Python x Syntax - Cheatsheet

Variables

a = "Ram"

- to print data type of variable : print(type(a))
  ↓↓
  <class 'str'>

- i++ : → invalid here
  use : i += 1

- boolean : True, False

- typecasting : a = "3534" → string
  print(type(a)) → <class 'str'>
  a = int(a)
  print(type(a)) → <class 'int'>

- to take input : a = input(" --- ")
  print(a)
  print(type(a)) → <class 'str'>

  ↓
  it bydefaut takes string as datatype while taking input from the user

Strings

- a = "Aryan" or a = 'Aryan' or a = '''Aryan'''

  print(a)

- string concatation : s = "Hi" , u = "Bye"

  print(s + u)

  print("name", s)

- index starts from '0' (zero)
- slicing of string : • print(name[0:3])

  (0, 1, 2) → index wala string character gets printed.

  - [:4] → ≈ [0:4]
  - [2:] → ≈ [2: last length]
  - [0::2] → skips 1 character
  - [0::3] → skips 2 character
  - [-4:-1] ≈ [1:4]

  - len(st) → length
  - st.endswith("...")
  - st.count(" ")
  - st.capitalize() → only 1 letter of sentence
  - st.find()
  - st.replace("}", " ")

Lists

- a = [1, 2, 78, True, "Ram", 58.25]
  print(a)
- index start from 0 (zero)
- Can change the values in the list
- list slicing similar to that of string
  - l1.sort() → ascending me sort
  - l1.reverse() → descending me sort
  - l1.append(2) → add at the end of list
  - l1.insert(pos, elem) → insert
    (pos ← index, elem ← value)
  - l1.pop(pos) → delete element at position = pos
  - l1.remove(21) → removes 21 from list

Tuples

- t = (1, 2, 4, 5)
- t1 = () : empty tuple
- t2 = (2,) : tuple with 1 element
- index stat from 0
- we can't update tuple
- t.count(4) → no of occurance of 4
- t.index(2) → 1st index of any value

Dictionary

```
myDict = { "key" : "values" ,
           "    " : "      " ,
           "another" : { "  " : "    " ,
                         "  " : "    " }
         }
```

- print(myDict)
- can update changes in dictionary → myDict.update(":")
- print(myDict['key']) ⇒ values
- print(myDict.keys()) ⇒ print keys of dictionary
- print(myDict.values()) → values of dictionary
- print(myDict.items()) → (key, values) → print
- print(myDict.get('key')) → gives error if key not present, print values associated with it
- print(myDict['key']) → given 'none' if key not present

Sets

- a = {1, 3, 5, 8}
- b = { } → empty dictionary
- c = set()
  print(c) → empty set
- non-repetative
- can't update
- a.add(8) → add element to set

- a.add((4,8,3)) → adding tuple to set
- Only tuple can be added to the set
- len(a) → length of set
- a.remove(6) → to remove 6 from set
- a.pop() → to remove any random elemt from set
- a.clear() → to clear all elemt present in set

Conditionals

(1) if-else statement

```
a=45
if(a>50):
    print("Yes")
else:
    print("NO")
```

(2) if else ladder

```
a=45
if(a<18):
    print("A1")
elif(a>18 and a<50):
    print("A2")
elif(a>50 and a<60)
    print("A3")
else:
    print("A4")
```

(3) Logical Operators :

and , or , not

(4) is → operator :

```
a=None
if(a is None):
    print("Yes")
else:
    print("NO")
```

- it compares the value

(5) in → operator

- checks whether particular is present or not
- a=[2,6,8,10]
- print(8 in a) → true
- print(16 in a) → false

Loops

(1) while

```
i = 0
while (i <= 10):
    print ("Yes!")
    i += 1
```

- To print list

```
f = [6, 6, 22, 83, 99]
i = 0
while i < len(f):
    print(f[i])
    i = i + 1
```

(2) for

- fruits = [' ', ' ', ' ']

```
for item in fruits:
    print (item)
```

- range (start, stop, step)

  start = 0 ⇒ by default

  for i in range (1, 10)

  ↓

  i = (1 to 9)

```
for i in range (11):
    print (i)
else:
    print ("Done!")
```

→ this else executes only after when the loop exhausts naturally.

- pass → to do nothing

Functions

(1)

```
def functionname (value):
    return xyz
```

(2) default argument

```
def greet (name = "Stranger")
    print ("Hm" + name)
```

File Handling

```
//open file                          → reading mode
f = open('file.txt', 'r')
data = f.read()
print(data)
f.close()
```

→ 'r' is also taken as default

- data=f.read(10) → read 1st 10 character of the File
- data = f.readline() → read 1st line of the file

```
f = open('file.txt', 'r')
data = f.readline()
print(data)          → 1st line prints
data = f.readline()
print(data)          → 2nd line prints
f.close()
```

- 'r' → reading
  'w' → writing.
  'a' → appending

- 'rb' → binary files
- 'rt' or 'r' → text files.

// write file

- f = open ('file.txt', 'w')
  f.write(" . . . . ") → it will erase prev content & write this in file
  f.close()

- f = open ('file.txt', 'a')
  f.write(" . . . . ") → it will add this at the end of content of file
  f.close()

// with file → no need to close ur file here

```
with open('file.txt', 'r') as f:
    a = f.read()
    print(a)

with open('file.txt', 'w') as f:
    a = f.write(" . . . ")
```

** file content are in string data type

- import os

os.remove(filename) → remove/delete this file

Creating Class

```
class Railway :
    form = "Railway Forum"

    def printDetails(self) :          → self parameter
        print(f"Name  : {self.name}")
        print(f"Train : {self.tr}")

rk = Railway()  # object creation
rk.name = "Rahul"
rk.tr = "Rajdhani Exp."

rk.printDetails()
```

Attributes

```
class Employee :
    company = "Google"   # > class attributes
    salary = 200

rahul = Employee()  # object instantiation
esha = Employee()
print(esha.company)  → Google
rahul.salary = 180   # > instance attribute
esha.salary = 220    #

Employee.company = "Tesla"  → changing class attribute for both
```

- Instance attribute preffered over class attributes
- if no instance attribute, then it will print class attributes

print(esha.company) → Tesla

Static-Method

- self parameter is automatically passed with a fn call from an object

```
class Emp:
    company = "Google"
    def getsalary(self):
        print("salary is 20K")

rk = Emp()
rk.getsalary()  ≈  (Emp.getsalary(rk))
```

- if self not → then throw-error
- Not Using self parameter

```
@staticmethod
def greet():
    print("Good byee ... tata ... !!")

rk.greet()
```

Constructor

```
class Employee:                                  # class
    company = "Google"                           # class attribute

    def __init__(self, name, salary):            # Constructor
        self.name = name
        self.sal = salary

    @staticmethod                                # static method
    def greet():
        print("Bye ... Bye .... Esha")

    def getDetails(self):                        # method
        print(f"Name   : {self.name}")
        print(f"Salary : {self.sal}")

rahul = Employee("Rahul", 1500)
ram = Employee("Ram", 1200)

rahul.greet()
rahul.getDetails()

ram.greet()
ram.getDetails()
```

Inheritance

```
class Employee :          # Parent
    company = "Adobe"

    def printD(self):
        print("Employee class ...")

class Programmer(Employee) :    # Child
    language = "Python"

    def printd(self):
        print("Programmer ... class ....")

e = Employee()
p = Programmer()

print(p.company) ✓
print(e.language) X
```

- if any detail present in both then child class data gets printed otherwise it will take from its parent class

(1) Single Inheritance

```
class Employee :
    ==
    ==
    =

class Programmer(Employee):
    ==
    ==
```

(2) Multiple Inheritance

```
class Employee :
    ===
    ===
    ===

Class Freelancer :
    ===
    ===
    ===

Class Programmer (Employee, Freelancer) :
    ===
    ===
    ===
```

I II

we can change this order as per our choice

Programmer will inherits 1st from Employe(I) then Freelancer (II)

- it will inherit 1st from written 1st in the ( )

(3) Multilevel Inheritance

```
class Person :
    ===
    ===
    ===
    ===

class Employee (Person) :
    ===
    ===
    ===
    ===

Class Programmer (Employee) :
    ===
    ===
    ===
    ===
```

- if it contains its own methods then they will run theirs respectively

- If any attributes with same name present in Employee & Person class then it will 1st inherits

* Person ⇒ Employee ⇒ Programmer

Super method

- used to access the methods of a super class in a derived class

```
class Person:
    def takeB(self):
        print("Hi...")

class Employee(Person):
    def takeB(self):
        super().takeB()
        print("Hellow")

p = Person()
e = Employee()

p.takeB()  ⟶ Hi
e.takeB()  ⟶ Hi
             Hellow
```

- Just use "super().fn name"
- if u want super for constructor :

super().__init__()

Class Methods

* Changing class attributes *

(1)

```
class Employee :
    salary = 100

    def sal(self, s):
        self.__class__.salary = s

e = Employee()

e.sal(500)
print(e.salary)
print(Employee.salary)
```

→ change salary over class

→ 500

(2)

```
class Employee :
    salary = 100

    @classmethod
    def sal(self, s):
        cls.salary = s
```

```
e = Employee()
e.sal(600)
print(e.salary)
print(Employee.salary)
```

→ 600

Property decorator

```
class Employe:
    salary = 5600
    bonos = 500

#getter
@property
def totalSalary(self):
    return self.salary + self.bonos

#setter
@totalSalary.setter
def totalSalary(self, val):
    self.bonus = val - self.salary

e = Employee()
e.totalSalary = 6800

print("New salary ", e.salary)
print("Bonus ", e.bonus)
print("Total ", e.totalSalary)
```

5600
1200
6800

Operator Overloading – dunder method

```
Class No :
    def __init__(self, num) :
        self.num = num

    def __add__(self, num2) :        → (+)
        return self.num + num2.num
```

__sub__ → (-)

__mul__ → (*)

__str__ → print statement

Try - Except

```
try :
    ===
except Exception as e :
    ===
```

Handling Exception - Specific

```
try :
    ===
except ValueError as e :
    ===
```

except ZeroDivisionError as e :

except Exception as e :

Raising Exception

try :

except :
    raise ValueError("Any Message")

try - else

try :

except Exception as e :

else : } -> else executed only when try is true (try executed succesfully)

try - finally

```
try :
    - - - -
except Exception as e :
    - - - - -
finally :
    - - - - -
```
} executed irrespective of any

global - keyword

→ modify variable outside of current scope

(1)

```
a = 54  # global variable
def fun1():
    a = 8  # local variable
    print(a)
```

fun1() → 8
print(a) → 54

(2)

```
a = 54  # global variable
def fun1():
    global a  # changed into global
    print(a) → 54
    a = 8
    print(a) → 8
```

fun1() → 8
print(a) → 8

Enumerate

```
l1 = [3, 52, False, 6.54]
for index, item in enumerate(l1):
    print(item + " --> " + index)
```

3 → 0
52 → 1
False → 2
6.54 → 3

List Comprehension

```
a = [2, 8, 16, 77, 89, 209, 208]
b = [i for i in a if i % 2 == 0]
print(b)
```

⇓ [2, 8, 16, 208]

Main - fn

```
print(__name__) → __main__
```

Virtual Environment

- to create : terminal ⇒ virtual env nameenv
- to activate : ⇒ ./myproject/Scripts/activate.ps1

Lambda fn

```
fun = lambda a : a+5

print(fun(2))
```

⇓

10

⇒ lambda argument : expression

lambda input : return

Join Method → creats string from iterable object (list, tupple, sets)

```
list = ["C", "A", "B", "D"]

Sentence = "-->".join(list)

print(sentence)
```

C → A → B → D

Format Method

```
a = "Hi, bro this is {} and {}".format
        (name1, name2)

name1 = "Rahul"
name2 = "Aryan"

print(a) ⇒ Hi, bro this is Rahul and Aryan
```

Map Method → fn to all items in input list

Syntax : map(fn, input)

```
l1 = [1, 2, 3, 4]

def Sq(num):
    return num * num

print(list(map(Sq, l1)))
```

⇓

[1, 4, 9, 16]

Filter method

Syntax : list(filter(fun, list))

```
la = lambda num : num > 5
print(list(filter(greatr_than_5, lt)))
lt = [2, 6, 8, 10]
```

⇓

[6, 8, 10]

Reduce Method

```
from functools import reduce
sum = lambda a, b : a + b
l1 = [1, 2, 3, 4, 8]
```

val = reduce (sum, l1)

print (val) → add in A.P

1 + 2 = 3 ,

3 + 3 = 6 , 6 + 4 = 10 , 8 + 10 = 18