Week7- Dictionary

Introduction :

- Python provides another composite data type called a dictionary, which is similar to a list in that it is a collection of objects.
- Dictionaries are Python's implementation of a data structure that is more generally known as an associative array. Dictionaries are sometimes found in other languages as "associative memories" or "associative arrays".
- Python Dictionary is used to store the data in a key-value pair format.
- It is the mutable data-structure.

Definition:

The dictionary is defined as an unordered collection that contains key- value pairs separated by commas inside curly brackets. Each key separated from its value by a colon (:)

- Dictionary keys must follow the following properties:
  1. Keys must be a single element.
  2. Keys are immutable(Keys can be numbers, string or tuples)
  3. Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.
  4. Dictionary keys are unique
- Dictionary keys must follow the following properties:
  1. Value can be any type such as list, tuple, integer, etc.
  2. Values in a dictionary can be of any data type
  3. Values can be duplicated.

Dictionaries and lists share the following characteristics:

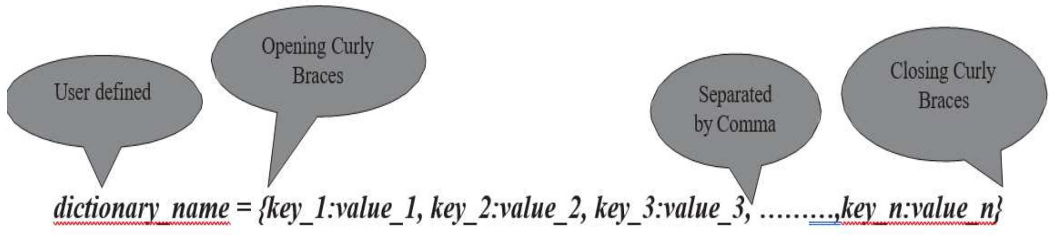
1. Both are mutable.
2. Both are dynamic. They can grow and shrink as needed.
3. Both can be nested. A list can contain another list. A dictionary can contain another dictionary. A dictionary can also contain a list, and vice versa.

Dictionaries differ from lists primarily in how elements are accessed:

1. List elements are accessed by their position in the list, via indexing.
2. Dictionary elements are accessed via keys.

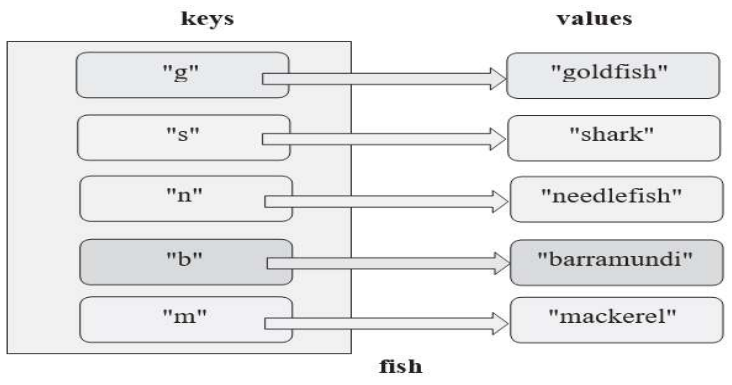
Creating Dictionary

- A dictionary is a collection of an unordered set of *key:value* pairs with the requirement that the keys are unique within a dictionary.
- Dictionaries are constructed using curly braces { }, wherein you include a list of *key:value* pairs separated by commas. Also, thereis a colon (:) separating each of these key and value pairs.
- Dictionaries are indexedby keys. Here a key along with its associated value is called a *key:value* pair.
- Dictionary keys are case sensitive.
- **Syntax of Creating Dictionary:**


User defined
Opening Curly Braces
Separated by Comma
Closing Curly Braces
dictionary_name = {key_1:value_1, key_2:value_2, key_3:value_3, ………,key_n:value_n}

- **Example:**

fish={"g":"goldfish", "s":"shark", "n": "needlefish", ”b”:”barramundi”, “m”:”mackerel”}
print(fish)
# **output:**{'g': 'goldfish', 's': 'shark', 'n': 'needlefish', 'b': 'barramundi', 'm': 'mackerel'}
The following figure demonstrate the representation of key-value pairs of a dictionary



- You can create an empty dictionary by specifying a pair of curly braces and without any key:value pairs. The syntax is,
dictionary_name = { }

For example,

```
 empty_dictionary = {}
 print(empty_dictionary)              # output : {}
 print(type(empty_dictionary))      # output : <class 'dict'>
```

- In dictionaries, the keys and their associated values can be of different types.

  For example,

  ```
  mixed_dict = {"portable":"laptop", 9:11, 179:"gpta"}
  print(mixed_dict)              # {'portable': 'laptop', 9: 11, 179:"gpta"}
  ```

-
  ```
  d={1:"abc", 2:"mno",3:"ghi"}
  d2={2:"mno",3:"ghi",1:"abc"}
  print(d)
  print(d2)
  print(d==d2)   # True
  ```

  - ➢ The dictionaries d and d2 have the same key:value pairs but in a different order, If you compare these two dictionaries, it results in Boolean True value. This indicates that the ordering of key:value pairs does not matter in dictionaries.
- Slicing in dictionaries is not allowed since they are not ordered like lists.

Accessing and Modifying key:value Pairs in Dictionaries

- Each individual key:value pair in a dictionary can be accessed through keys by specifying it inside square brackets.
- The key provided within the square brackets indicates the key:value pair being accessed.
- The syntax for accessing the value for a key in the dictionary is,

  dictionary_name[key]

- The syntax for modifying the value of an existing key or for adding a new key:value pair to a dictionary is,

  dictionary_name[key] = value

  Here,if the key is already present in the dictionary, then the key gets updated with the new value. If the key is not present then the new key:value pair gets added to the dictionary.

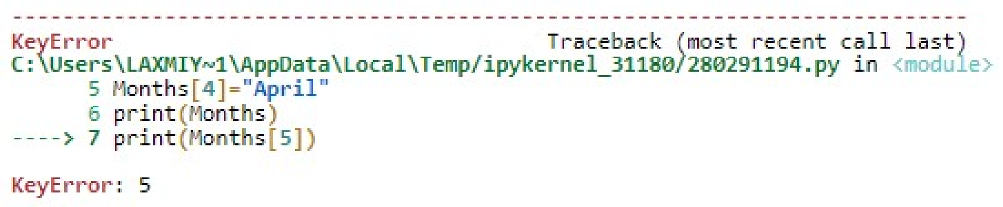
  For example,

```
Months={1:"Jan", 2:"Feb",3:"March"}
print(Months)                  # {1: 'Jan', 2: 'Feb', 3: 'March'}

# Modifying the value  existing key
Months[1]="January"
print(Months)                  #{1: 'January', 2: 'Feb', 3: 'March'}

# Trying to modify the value of non-existing key, in such new key:value
pair gets added into dictionary
Months[4]="April"
print(Months)                  {1: 'January', 2: 'Feb', 3: 'March', 4: 'April'}

# Trying to access the non-existant key from the dictionary raises key
error
print(Months[5])
```



```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
C:\Users\LAXMIY~1\AppData\Local\Temp/ipykernel_31180/280291194.py in <module>
      5 Months[4]="April"
      6 print(Months)
----> 7 print(Months[5])

KeyError: 5
```

Creating Dictionary using dict() :

- The function dict() returns a new dictionary initialized from an optional keyword arguments and a possibly empty set of keyword arguments.
- If no keyword argument is given, an empty dictionary is created.
- If keyword arguments are given, the keyword arguments and their values of the form kwarg = value are added to the dictionary as key:value pairs.

  For example,

  numbers = dict(one=1, two=2, three=3)

print(numbers)   #{'one': 1, 'two': 2, 'three': 3}

- You can also specify an iterable containing exactly two objects as tuple, the key and value in the dict() function.

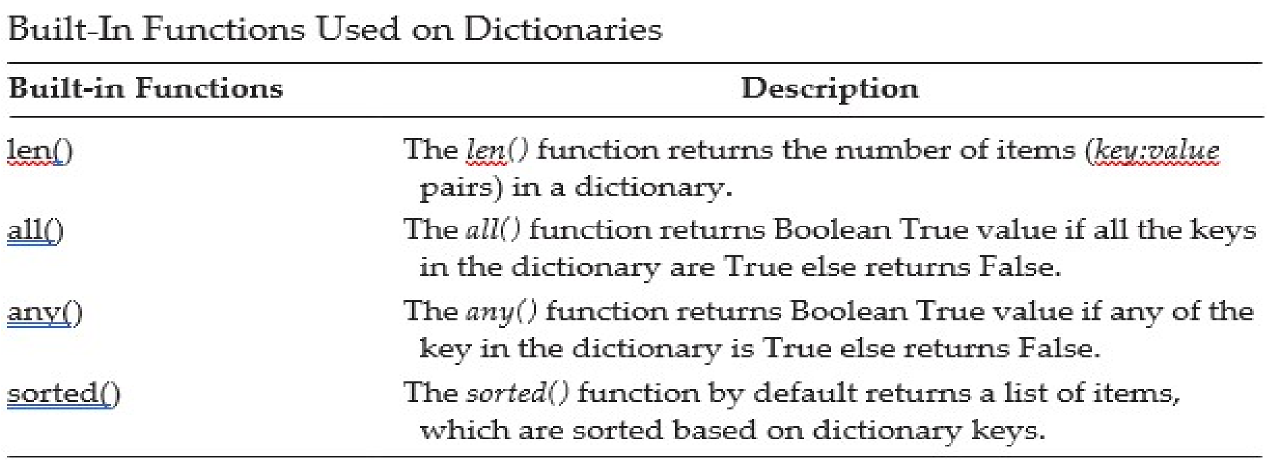
  For example

  numbers=dict([(‘one’,1),(‘two’,2)])

  print(numbers)   # {'one': 1, 'two': 2}

Built-in Function on Dictionary:

The following table shows the various built in function used on dictionary



Built-In Functions Used on Dictionaries

Built-in Functions	Description
len()	The *len()* function returns the number of items (*key:value* pairs) in a dictionary.
all()	The *all()* function returns Boolean True value if all the keys in the dictionary are True else returns False.
any()	The *any()* function returns Boolean True value if any of the key in the dictionary is True else returns False.
sorted()	The *sorted()* function by default returns a list of items, which are sorted based on dictionary keys.

For Example:

```
states_capitals={"Karnataka":"Bengaluru","Maharashtra":"Mumbai","Tamilnadu":"Chennai"
, "Kerala":"Tiruvantapur"}
print(states_capitals)
print("The no of data items in the dictioary are:-",len(states_capitals))
print("The keys of dictionary in asc order:-",sorted(states_capitals))
print("Key in descending order:-", sorted(states_capitals,reverse=True))
print("Arranging the values of dictionary in the dictionary:-")
print(sorted(states_capitals.values()))
print("Arranging the data items in ascending order:-")
```

print(sorted(states_capitals.items()))

Output :

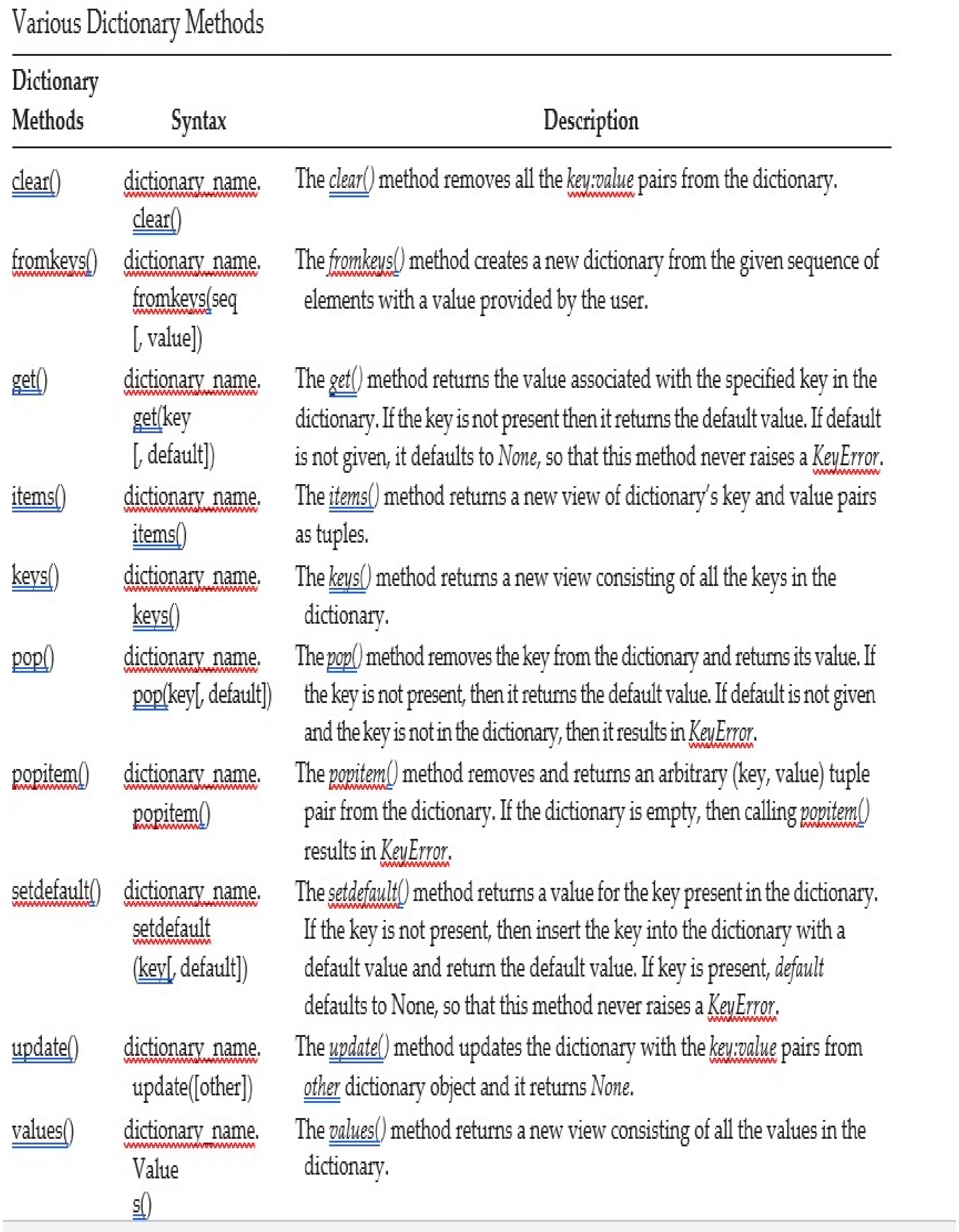
```
{'Karnataka': 'Bengaluru', 'Maharashtra': 'Mumbai', 'Tamilnadu': 'Chennai', 'Kerala':
'Tiruvantapur'}
The no of data items in the dictioary are:- 4
The keys of dictionary in asc order:- ['Karnataka', 'Kerala', 'Maharashtra', 'Tamilnadu']
Key in descending order:- ['Tamilnadu', 'Maharashtra', 'Kerala', 'Karnataka']
Arranging the values of dictionary in the dictionary:-
['Bengaluru', 'Chennai', 'Mumbai', 'Tiruvantapur']
Arranging the data items in ascending order:-
[('Karnataka', 'Bengaluru'), ('Kerala', 'Tiruvantapur'), ('Maharashtra', 'Mumbai'), ('Tamilnadu',
'Chennai')]
```

Let us illustrate how to use any() and all()

```
Any_Dict={0:True,1:False}
All_Dict={1:True,2:False}
print(any(Any_Dict))   # True
print(all(Any_Dict))    #False
print(all(All_Dict))     #True
```

Built-in Dictionary Methods:

The following are the built-in methods



Various Dictionary Methods

Dictionary Methods	Syntax	Description
clear()	dictionary_name. clear()	The *clear()* method removes all the *key:value* pairs from the dictionary.
fromkeys()	dictionary_name. fromkeys(seq [, value])	The *fromkeys()* method creates a new dictionary from the given sequence of elements with a value provided by the user.
get()	dictionary_name. get(key [, default])	The *get()* method returns the value associated with the specified key in the dictionary. If the key is not present then it returns the default value. If default is not given, it defaults to *None*, so that this method never raises a *KeyError*.
items()	dictionary_name. items()	The *items()* method returns a new view of dictionary's key and value pairs as tuples.
keys()	dictionary_name. keys()	The *keys()* method returns a new view consisting of all the keys in the dictionary.
pop()	dictionary_name. pop(key[, default])	The *pop()* method removes the key from the dictionary and returns its value. If the key is not present, then it returns the default value. If default is not given and the key is not in the dictionary, then it results in *KeyError*.
popitem()	dictionary_name. popitem()	The *popitem()* method removes and returns an arbitrary (key, value) tuple pair from the dictionary. If the dictionary is empty, then calling *popitem()* results in *KeyError*.
setdefault()	dictionary_name. setdefault (key[, default])	The *setdefault()* method returns a value for the key present in the dictionary. If the key is not present, then insert the key into the dictionary with a default value and return the default value. If key is present, *default* defaults to None, so that this method never raises a *KeyError*.
update()	dictionary_name. update([other])	The *update()* method updates the dictionary with the *key:value* pairs from *other* dictionary object and it returns *None*.
values()	dictionary_name. Value s()	The *values()* method returns a new view consisting of all the values in the dictionary.

Python program that illustrate all the built in methods:

```
currency = {"India": "Rupee", "USA": "Dollar", "Russia": "Ruble", "Japan":
"Yen", "Germany": "Euro"}

# fromkeys() creates a new dictionary from sequence

currency_fromkeys=currency.fromkeys(currency)

print(currency_fromkeys)

# fromkeys() creates a new dictionary from sequence with default value

currency_fromkeys=currency.fromkeys(currency,"RUPEE")

print(currency_fromkeys)

print("Use of get():")

print(currency.get("India"))

print(currency.get("UK"))

print("Printing the keys from dictionaries:-\n",currency.keys())

print("Printing the values from dictionaries:-\n",currency.values())

print("Printing the data-itemsfrom dictionaries:-\n",currency.items())

print("Updating the dictionary with update() method:-")

currency.update({"UK":"Pound"})

print(currency)

currency.setdefault("minions")

print(currency)

currency.setdefault("Ukrane","Hryvnia")

print(currency)

print("The pop() method returns the specified key and returns its value:")

print(currency.pop("minions"))

print("pop the data item:-")

print(currency.popitem())
```

Output:

```
{'India': None, 'USA': None, 'Russia': None, 'Japan': None, 'Germany': None}

{'India': 'RUPEE', 'USA': 'RUPEE', 'Russia': 'RUPEE', 'Japan': 'RUPEE', 'Germany':
'RUPEE'}

Use of get():

Rupee

None

Printing the keys from dictionaries:-

 dict_keys(['India', 'USA', 'Russia', 'Japan', 'Germany'])

Printing the values from dictionaries:-

 dict_values(['Rupee', 'Dollar', 'Ruble', 'Yen', 'Euro'])

Printing the data-itemsfrom dictionaries:-

 dict_items([('India', 'Rupee'), ('USA', 'Dollar'), ('Russia', 'Ruble'), ('Japan', 'Yen'),
('Germany', 'Euro')])

Updating the dictionary with update() method:-

{'India': 'Rupee', 'USA': 'Dollar', 'Russia': 'Ruble', 'Japan': 'Yen', 'Germany': 'Euro',
'UK': 'Pound'}

{'India': 'Rupee', 'USA': 'Dollar', 'Russia': 'Ruble', 'Japan': 'Yen', 'Germany': 'Euro',
'UK': 'Pound', 'minions': None}

{'India': 'Rupee', 'USA': 'Dollar', 'Russia': 'Ruble', 'Japan': 'Yen', 'Germany': 'Euro',
'UK': 'Pound', 'minions': None, 'Ukrane': 'Hryvnia'}

The pop() method returns the specified key and returns its value:

None

pop the data item:-

('Ukrane', 'Hryvnia')
```

Traversing of Dictionary

- A for loop can be used to iterate over keys or values or key:value pairs in dictionaries.
- If you iterate over a dictionary using a for loop, then, by default, you will iterate over the keys.
- If you want to iterate over the values, use values() method
- And for iterating over the key:value pairs, specify the dictionary's items() method explicitly.
- The dict_keys, dict_values, and dict_items data types returned by dictionary methods can be used in for loops to iterate over the keys or values or key:value pairs.

Program to Illustrate Traversing of key:value Pairs in Dictionaries Using for Loop

```
currency = {"India": "Rupee", "USA": "Dollar", "Russia": "Ruble",
"Japan": "Yen","Germany": "Euro"}
print("List of Countries")
for key in currency.keys():
        print(key)
print("List of Currencies in different Countries")
for value in currency.values():
        print(value)
for key, value in currency.items():
    print(f"'{key}' has a currency of type '{value}'")
```

OUTPUT

```
List of Countries India
USA
Russia Japan Germany
List of Currencies in different Countries Rupee
Dollar Ruble Yen Euro
'India' has a currency of type 'Rupee'
'USA' has a currency of type 'Dollar'
'Russia' has a currency of type 'Ruble'
'Japan' has a currency of type 'Yen'
 'Germany' has a currency of type 'Euro'
```

Write Python Program to Check for the Presence of a Key in the Dictionary and to Sum All Its Values

```
historical_events = {"Independence": 1947, "Republic": 1950, “NEP”:2020}
key = input("Enter the key to check for its presence:-")
if key in historical_events.keys():
        print(f"Key '{key}' is present in the dictionary")
else:
        print(f"Key '{key}' is not present in the dictionary")
print("Sum of all the values in the dictionary is")
print(f"{sum(historical_events.values())}")
```

OUTPUT:

```
Enter the key to check for its presence:- NEP
Key 'NEP' is present in the dictionary
Sum of all the values in the dictionary is
5917
```

** Write Python Program to Generate a Dictionary That Contains (i: i*i) Such that i Is a Number Ranging from 1 to n.

```
number = int(input("Enter a number "))
create_number_dict = dict()
for i in range(1, number+1):
        create_number_dict[i] = i * i
print("The generated dictionary of the form (i: i*i) is")
print(create_number_dict)
```

OUTPUT:

```
Enter a number 10
The generated dictionary of the form (i: i*i) is
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

** Write a Program That Accepts a Sentence and Calculate the Number of Digits, Uppercase and Lowercase Letters **

```
sentence = input("Enter a sentence ")
construct_dictionary = {"digits": 0, "lowercase": 0, "uppercase": 0,'other_special':0}
for each_character in sentence:
    if each_character.isdigit():
        construct_dictionary["digits"] += 1
    elif each_character.isupper():
        construct_dictionary["uppercase"] += 1
    elif each_character.islower():
        construct_dictionary["lowercase"] += 1
    else:
         construct_dictionary['other_special']+=1
print("The number of digits, lowercase and uppercase letters are")
print(construct_dictionary)
```

OUTPUT:

```
Enter a sentence Hello i am Student studying @179 GPT ATHANI
The number of digits, lowercase and uppercase letters are
{'digits': 3, 'lowercase': 21, 'uppercase': 11, 'other_special': 8}
```

What is Dictionary Comprehension in Python?

Dictionary comprehension is an elegant and concise way to create dictionaries.

The minimal syntax for dictionary comprehension is:

```
dictionary = {key: value for vars in iterable}
```

Example:

1) Create a dictionary that contains numbers in the form { i:i**2}from a range of number 1to 10 using comprehension

```
square_dict = {num: num*num for num in range(1, 11)}
print(square_dict)
# {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

2) #Conditonal Dictionary Comprehension

```
original_dict = {'jack': 38, 'michael': 48, 'guido': 57, 'john': 33}

even_dict = {k: v for (k, v) in original_dict.items() if v % 2 == 0}
print(even_dict)
```

Populating Dictionaries with key:value Pairs (Building dictionary incrementally)

- One of the common ways of populating dictionaries is to start with an empty dictionary{ }, then use the update() method to assign a value to the key using assignment operator. If the key does not exist, then the key:value pairs will be created automatically and added to the dictionary.
- The dictionary can also be built using dictionary_name[key] = value syntax by adding key:value pairs to the dictionary.

- 1st Method: Using update() and for loop

```
print("Method 1: Building Dictionaries")
build_dictionary = {}
for i in range(0, 2):
    dic_key = input("Enter key ")
    dic_val = input("Enter val ")
    build_dictionary.update({dic_key: dic_val})
print(f"Dictionary is {build_dictionary}")
```

- 2nd Method: Using Assignment statement and for loop

```
print("Method 2: Building Dictionaries")
build_dictionary = {}
for i in range(0, 2):
        dic_key = input("Enter key ")
        dic_val = input("Enter val ")
build_dictionary[dic_key] = dic_val
print(f"Dictionary is {build_dictionary}")
```

- 3rd Method: Using update() and for loop

```
print("Method 3: Building Dictionaries")
build_dictionary = {}
i = 0
while i < 2:
    dict_key = input("Enter key ")
    dict_val = input("Enter val ")
    build_dictionary.update({dict_key: dict_val})
    i = i + 1
print(f"Dictionary is {build_dictionary}")
```

OUTPUT

```
Method 3: Building Dictionaries
Enter key NAME
Enter val LAXMI
Enter key REGISTER NO
Enter val 1DA05CS034
Dictionary is {'NAME': 'LAXMI', 'REGISTER NO': '1DA05CS034'}
```