None


5 13
```

Didn't mention about 2nd parameter, but default was assigned as 5, So it took b as 5

But it can take new values as well

How do I use a variable in the code & not a new one in the function?

- There are 2 types of variables
  - i) Global variables → Defined Outside function. Can be used in & out
  - ii) Local variables → Defined Inside function. Can be used only in

Note: Only read is possible. Can't change outside value inside function

```
x=10
print('global ->',x)
def f1():
    x=10
    print('local ->',x)
f1()
print('global ->',x)
```

```
global -> 10
local -> 10
global -> 10
```

Not affected by function

```
x=10
print('global ->',x)
def f1():
    global x
    print('local ->',x)
f1()
print('global ->',x)
```

```
global -> 10
local -> 10
global -> 10
```

Defined Global x.
Now we can perform functions on x
but whatever stays in vegas, stays in vegas (aka function) ;)

How to take input as a tuple or dictionary?

```
def f1(*arg):
    print(arg,type(arg))
    for i in arg:
        print(i)
f1(1,2,3,4)
f1()
f1(5,6)
```

Takes input only as tuple

```
(1, 2, 3, 4) <class 'tuple'>
1
2
3
4
() <class 'tuple'>
(5, 6) <class 'tuple'>
5
6
```

```
def f1(**arg):
    print(arg,type(arg))
    for i in arg:
        print(i,end ='')
    print()
    for i in arg.keys():
        print(i,end='')
    print()
    for i in arg.values():
        print(i,end ='')
f1(a=1,b=2,c=3)
```

Takes input only as Dictionary

```
{'a': 1, 'b': 2, 'c': 3} <class 'dict'>
abc
abc
123
```