Dictionary Data Structure

Ex: lst = [10, 20, 30]

idx → 0 → 10, 1 → 20, 2 → 30 automaticly

Print (lst[1]) → 20

dict = {0:10, 1:28, 2:55, 132:37}

Print (dict[2]) → 55

Print (dict[132]) → 37 → manual assign for keys to the dictionary

Flexible Keys → Return Data By any Key Where Key Can be any Immutatable Value.

dict = {

'a' = 'alpha'

'o' = 'omego'

'g' = 'gamma'

}

keys

'a' → 'alpha'

'o' → 'omega'

'g' → 'gamma'

Print (dict) → Return it

Print (dict.keys()) → Return dict.keys(['a', 'o', 'g'])

Print (dict.values()) → ['alpha', 'omega', 'gamma']

Print (dict['a']) ⇒ alpha

immutables like → Float, int, tuple, String
↳ Be careful from Float cause it's an approximate don't use it except in High Need situations

Tuble contain list Can't Be akey. We can use another Dictionary as key

Dictionary is mutable (we can update it's value)

⇒ dict[12] = [405, (1, 'mostafa')]
dict['mostafa'] = 20 → value of it is [12]

⇒ dict[12] = Hello → Re assign to key 12
↳ value of key 12

⇒ del dict[12] → To delete a key
Print(dict.keys()) = ('mostafa')
↳ only

⇒ dict[12] = 10, dict[12] += 5 → So dict[12] = 15

⇒ Print(dict[12]) = 15, Print(dict.pop()) → Return and
Print(dict.pop('hey', 37)) Remove
↳ Print(37 if hey doesn't exist (Providing a value))

⇒ Print(dict) → Return it's a class Name
↳ We should use different Name

Indexing dict values :-

```
dct = {
        'Ahmed' : 'diaa' ,
        1 : [1, 5, 7, 9] ,
        3 : [[5, 7] , [8, 9, 10]]
      }
```

Print (Dict['Ahmed'][1]) → A

Print (Dict[1][1]) → 1

Print (Dict[3][1][2] → (10)

→ We can use Data type as key Because it's immutable

```
dct = {
        int : [6, 9, 10] ,
        float : 10 ,
        b : 20 ,
        6 : 80 ,
      }
```

Print (dict[float])

→ Print (dict[b])

80 → last value

→ Print (dict.Setdefult [b, -8])

if b is Not a key → Make a key and Set it with Value (-8)

```
dct = {
        -120000 : 'Ahmed' ,
        'Ahmed' : 25.5 ,
        (4, 6) : [5, 8, 9]
      }
```

⇒ Ex: Print (dct.Setdefult (7, 20))

(7, 20) → in dct

Print ('Ziad' in dct) → True

Print (100 in dct) → False.

↳ Check key only.

If 7 in dct and dct[7] == 5! → Check if a value in dct
Pass

```
dct = {
    -1200001 : 'Ahmed',
    'Ziad' : 25.5,
    (4,6) : [5,8,9]
    }
```

→ Print(dct.get(7)) → it will Return None
→ Print(dct.get(7,15)) → if 7 doesn't exist it will Set value to 7:15
→ Print(7 in dct:) → False
→ Print(dict.get((4,6))) → [5,8,9]

dict.clear() → Remove All Keys.

PopItem() :- This Function Return (key, value) of last item in dictionary
↳ Returned As tuple

```
dct = {'x' : 11 , 'b' : 20 , 'y' : 30}
dict['a'] = 30
```
↗ Inserted at the end.

```
while dct :
    Print(dict.PopItem())
```

('a', 30)
('y', 30)
('b', 20)
('x', 11)

↳ Result of PopItem()

Time Complexity :

	Avg	Worst
K → d	O(1)	O(n)

Dict 3 :-

Insertion order : Now Preserved

dict = {}

dct[20] = 10 , dct['Ahmed'] = 20 , dct[30] = 15, dct[(9,7)] = 150

dct[30] = 10

dct = {
20 : 10 ,
'Ahmed' : 20 ,
30 : 10
(9,7) : 150
}

---

dct = {'x': 11, 'b': 22, 'y': 30}

Print(dct.items())
↳ dct_items([('x', 11), ('b', 22), ('y', 30)])

For key, value in dct.items():
Print(key, value)
↳ x 11 b 22 y 30

⇒ Print(dct.keys())
↳ dct_keys(['x', 'b', 'y'])

Print(list(dct.keys())) ⟷ For key in dct.keys():
↳ ['x', 'y', 'b'] Print(key, dct[key])
↳ same as key, value but slower.

→ For key in sorted(list(dct.keys())):
Print(key, Dct[key]) ⟶ (b x y)
↕ (they are both same)

→ For key in sorted(dct):
Print(key, dct[key]) ⟶ b x y

List Vs Dict:

List: Ordered Sequance
Can Be indexed or Sliced

Dict: Bunch of Keys : Values
Insertion Order is Preserved

Shallow Copies:

```
Class Employee:
    def __init__(self):
        self.id = 10
    def __repr__(self):
        Return str(self.id)

emp = Employee()
lst = [5, 8, 9]
dct = {"Ziad": 25.2, 2: lst, 'Hey': emp}
Print(dct) → {"Ziad": 25.2, 2: [5, 8, 9], 'Hey': 10}
lst.pop() → only methods effect lsts as values
emp.id t=100  In adictionary.
Print(dct) → {"Ziad": 25.2, 2: [5, 8], 'Hey': 110}
lst[5] → This Doesn't effect the dictionary → dct[2] = (lst get effected) ← [5]
d2 = dict.copy()
Print(d2['Hey'] is emp) → it will Return True Because Shallo copy Return → Simple Version of the orginal (dct)
```

Functions of Dictionary: Merge, len, all, any

```
dct = {'x': 11, 'b': 22, 'y': 30}

dct['a'] = 33
```

→ you can pass dict or a list of tuples

```
dct.update({'aaa': 3, 'b': -2})   → Merge it with dct
print(dct)  → {'x': 11, 'b': 22, 'y': 30, 'aaa': 3, 'b': -2}

print(len(dct))  → Return 7

print(all(dct))  → True if all keys → True
dct[''] = hey   → empty variable, 0, False → will give False in dictionary

print(all(dct))  → False
any  → True
```

Dict Comperhension:-

Without:

```
squares = {}
for x in range(6):
    squares[x] = x*x
print(squares)
```

With

```
square = {x: x*x for x in range(6)}
print(square)
```

{0:0, 1:1, 2:4, 3:9, ...}

Constractor:-

```
a = {'one': 1, 'Two': 2, 'three': 3}
b = dict([('two', 2), ('one', 1), ('three', 3)])
```

List of tubles

```
c = dict(one=1, two=2, three=3)
```

→ keyword

Fromdict → keyword ←

```
d = dict({'one': 1, 'three': 3}, two=2)
E = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
```

(one, 1) (Two, 2) (three, 3)

a = [1, 2, 20, 6, 210, 2, 1]
d = dict.fromkeys(a)
⇒ {1: None, 2: None, 20: None, 6: None, 210: None}

⇒ We can add 7 → fromkeys(a, 7) → instead of None it will be default value

unique_keys = dict.fromkeys(a).keys()
print(unique_keys) → dict_keys([1, 2, 20, 6, 210])
⇒ No duplicates, preserved order.

unique_keys = list({10:2, 1:5}) → [10, 1]
↳ list keys only.

P1: Read string of lower/upper letters
Convert all of them to lower → Compute frequency

```
String = input("Enter String")
String = String.lower()
dct = dict.fromkeys(string) → {b:, a:, 2:}
for i in string:
    for grey, value in dct.items():
        if key == i:
            value += 1
```

To make it increase value of every key when a part of string is founded.

{b:, a:, 2:} → fromkeys(string, 0)

```
for key in dct:
    for i in string:
        if key == i: Dct[key] += 1
print(Dct)
```

P23 Find most frequent number :

Read line of N integers ⟶ Find most Repeated Numbers (small ⟶ large)

N = int(input()) ⟶ No. of inputs

```
For i in Range(N):
    Num = int
```

```
List(map(int, input()))
X = input("Enter your nums:")
.split()
print("inputs : {X}")
n=3, (1, 2, 2)
```

```
dct = dict.fromkeys(X, 0) → {1: , 2: }
Counter = 0
For keys in dct:
    if keys == X[i]:          # 1 == X[0]      ## keys
        dct[key] += 1,        # dct[key] + 1
                                  dct[1]      ## [1:1, 2:1]
    if i < N:                 ## i<N → i=2 → X=[1,2,2]
        i += 1                → i=1          X[2] = 2
    else:                                    key 2
        Break                                Value+1
```

⇒ Try: `for num in X:` ⟶ Loop over [1, 1, 1, 1]

```
    dct[num] += 1
print(dct)
```

dct[1] → +1
dct[1] → +1
dct[2] → +1
dct[1] → +1 } ×4

To Print Two greatest values:

```
dct = dct.sort()
output = dct.pop(), dct.pop()
```

To get highest frequency: -mx = max(dct.values())

freq = [key for key, value in dct.items() if value == mx.]

↳ Return most Repeated no.

Practise 3: Search for a number.

lst = list(map(int, input().split()))

queries = list(map(int, input().split()))

dct = {}

For idx, value in lst:

dct[value] = idx → Set every value with numbers

For q in queries:

ans = dct.get(q, -1) → Return value of key(q) if not (-1).

Print(f"Queries {q} answer {ans}") → Print style.

`input = list(input()) x` X

`input = input()`

`if input is str:` X

`String = input("Enter String")`

`map = input("Enter map")`

`Dct = {}`

`Dct = dict.fromkeys(string, " ")`

for X

Problem #1: Special String Mapping:

```
From_Str = [a - z]
to_Str = [Data]

dct = { from_str[idx] : to_str[idx] for idx in Range(len(from_str)) }
                                    (assign F_Str : to_Str)

String = input("Enter String") → enter a String.
Res = " " → variable to store the output Value
For Char in String: → for every char in String
    if Char in dct: → Char is a key in dct
        Var = dct[Char] → Var is equal to the value of key in dct
        Res += Var → Add it to Variable
                    ↳ To Print it.
```

Problem #2 : Sort by Type:

⇒ implement function → def Sort_different_types(lst)
it takes (int, float, String, list, tuple)

every int come first stay first.

```
def Sort_different_type(lst):
    lst = List(input("Enter different Data Type").split())
    S_lst=[] for item in lst:
        if len(S_lst) = 0:
            S_lst.append(item)
        elif item is list:
            S_lst.append(item)
        elif item is float:
            if S_lst[-1] is Not list:
                S_lst.append(item)
            else:
                for idx, Value in enumerate(S_list):
                    if Value is float:
                        (S_lst.insert(idx, item)
        elif item is string:
            if S_lst[-1] is Not String:
                S_lst.append(item):
            else:
                for idx, Value in enumerate(S_lst):
                    if Value is string:
                        S_lst.insert(idx, item)
        else:
```

if len(S_lst) or item is list:
    S_lst.append(item)

Solution:

```
def sort_by_dataType(lst):
    dct = {}
    for item in lst:
        t = type(item)
        dct.setdefult(t, [])  → Create a key of Data types and lst as value
        dct[t].append(item)   → Add item to it's Type list

    lsts = [sorted(lst) for lst in dct.values()] → Sort every lst
    Return [item for lst in lsts for item in lst] → get every item in every lst sorted

if __name__ == '__main__':
```

First Prefix Finder :-

⇒ Read an input as Number of inputs in DB, enter the DB

⇒ Read a number which is Number_of inputs, then enter string:

⇒ For every query list all DB strings that start with this query.

```
if __name__ == '__main__':

    n = int(input("Enter number of string"))
    for i in range(n):
DB = input("Enter Data").split()
    dct = {}
    for idx, value in enumerate(DB):
        dct[idx] = value
```

```
num = int(input())
for i in range(num):
    s = input()
    output = []
    for key in range(n):
        if s in dct[key]:
            output.append(dct[key])
    if len(output) == 0:
        print("No items founded")
    else:
        print("items founded : {output}")
```

Assum: d = A

```
dct = {0: 'Ahmed',
       1: 'Diaa',
       2: 'Eldeen'
       }
```

---

def filter_dublicates_preserve_order(lst):

→ input (lst of integers)

→ you must preserve order

→ Use a dict →

def filter_dublicates_preserve_order(lsts):

dct = {} → create dictionary

dct = {}

Turn lst ← Tubles = [tuple(lst) for lst in lsts]

into immutable → Tuble

(1,1): None

(2,4): None

same keys ← dct = dict.fromkeys(Tubles)

of the Tubles) Return [list(tup) for tup in dct]

↳ (Return every key which is Tuble as a list)

Set Data Structure

- Set: Unordered
  - Don't Preserve Insertion order.
- Unique: "Dublicates" are Ignored
- items must be immutable.

`St = Set()` → This How to initialize a Set

`St.Add(20) , St.Add(10) , St.add(20) , St.add(-2537) , st.add(10)`

This line (`Print(st)`) → {10, 20, -2537}

`St = {1,5,1,3,5}` → if it's empty → it's dictionary.

`Print(st)` → {1,3,5}

`St = Set(['Ahmed', 'Diaa', 'Kamel'])` → Takes iterable.

`Print(St)` ↗ Return all of them

`Print('zy' in Set)` → Return False.

```
For item in St:                    → No garanted order
    Print(item , end = " ")        → Print item each in aline
Print()                            ↘ Ahmed Diaa Kamel
```

`Print(list[St])` → Turn Set values into list.

`Print(Set({1:10 , -2:30}))` → When you Pass a dictionary to an Iterable Return keys (Iterate over keys only)

When you want to Return value of items → dct.items → only do this)

Print(Set('Hey')) → Here you are Passing 3 items → Bec it iterate over String (iterable)

When you want to Pass it as single item → Print(Set(['Hey']))

2- Print(Set({'Hey'})) Pass it as list or set

Functions

⇒ We Can use Several :

- → len() → number of items in Set
- → max() → Biggest Value → Return max
- → Sorted() → Sort values
- → Sum() → Return Sum of Unique
- → All() → Return True if Value is Not empty or Zero.

Methods

⇒ Some methods :

- → add() → add in Random Place
- → Remove() → Remove if it doesn't exit → Return key error
- → discard() → Remove with No errors.
- → Pop() → Discard any Random item
- → Remove() → Remove all elements.

Set Comperhension

line = "Ahmed Diaa Eldeen Kamel" → line to iterate over it

Unique_vowels = {i for i in line if i in 'AEIOUaeiou'}

↳ if i in line and Print it in a set (Unique Values)

Print(Unique_Vowels) → Print Set

Union and intersection → Str1 = {1,5,7,8}
Str2 = {1,5,3,10}

→ Print(Str1 | Str2) → Union using two integers sets
→ Print(Str1.Union(Str2)) → {1,3,5,7,8,10}
→ Print(Str2.Union([1,-5,-7])) → you can pass any iterable
Note: Str1 is Not Updated

Str3 = {5,6,1}

→ Su = Str1 | Str2 | Str3 → Set of Union

→ Si = Str1 & Str2 & Str3 → Set of interaction

Print(Si) → {1,5}, Print(Str1.intersection(Str2).intersection(Str3)) {1,5}
, Print(Str1.intersection(Str2.Str3)) → All of them give same Result.

Difference → Difference → (Str1 - Str2) → What is found in Str1 Not in Str2, (Str2 - Str1) → What is found in Str2 Not in Str1 → Both called Symetric_difference {in Str1 Not in Str2, in Str2 Not in Str1}

Disjoint → True if there is No interaction Between two Sets.

Subset → (Str1 <= Str2) = (Str1.issubset(Str2)) → True
→ if Str1 is apart of Str2 → All of Str1 in Str2.
→ (Str1 < Str2) → Return True if Str1 elements is in Str2, But Not equal
(Str1 < {1,5}) → They are equal → False.

(str2.issuperset(str1)) = (str2 >= str1) → means that str2 Contain All elements of str1 → Return True

(str1 >= {1,5}) → : str1 Contain All elements of set {1,5}

(str1 > {1,5}) → Suberset But it doesn't have to be equal

Updates → (st1 |= st2) → This means Union and update str1
→ st2.update(st1)
same = "&=" , "^="

Frozenset → immutable set → frozenset([1,2,3,4,5])
Print(id(st1)) → Will Return an address
Can't add / remove (to be used as key in dict)
st1 |= st2 → Will get different memory address
Changed

dct = {str1 : 5} → as this example

for item in sorted(str1): → This to iterate over an set
print(item, end = " ")

Filter Dublicates V2 :

```
def filter_duplicates(lst_of_lsts):
    st = set()                          1- create a set
    result = []                         2- empty list created
    for lst in lst_of_lsts:             3- iterate over every lst
        Tuble = tuple(lst)              4- Turn lst into a tuble
        if Tuble not in st:             5- Check if it's in st or Not
            st.add(Tuble)               6- If Not add it to set
            result.append(lst)          7- append it to result
```