omsairam   omsrivenkateswarayanamaha   ommamaheshwarayale   ommahaganadhipataye yanamaha

Python

Identity Operator:-

1. "in" → It check whether, both the objects or elements referring to same object or not.

 • Even though, if there are same but different (or have same content) references, then it'll return false.

Ex:
```
a = ["abc", "bcd"]
b = ["abc", "bcd"]
c = a
If (a is c) → True     | If (a==b) = True
If (a is b) → False    |
```

→

collection data types in Python:-

1. List → ordered + changeable

→ Add Items:-

 • at end of the list
  ⇒ list.append ("orange")

 • To insert at specific index:
  ⇒ list.insert(1, "orange")

→ Remove Items:-

 • remove() ⇒ list.remove("banana")
 • pop() ⇒ at specific index (or) at end
 • del ⇒ del this list[0].

- clear() ⇒ empties the list

```
list.clear();
```

→ list() constructor ⇒ list1 = list(("app", "ban", "che"))

Python Tuples:

- ( ) ⇒ round brackets
- Unchangeable

```
thistuple = ("abc", "efg", "jkl")
thistuple[1] = "xyz" → X (error)
```

Python Sets:-

- unordered & unindexed
- curly brackets { }

→ thisset = {"apple", "ban"}

- "CANNOT ACCESS THROUGH INDEX"

→ **Add Items:-**

- add() ⇒ for single item
- update() ⇒ update(["new1", "new2", "new3"])
  ↳ for multiple items

- Remove Item :-

- remove() ⇒ remove("abc") ← If "abc" is not present raises error
- discard() ⇒ discard("abc") → It doesn't raise error, even "abc" doesn't exist
- pop() → removes only last element as no indexing is present in sets, {}. & we don't know which element it'll remove

→ Dictionary :-

- Unordered, Changeable, Indexed.
- {} ⇒ Keys & values ⇒ {"1": "abc"}

- Looping :-

- for x in thisdict: ← keys
  print(x)
- for x in thisdict.values(): ← values
  print(x)
- for x, y in thisdict.items(): ← x: key, y: value
  print(x, y)

→ Removing Items :-

- pop() ⇒ thisdict.pop("model") ⇒ removes the item with the specified key name
- popitem() ⇒ last inserted item

dict() constructor:-

```
thisdict = dict(brand="Ford", model="1964")
```

SHORTHAND IF-ELSE:-

```
print("A") if a > b else print("=") if a == b else
    print("B").
```

FOR-LOOP

range(6) ⇒ 0 to 5

range(2,6) ⇒ 2 to 5

range(2,30,3) ⇒ 2 to 29 ; Increment the seq by '3'

ELSE in FOR LOOP:-

```
for x in range(6):
    print(x):
else:
    print("finished")
```

// print finish after completion of for loop

Python Lambda :-

- A lambda function is a small anonymous function.

- Can take any number of arguments, but can only have one expression.

- lamba arguments : expression

Example :-

1) x = lambda a : a+10
print (x(5)).

2) x = lambda a, b : a*b
print (x(5, 6))

2a)

why Lambda fn ?

The power of lambda is better shown when you use them as an anonymous function inside another function.

```
def myfun(n):
    return lambda a : a * n
```

↑

Use above function, to make a function that always doubles the number you send in:

```
mydoubler = myfun(2)
mytripler = myfun(3)
```

→ Now, you'll be same ~~boost~~ function with lambda expression for doubling & tripling.

```
print(mydoubler(11)) = 22
print(mytripler(11)) = 33.
```

- Common applications are 'Sorting & filtering' data'

Map, Reduce, Filter:

Map:-

It is useful when we need to call (or) apply a function on each element of the list.

For example compute the area of circle of diff radii

```
def area(r):
    return math.pi * (r**2)

radii = [2, 5, 7, 7.1, 0.3, 10]

x = map(area, radii)
```

(area → function; radii → list → data)

```
list_a = list(x)
```

→

Filter:-

- Used to select certain pieces of data from list, tuple and other collections
- Filters out data you don't need

→ filter (function_name , data)

→ filter (lambda x : (x > avg), data) → If True, then return

Reduce :-

- You have sequence of data, ~~and~~

Data : $[a_1, a_2, a_3 \cdots a_n]$ and a function, with two arguments,

Function : $f(x, y)$, It'll reduce the data into a single value.

reduce(f, data) :

Step 1 : $val_1 = f(a_1, a_2)$

$val_2 = f(val_1, a_3)$

$val_3 = f(val_2, a_4)$

⋮

return val_{n-1}

Python classes and Objects :-

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myFunc(self):
        print("Hello " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

→

In Python instead of function overloading we can make parameters, which we don't want to pass as "None" for a function.

Python JSON:

```
import json
```

JSON → PYTHON

```
x = '{"name": "John", "age": 30}'

y = json.loads(x)
print(y["name"])
```

PYTHON → JSON

```
x = {"name": "John", "age": 30}

y = json.dumps(x)

print(y)
```

PYTHON RegEx:-

- RegEx can be used to check if a string contains the specified search pattern

```
import re

txt = "The rain in Spain"

x = research("^The.*Spain$", txt)
```

Metacharacters for pattern matching:

1) [] - a set of characters → "[a-m]"

2) . - Any character (except new line) - "he..o"

3) ^ - starts with - "^hello"

4) $ - ends with - "hello$"

5) * - 0 or) more occurences - "ai*x*"

6) + - 1 or) more occurences - "ai+x+"

```
str = "The rain in Spain"
x = re.search("\d", str)
print(x.start());
```

→ Regular expression

→ string.

Python file handling:-

```
f = open("demofile.txt")
```

diff modes of opening a file:

"r" - read, "w" - write, "a" - append

"x" - create - Creates the specified file, returns an error if the file exists.

→ In addition, you can specify, how the file should be handled as "binary" (or) "text" mode.

"t" - text - Default mode

"b" - binary - Binary mode (eg. Images)

→

```
f = open("demo.txt", 'rt')
```

Read file:

```
f = open("abc.txt","r")
```

f.read() → whole line

f.read(5) → first '5' chars

f.readline() → read one line

→ Remove file:

```
import os

os.remove(" ")
```

Python MySQL

```
import mysql.connector

mydb = mysql.connector.connect(
        host = "localhost"
        user = "your_user_name"
        passwd = "your_password"
        database = "my_database"
    )

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE abc
        (name VARCHAR(255), address VARCHAR(255)
        )")
```

→

LIST COMPREHENSION → ~~AFTER KINUX~~

ON OTHER SIDE OF NOTES

List Comprehensions

- List comprehension is surrounded by brackets ([]), similar to a normal list but has a expression, instead of elements.
  followed by for-loops & if-clauses

- [expr for val in collection]
- [expr for val in collection if <test>]
- [expr for val in collection if <test1> and <test2>]
- [expr for val1 in collection1 and val2 in collection2].

1) List of squares of first 100 +ve integers:

squares2 = [i**2 for i in range(1, 101)]

Remainders of when first 100 numbers are divided by '5'

```
remainders = [(x**2)%5 for x in range(1,101)]
```

→ Movies start with letter "G" :-

```
gmovies = [title for title in movies if
                    title.startswith("G")]
```

→

```
movies = tuple
       = [('Gandhi', 2005), ('The Aviator', 2004),
                                        ...]
```

Task: Find movies released before 2000

```
pre2kmovies = [(a,b) for (a,b) in movies if b<2000]
```

Scalar Multiplication:

$V = [2, -3, 1]$

$4 * V \Rightarrow [2, -3, 1, 2, -3, 1, 2, -3, 1, 2, -3, 1]$

In python, if we multiply lists, it concatenates

$a = [1, 2] + [3, 4]$

$\Rightarrow a = [1, 2, 3, 4]$

So, we can use list compre to multiply lists

$V = [4 * i \text{ for } i \text{ in } V]$

CARTESIAN PRODUCT:

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

$C = [(a, b) \text{ for } a \text{ in } A \text{ for } b \text{ in } B]$

resource manager

memory management:

list

range & xrange

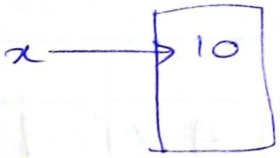
multi thread()

HCI

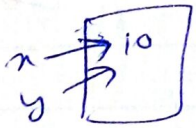
Memory management in Python:

- Everything is object in python

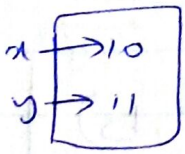
1) $x = 10$



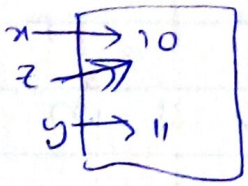
2) $y = x$



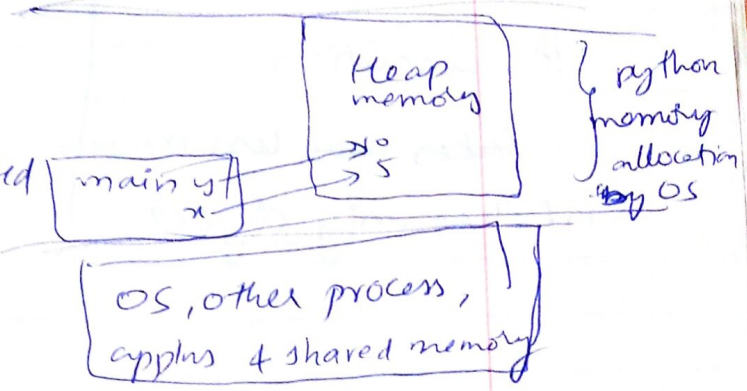
3) $y = y + 1$



4) $z = 10$



→ python optimizes memory by assigning the same object, if object already exists.



- whatever memory allocated to python will be further divided into heap memory & stack memory.
- In heap memory, objects are created
- In stack memory, you have function calls, variables and others.

- In python "Reference counting" algorithm is used for garbage collection
- Java uses "Mark & Sweep" algorithm

Difference b/w xrange & range in

xrange	range
→ returns Generator object	→ returns list
→ we need loop to print values	→ as it returns list no need of
→ xrange in python-2 changed to range in Python 3	→ present in py 2 & rem in py3

as it evaluates(lazily) by generating we request for a value

- Takes more time for execution
- Takes less memory as it has only object

everytime

- Takes less time for execution
- Takes more memory as it contains entire list.

---

xRange in py2 → Range in py3

- Closures – Please refer

www.geeksforgeeks.org/python-closures/