x = (Hello, '4', '5') → Hello not def → Error

24/05/2025

Sets

Set: ~~unorded~~ ① unordered collection of items.
② doesnot have duplicate values.

Ex: Colors = {'red', 'blue', 'black'}

⊛ Checking value if it is present using 'in' keyword.

characteristics
→ Sets are a type of collection like lists, tuples storing mixed data
→ enclosed within curly brackets with comma seperated
→ Sets are unordered
→ Does not allow duplicates.

⊛ Declaring a set:
set1 = {1, 2, 3, 4}

~~Set~~

⊛ Adding element to set:
set1. add ("India")
Set1 = {1, 2, 3, 4, "India"}

⊛ To remove you simply do,
Set1. remove ("India")

→ if i want to count the unique elements then it is going to be converting the list into set.

Ex: A = [1, 1, 2, 4, 3, 3, 2]
A = set(A)
print(A) = {1, 2, 3, 4}

Set Operations

A = {0, 2, 4, 6, 8}
B = {1, 2, 3, 4, 5}

Union operation: OR operation is called Union operation (|)
Intersection operation: AND operation is called Intersection operation.

Union: can be done by (A|B) or (A.union(B))

Intersection: print(A & B) (or) (A.intersection(B))

Difference: print(A-B) or print (A.difference(B))

Symmetric difference: Removes all the common elements.
Print (A^B)

Dictionary: Unordered collection of Data.
→ Data in dictionary is stored as a key: value pair
→ key shoud not be mutable and value can be of any type.

Dict = {"name": "John", 'age': 10}

→ key is like an index, its always unique and immutable
→ values are objects that contain information,
→ values are accessed using their keys.
→ Each key is followed by a value seperated by colon
→ values can be immutable, mutable & duplicates

* **Declaring a Dictionary**:

d1 = {"India": INR, "USA": USD, "Hong Kong": "HKD"}

* Accessing value using keys

d1 ["India"]

"INR"

* Replacing value for a key in dictionary

d1["Hong kong"] = "HK$"

d1 = {"India": 'INR', "USA": 'USD', "Hong kong": "HK$"}

⊛ Inserting a New key-value pair

d1["Japan"] = YEN"

→ d1 = {"India": INR, "USA": USD, "Hong Kong": HK$, 'Japan': 'YEN'}.

⊛ Deleting a key value pair.

→ del d1["Japan"]

⊛ Sorting a dictionary

sorted(d1)

⇒ ['Hong Kong', 'India', 'USA']

⊛ values() method

d1.values() ⇒ ['INR', 'USD', 'HK$']

d1.keys() ⇒ ['India', 'USA', 'HongKong']

⊛ get() method → Returns the value ⇒ d1.get('USA')

⊛ update() method → d1.update({"India": "Rupee"})

d1["India"] = "Rs"

__Python__ __Loops__, __Functions__ __and File handling__

__Loops__: Used for repetition - iterating over a iterable.

→ __iterable__: object that can be iterated over.

Eg: Lists, Tuples, Dictionaries, strings, sets.

→ iterator: variable that goes through each element in the iterable

→ __2 Types__ of __loops__: For loops & while loops.

⊛ iterating through a list

```
list1 = [1, 24, 6]
for i in list1:
    print(i)
```

o/p:
```
1
24
6
```

⊛ iterating through string

```
str = "Krishna"
for i in str:
    print(i)
```

o/p:
```
K
r
i
s
h
n
a
```

⊛ iterate over a dictionary. (Keys)

i/p: Student-data = {1: ["KK", 25], 2: ["preethi", 23]}
    for i in student-data:
        print(i)

o/p: 1
     2

→ But to print the records or the values in a dictionary, we can use items() function. (Keys, values)

i/p: Student_data = {1: ["KK", 25], 2: ["preethi", 23]}
    for i in student_data.items():
        print(i)

o/p: (1, ["KK", 25])
     (2, ['preethi', 23])

→ if i just need values i can simply use a new iterator "j" and print it.

i/p: for i, j in student_data.items():
        print(j)

o/p: ['KK', 25]
     ['preethi', 23]

⊛ Comprehensions

list1 = ["KK", "Kanth", "Mark", "preethi"]
list2 = [len(i) for i in list1]

→ As same as
list2 = [ ]
for i in list1:
    list2.append(len(i))

o/p ⇒ [2, 5, 4, 7]

⇒ create a dictionary.

d1 = {i: len(i) for i in list1}

o/p = {'KK': 2, 'Kanth': 5, 'Mark': 4, 'Preethi': 7}

→ As same as.
d1 = { }
for i in ~~listofs~~ list1:
    d1[i] = len(i)

functions

→ Named sequence of statements that performs some operations.

→ User defined functions are created using a keyword called def.

```
def function name ( arg1, arg 2, ... ):
        obj = arg1 + arg 2
        return obj
function name ( val1, val2 ) → calling it.
```

(*)

```
def func_1 (name, age)
func_1 ("KK", 25)
```

(*) default value

```
def func_1 (name, a=30)
func_1 ('KK')
```

→ Lambda function: small anonymous function to make developers life easier.

→ can take any no. of arguments, but only one expression.

Syntax: lambda argument : Expression

Example

```
x = lambda  a : a+10
print(x (5))
```

o/p: 15

Example: print even (or) odd

```
x = lambda a: 'even' if a%2==0
                    else 'odd'
print(x (100))
```

Example: Adding two numbers

```
c = lambda  a,b : a+b
print (c(4,5))
```

o/p: 9