In python every iterable can be unpacked

but when it comes to dictionary we can only unpack dictionaries keys() but we can not access its values

Dictionary

1. it is a collection of key value pairs
2. ordered data type
3. it is mutable
4. duplicate keys are not allowed, but values can be duplicate
5. We can not access items by using an index
6. dictionaries are comma separated
7. syntax and dictionary constructor :-

> ```
> - {"key1":"value1", "key2":"value2", "key3":"value3"}
> - dict()
> ```

8. orderd datatype in python version >= 3.7
9. unorderd datatype in python version <= 3.7

```
key        ===> Immutable datatypes (int, float, string, tuple, frozenset)
value      ===> Any datatype (int, float, list, set, dict, string, tuple, frozenset, etc...)
```

In [ ]:

```

```

1. Access Dictionary items

```
1. dict_name[Key_Name]

    - if the key is not present in a dictionary then this  method will return error

2. dict_name.get(key_name)

    - if the key is not present in a dictionary then this get method won't return any error
```

2. Access dictionary keys

```
1. dict_name.keys()

2. for key in dict_name.keys():
         print(key)
```

3. Access dictionary values

```
- dict_name.values()
- for value in dict_name.values():
    print(values)
```

4. Accessing items from a dictionary

```
- dict_name.items()
- for i in dict_name.items():
    print(i)
```

In [ ]:

```

```

5. Add values in dictionary or update dictionary

```
1. dict_name[key_name] = value_name
```

tuple unpacking

```
- (a, b) -- key = a, value = b

- for key, value in (a,b):
        print(key, value)

```

merge two dictionaries

1. dict1.update(dict2)
2. new_dict = dict1 | dict2
    - This fuction can mearge two or more than two dictionaries but not any iterators except dictionary
3. zip(iterator1, iterator2)
    - This operator can accept more than two iterators
    - if the iterators has less element than iterator second iterator, then it just going to ignore extra elemets of the iterators and will make a pair of equal elements of tuple
4. new_dict = dict.fromkeys(itorator2 or (keys), iterator2 or (values))
    - The fromkeys() method returns a dictionary with the specified keys and the specified value.

In [ ]:

6. Dictionary Comprehension

```
1. new_dict  =  { new_key:new_value for (key, value) in dict_name.items() }

2. new_dict  =  { new_key : new_value for (key, value) in dict_name.items() if condition }

3. new_dict  =  { new_key : new_key if condition else statement new_key : new_key for (key, value)
in dict_name.item() }
```

In [ ]:

7. Combining iterators to form new dictionary

1. Adding dictionary to dictionary

```
- dict1.update(dict2)

- dict3 = dict1 | dict2 | dict3 ... | ...

- new_dict = {**dict1, **dict2, **dict4 ...}
```

2. Adding iterator to iterator

```
# first_iter = iterator1 is going to be a key
# second_iter = iterator2 is going to be a value

```

```
4 - dict(zip(iterator1, iterator2))

```

3. Assigning values to the iterators

Syntax :

syntax = dict.fromkeys(seq, val)

- Parameters :
  - seq : The sequence to be transformed into a dictionary.
  - val : Initial values that need to be assigned to the generated keys. Defaults to None.
- Returns : A dictionary with keys mapped to None if no value is provided, else to the value provided in the field.

4. setdefault()

syntax:-

syntax = dict.setdefault(keyname, value)

- Python Dictionary setdefault() returns the value of a key (if the key is in dictionary). Else, it inserts a key with the default value to the dictionary.
  - defaule value is None

In [ ]:

```

```

8. Delete methods in Dictionary

```
1. pop(key) :- This method will delete perticular key and there value from the dictionary.
```

```
2. .popitem() :- This method will  delete last key value pair of the dictionary

3. .clear()   :- This method will delete all the items from the dictionary and will return blank dictionary

4. del dict_name[key] or del dict_name :- This method will completly delete a dictionary or it can delete a specified key value from the dictionary
```

9. sort dictionary

- new_dict = sorted(dict_name.keys(), reverse = True)
    - This sorted method will sort the keys according to order number of the characters or assending order of a number.
    - reverse True will reverse the keys in reverse order

In [ ]:

```
## 10.
```