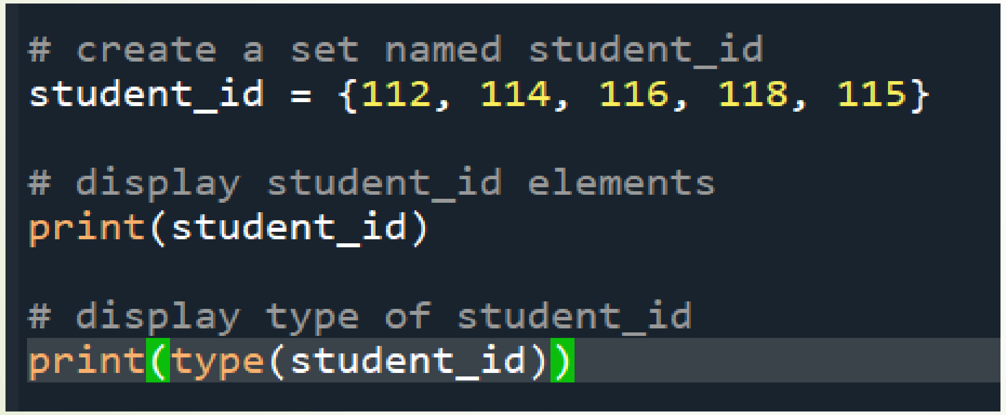
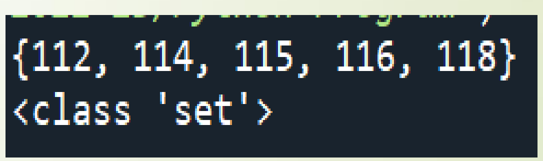
Python Programming

Python Set Data Type:

- Set is an unordered collection of unique items. Set is defined by values separated by commas inside braces { }.
- A set is a collection of unique data. That is, elements of a set cannot be duplicate.



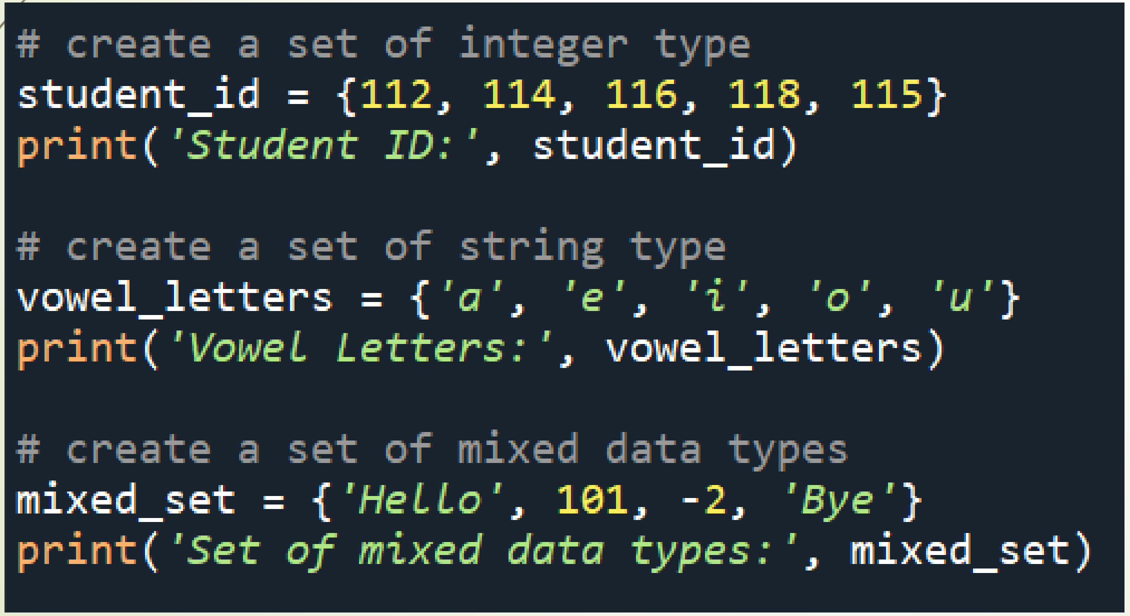
```python
# create a set named student_id
student_id = {112, 114, 116, 118, 115}

# display student_id elements
print(student_id)

# display type of student_id
print(type(student_id))
```



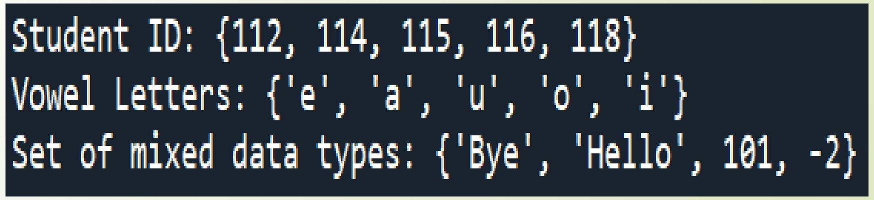
```
{112, 114, 115, 116, 118}
<class 'set'>
```

Create a Set in Python:

- In Python, we create sets by placing all the elements inside curly braces {}, separated by comma.
- A set can have any number of items and they may be of different types (integer, float, tuple, string etc.).



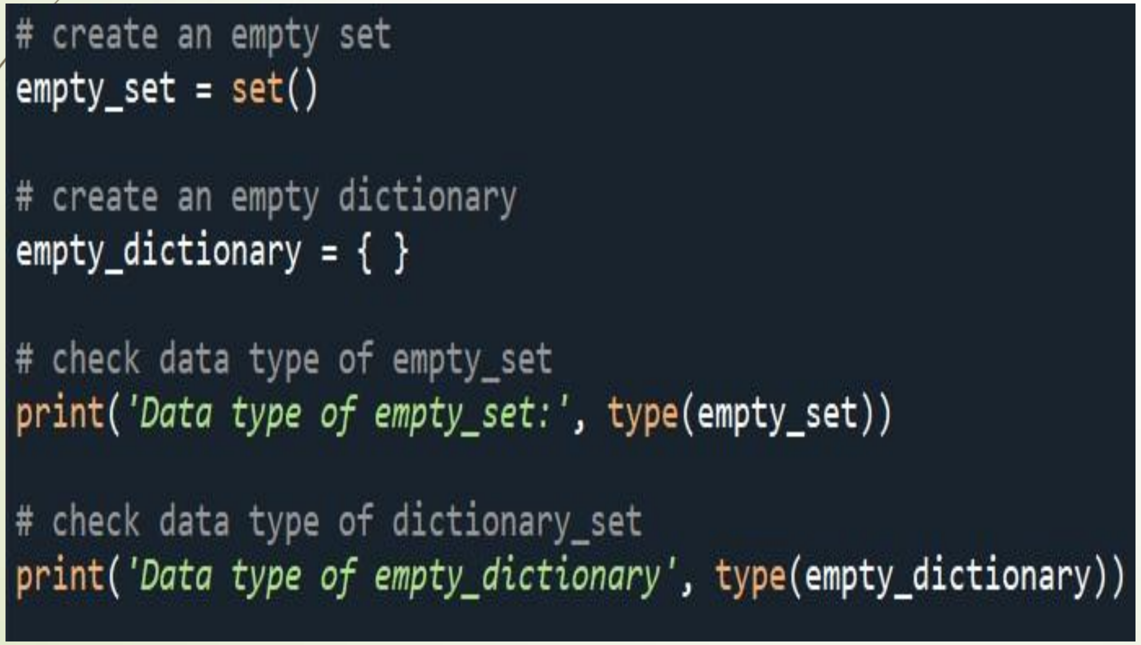
```python
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)

# create a set of string type
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
print('Vowel Letters:', vowel_letters)

# create a set of mixed data types
mixed_set = {'Hello', 101, -2, 'Bye'}
print('Set of mixed data types:', mixed_set)
```



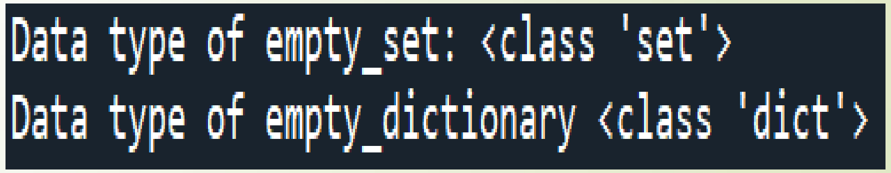
```
Student ID: {112, 114, 115, 116, 118}
Vowel Letters: {'e', 'a', 'u', 'o', 'i'}
Set of mixed data types: {'Bye', 'Hello', 101, -2}
```

Create an Empty Set in Python:

- Creating an empty set is a bit tricky. **Empty curly braces {} will make an empty dictionary in Python.**
- To make a set without any elements, we use the **set() function without any argument.**



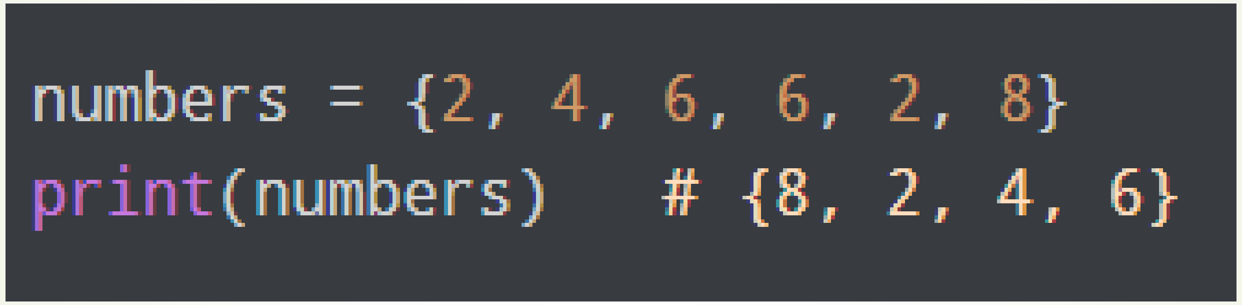
```
# create an empty set
empty_set = set()

# create an empty dictionary
empty_dictionary = { }

# check data type of empty_set
print('Data type of empty_set:', type(empty_set))

# check data type of dictionary_set
print('Data type of empty_dictionary', type(empty_dictionary))
```



```
Data type of empty_set: <class 'set'>
Data type of empty_dictionary <class 'dict'>
```

Duplicate Items in a Set:

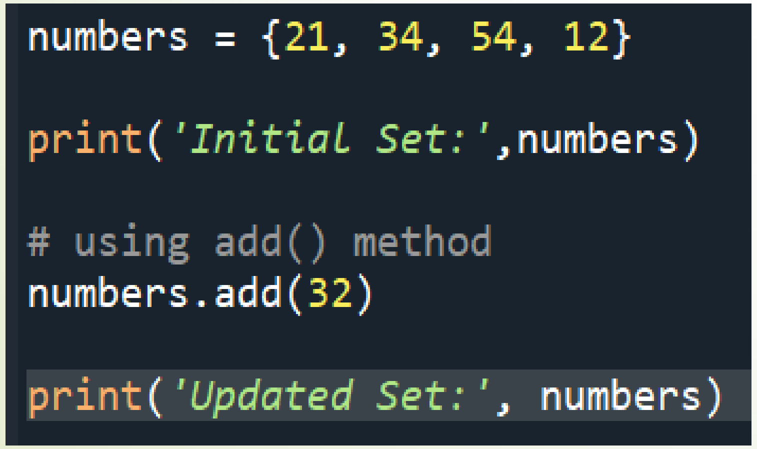
There are no duplicate items in the set as a set cannot contain duplicates.



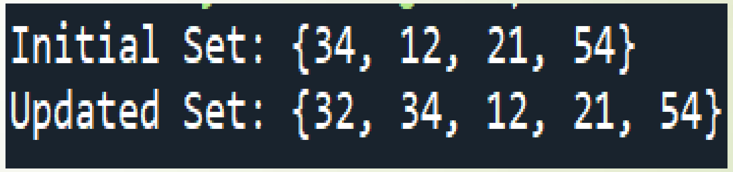
```
numbers = {2, 4, 6, 6, 2, 8}
print(numbers)   # {8, 2, 4, 6}
```

Add and Update Set Items in Python:

- Sets are mutable. However, since they are unordered, indexing has no meaning.
- We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.
- In Python, we use the **add() method** to add an item to a set. For example,



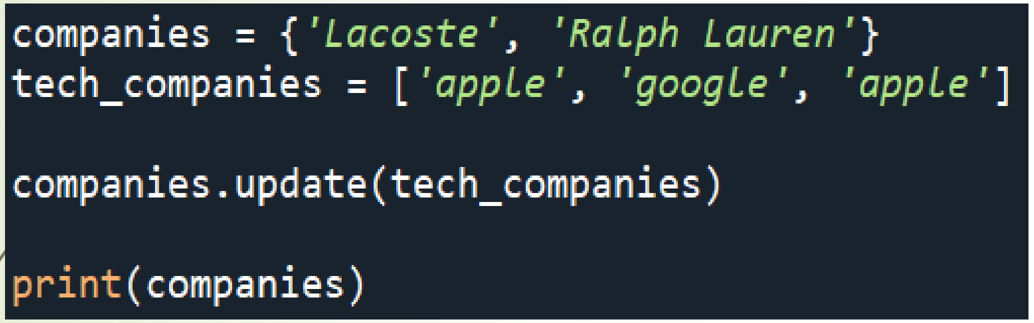
```
numbers = {21, 34, 54, 12}

print('Initial Set:',numbers)

# using add() method
numbers.add(32)

print('Updated Set:', numbers)
```



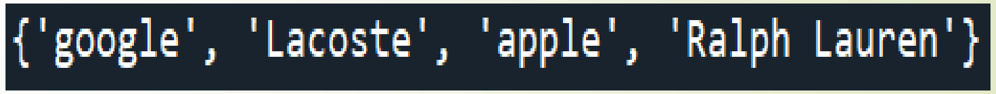
```
Initial Set: {34, 12, 21, 54}
Updated Set: {32, 34, 12, 21, 54}
```

Update Python Set:

- The update() method is used to update the set with items other collection types (lists, tuples, sets, etc).



```
companies = {'Lacoste', 'Ralph Lauren'}
tech_companies = ['apple', 'google', 'apple']

companies.update(tech_companies)

print(companies)
```

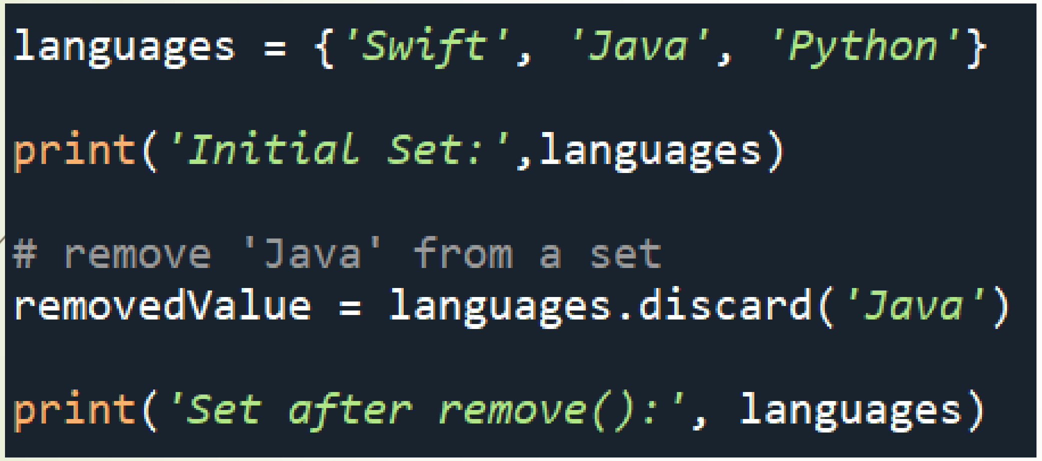


```
{'google', 'Lacoste', 'apple', 'Ralph Lauren'}
```

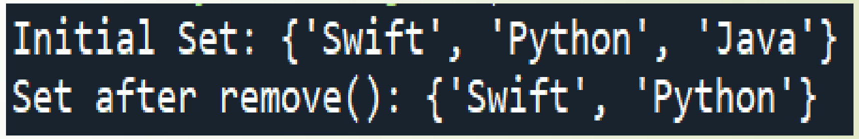
- Here, all the unique elements of tech_companies are added to the companies set.

Remove an Element from a Set:

- We use the discard() method to remove the specified element from a set. For example,



```
languages = {'Swift', 'Java', 'Python'}

print('Initial Set:',languages)

# remove 'Java' from a set
removedValue = languages.discard('Java')

print('Set after remove():', languages)
```

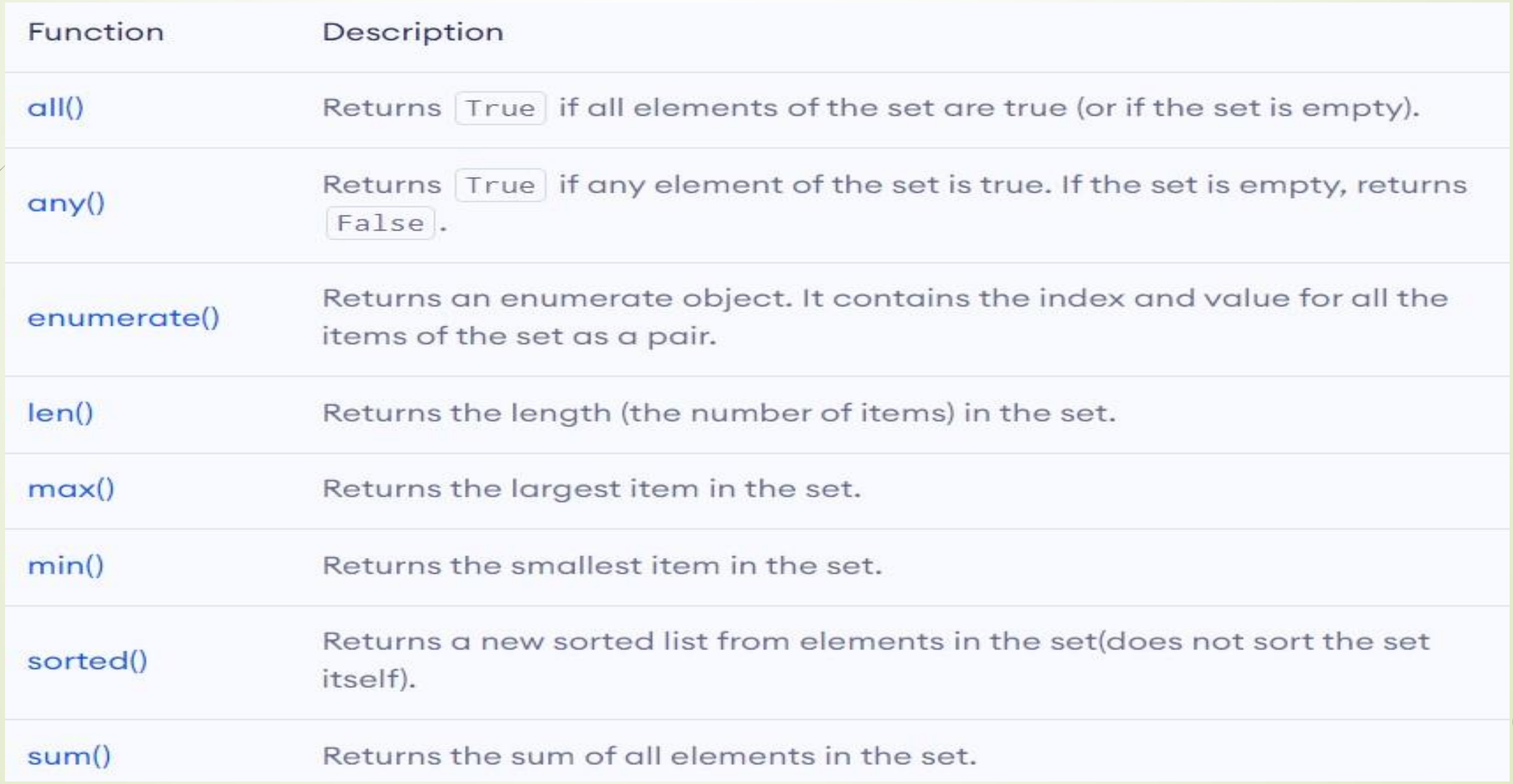


```
Initial Set: {'Swift', 'Python', 'Java'}
Set after remove(): {'Swift', 'Python'}
```

- Here, we have used the discard() method to remove 'Java' from the languages set.

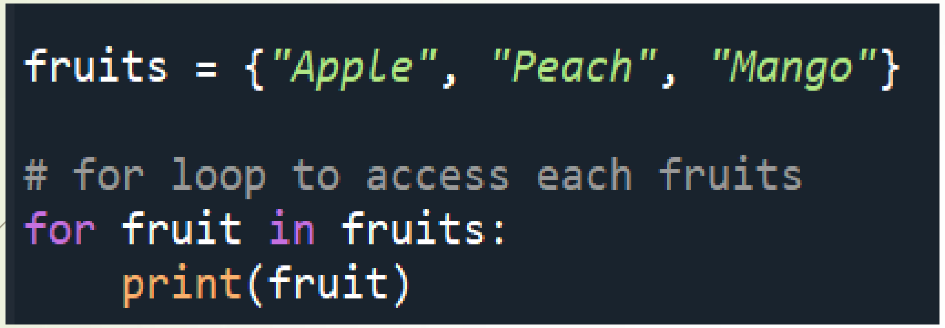
Built-in Functions with Set:

- Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with sets to perform different tasks.

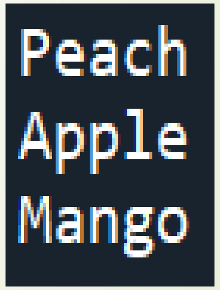


Function	Description
all()	Returns `True` if all elements of the set are true (or if the set is empty).
any()	Returns `True` if any element of the set is true. If the set is empty, returns `False`.
enumerate()	Returns an enumerate object. It contains the index and value for all the items of the set as a pair.
len()	Returns the length (the number of items) in the set.
max()	Returns the largest item in the set.
min()	Returns the smallest item in the set.
sorted()	Returns a new sorted list from elements in the set(does not sort the set itself).
sum()	Returns the sum of all elements in the set.

Iterate Over a Set in Python:



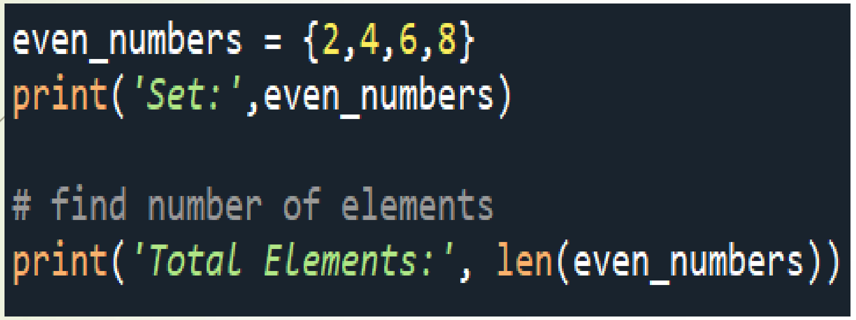
```
fruits = {"Apple", "Peach", "Mango"}

# for loop to access each fruits
for fruit in fruits:
    print(fruit)
```



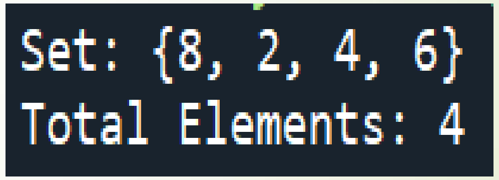
```
Peach
Apple
Mango
```

Find Number of Set Elements:

 Use the len() method to find the number of elements present in a Set.



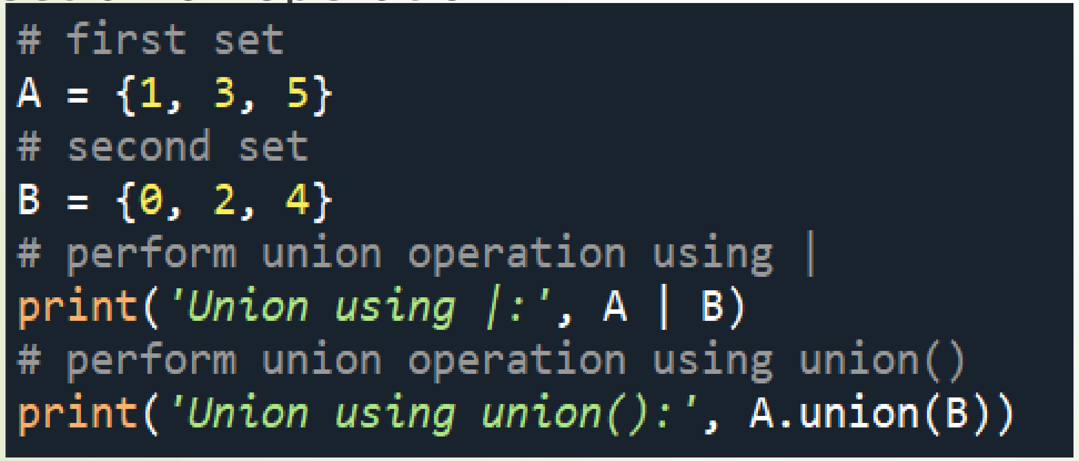
```
even_numbers = {2,4,6,8}
print('Set:',even_numbers)

# find number of elements
print('Total Elements:', len(even_numbers))
```



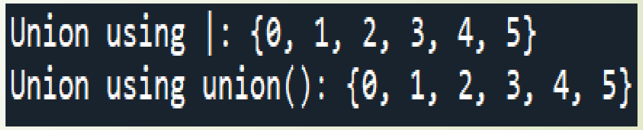
```
Set: {8, 2, 4, 6}
Total Elements: 4
```

Python Set Operations: Union of Two Sets

- Python Set provides different built-in methods to perform mathematical set operations like union, intersection, subtraction, and symmetric difference.
- The union of two sets A and B include all the elements of set A and B.
- We use the **|** operator or the **union()** method to perform the set union operation.



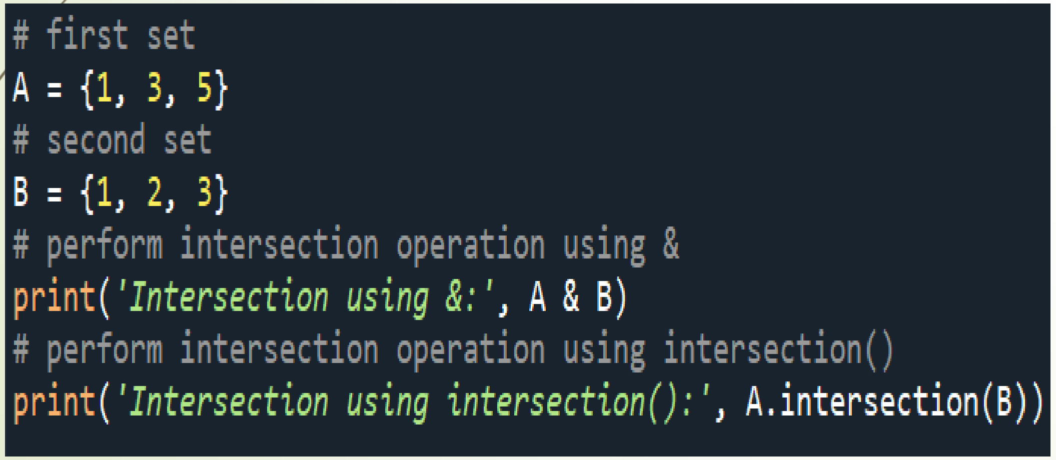
```
# first set
A = {1, 3, 5}
# second set
B = {0, 2, 4}
# perform union operation using |
print('Union using |:', A | B)
# perform union operation using union()
print('Union using union():', A.union(B))
```



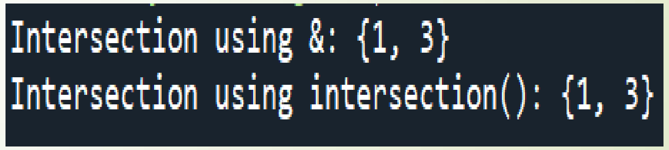
```
Union using |: {0, 1, 2, 3, 4, 5}
Union using union(): {0, 1, 2, 3, 4, 5}
```

Python Set Operations: Set Intersection

- The intersection of two sets A and B include the common elements between set A and B.
- We use the **&** operator or the **intersection()** method to perform the set intersection operation.



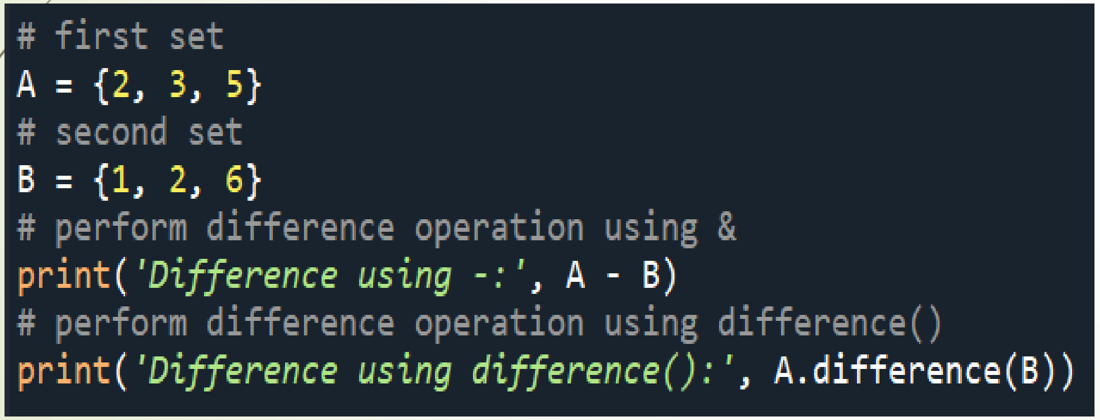
```
# first set
A = {1, 3, 5}
# second set
B = {1, 2, 3}
# perform intersection operation using &
print('Intersection using &:', A & B)
# perform intersection operation using intersection()
print('Intersection using intersection():', A.intersection(B))
```



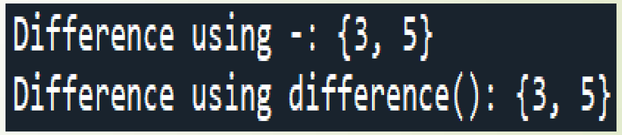
```
Intersection using &: {1, 3}
Intersection using intersection(): {1, 3}
```

Python Set Operations: Difference between Two Sets:

- The difference between two sets A and B **include elements of set A that are not present on set B.**
- We use the - operator or the difference() method to perform the
  difference between two sets.



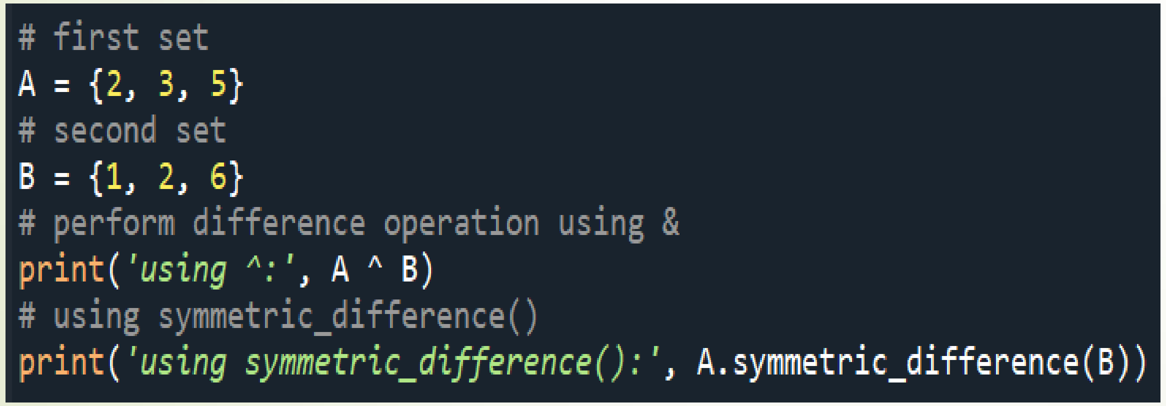
```
# first set
A = {2, 3, 5}
# second set
B = {1, 2, 6}
# perform difference operation using &
print('Difference using -:', A - B)
# perform difference operation using difference()
print('Difference using difference():', A.difference(B))
```



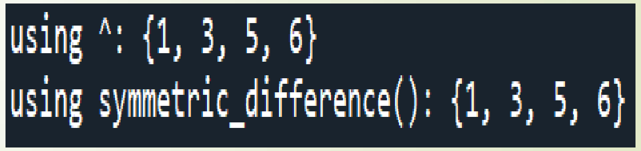
```
Difference using -: {3, 5}
Difference using difference(): {3, 5}
```

Python Set Operations: Set Symmetric Difference:

- The symmetric difference between two sets A and B includes all elements of A and B without the common elements.
- In Python, we use the ^ operator or the **symmetric_difference()** method to perform symmetric difference between two sets.



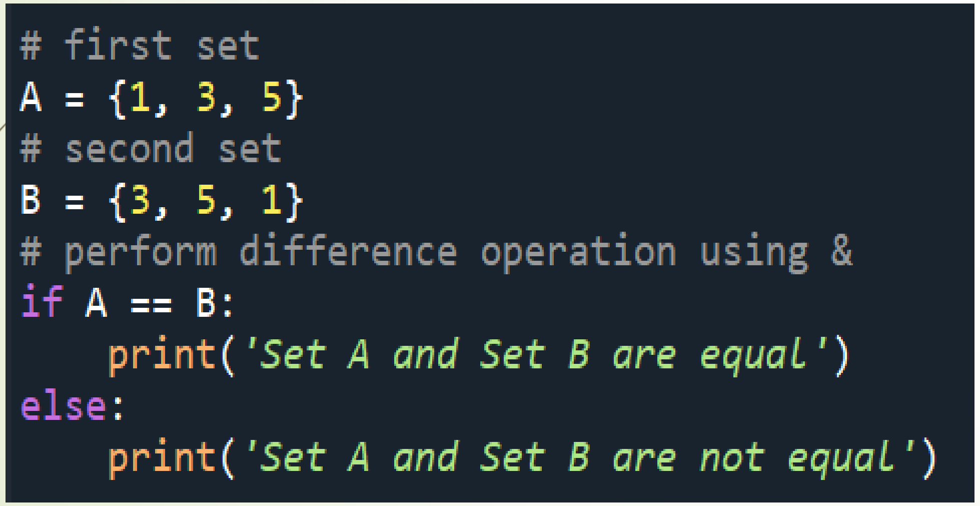
```
# first set
A = {2, 3, 5}
# second set
B = {1, 2, 6}
# perform difference operation using &
print('using ^:', A ^ B)
# using symmetric_difference()
print('using symmetric_difference():', A.symmetric_difference(B))
```



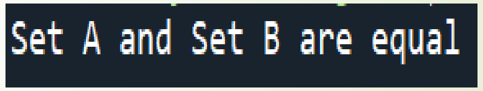
```
using ^: {1, 3, 5, 6}
using symmetric_difference(): {1, 3, 5, 6}
```

Python Set Operations: Check if two sets are equal:

- We can use the == operator to check whether two sets are equal or not.



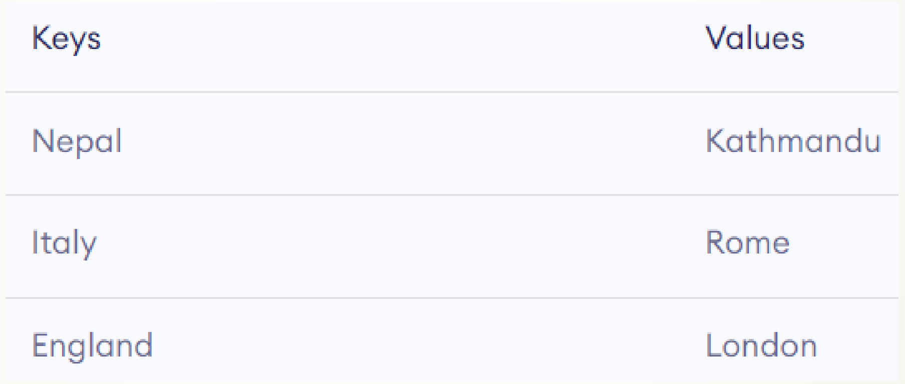
```
# first set
A = {1, 3, 5}
# second set
B = {3, 5, 1}
# perform difference operation using &
if A == B:
    print('Set A and Set B are equal')
else:
    print('Set A and Set B are not equal')
```



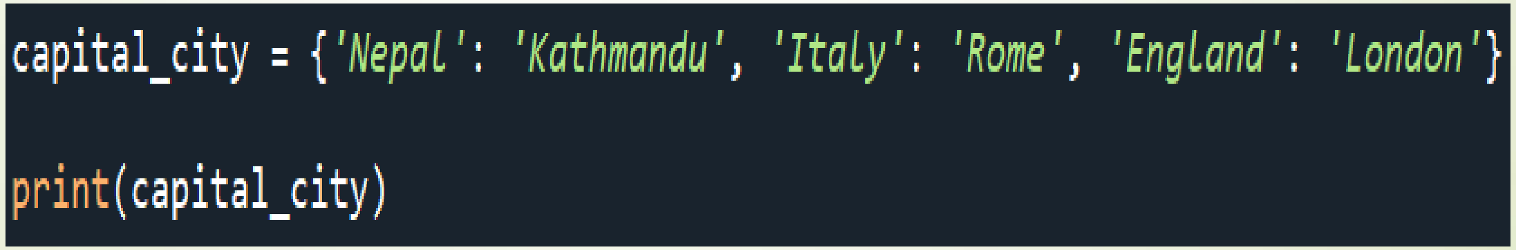
```
Set A and Set B are equal
```

Python Dictionary Data Type:

- Python dictionary is an ordered collection of items. It stores elements in key/value pairs.
- Here, keys are unique identifiers that are associated with each value.



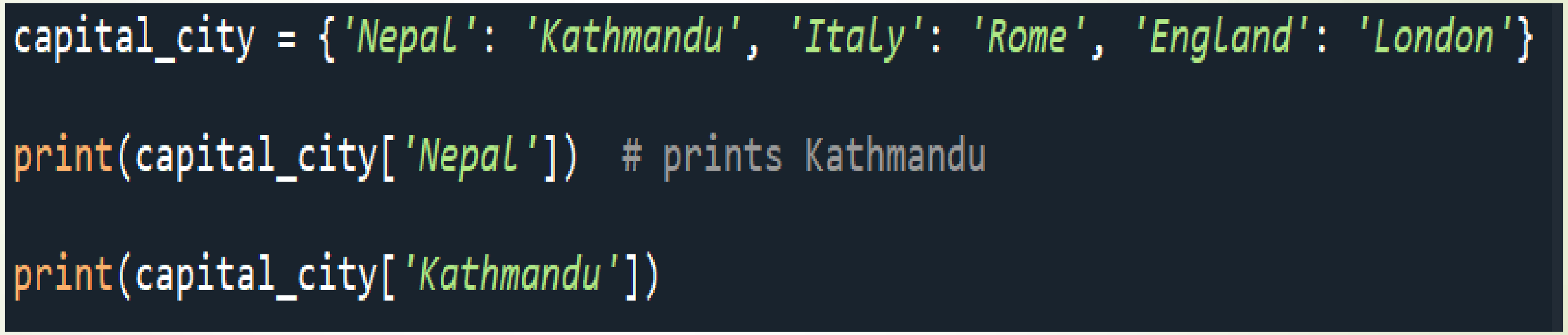
Keys	Values
Nepal	Kathmandu
Italy	Rome
England	London



```
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

print(capital_city)
```

Python Dictionary Data Type:

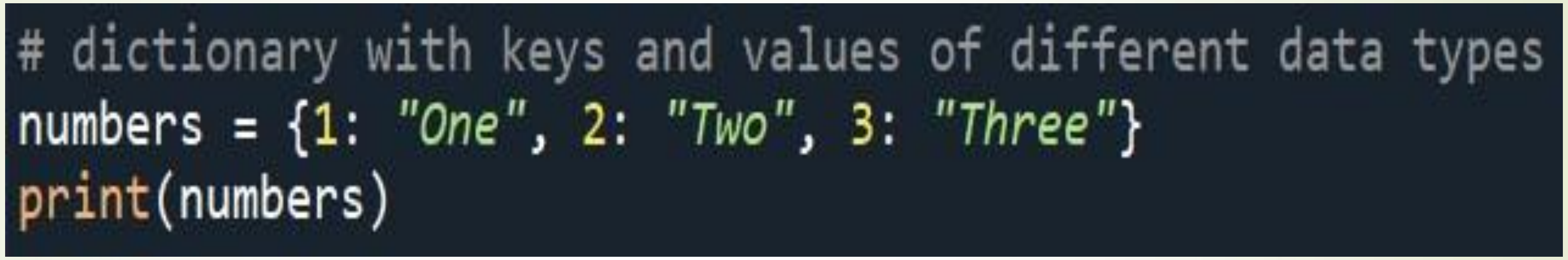
- We use keys to retrieve the respective value. But not the other way around. For example,



```
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

print(capital_city['Nepal'])  # prints Kathmandu

print(capital_city['Kathmandu'])
```

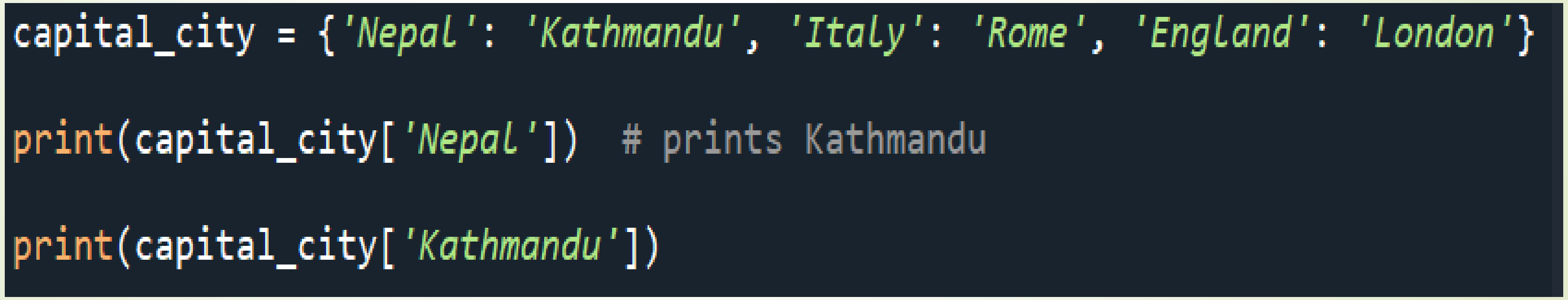
- Note: Here, keys and values both are of string type. We can also have keys and values of different data types.

Python Dictionary Data Type:



```python
# dictionary with keys and values of different data types
numbers = {1: "One", 2: "Two", 3: "Three"}
print(numbers)
```

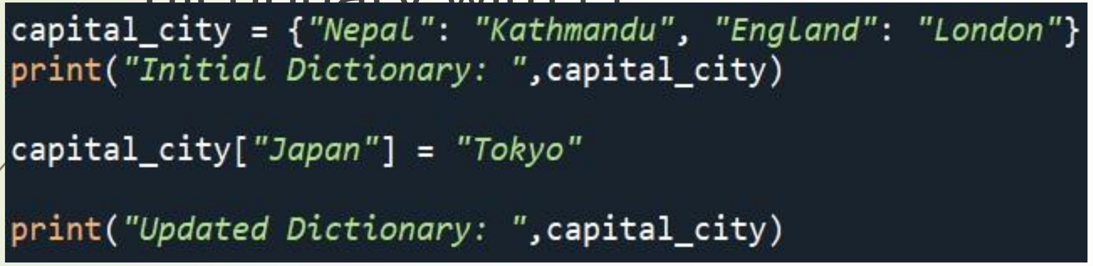
- We use keys to retrieve the respective value. But not the other way around. For example,



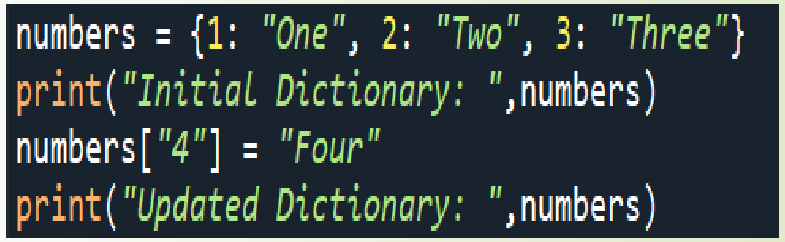
```python
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

print(capital_city['Nepal'])  # prints Kathmandu

print(capital_city['Kathmandu'])
```

Add Elements to a Python Dictionary:

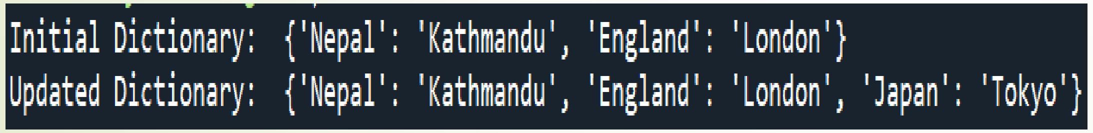
- We can add elements to a dictionary using the name of the
  dictionary with []



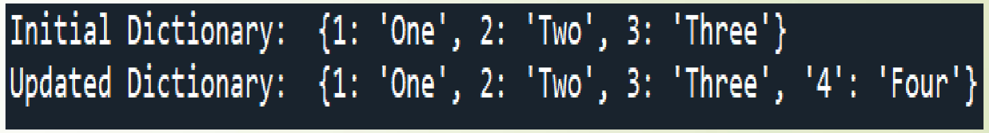
```python
capital_city = {"Nepal": "Kathmandu", "England": "London"}
print("Initial Dictionary: ",capital_city)

capital_city["Japan"] = "Tokyo"

print("Updated Dictionary: ",capital_city)
```



```python
numbers = {1: "One", 2: "Two", 3: "Three"}
print("Initial Dictionary: ",numbers)
numbers["4"] = "Four"
print("Updated Dictionary: ",numbers)
```



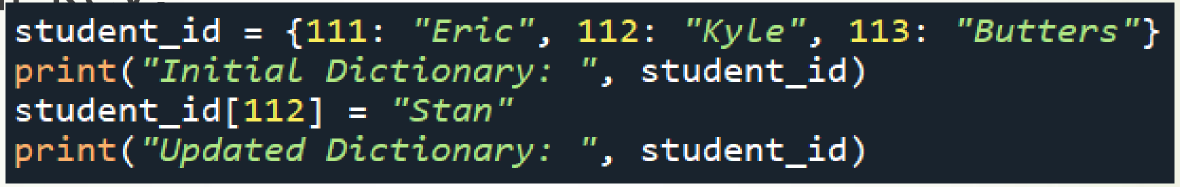
```
Initial Dictionary:  {'Nepal': 'Kathmandu', 'England': 'London'}
Updated Dictionary:  {'Nepal': 'Kathmandu', 'England': 'London', 'Japan': 'Tokyo'}
```



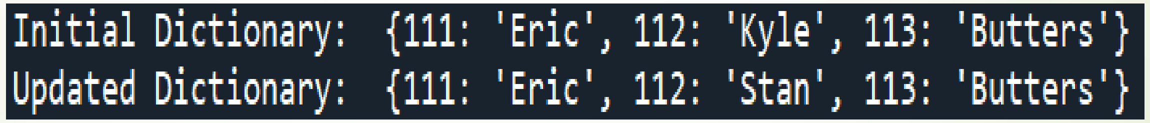
```
Initial Dictionary:  {1: 'One', 2: 'Two', 3: 'Three'}
Updated Dictionary:  {1: 'One', 2: 'Two', 3: 'Three', '4': 'Four'}
```

Change Value of Dictionary:

- We can also use [] to change the value associated with a particular key.



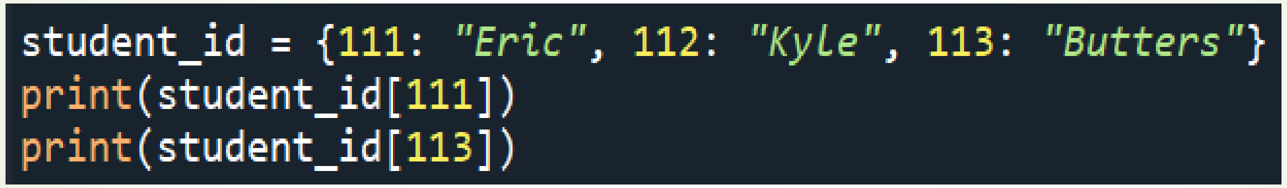
```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print("Initial Dictionary: ", student_id)
student_id[112] = "Stan"
print("Updated Dictionary: ", student_id)
```



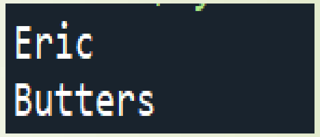
```
Initial Dictionary:  {111: 'Eric', 112: 'Kyle', 113: 'Butters'}
Updated Dictionary:  {111: 'Eric', 112: 'Stan', 113: 'Butters'}
```

Accessing Elements from Dictionary:

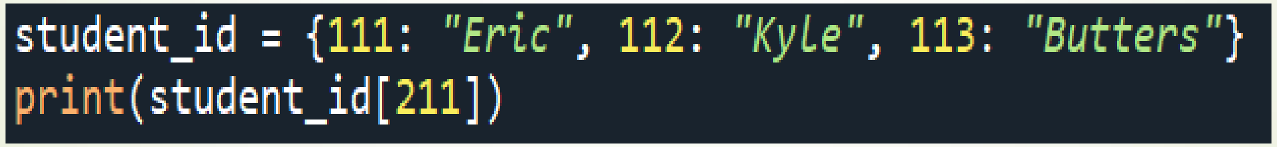
- we use the keys to access their corresponding values.



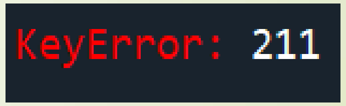
```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print(student_id[111])
print(student_id[113])
```



```
Eric
Butters
```



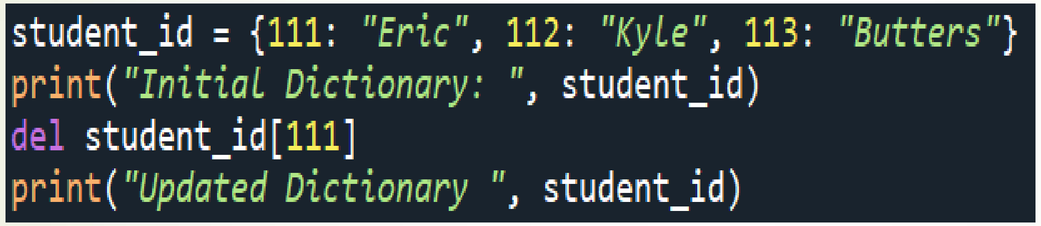
```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print(student_id[211])
```



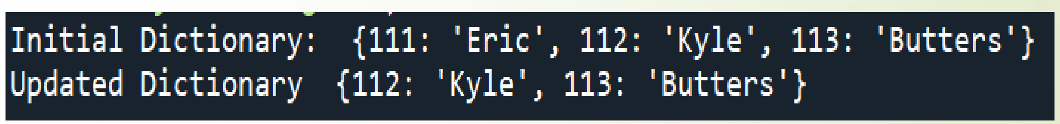
```
KeyError: 211
```

Removing elements from Dictionary:

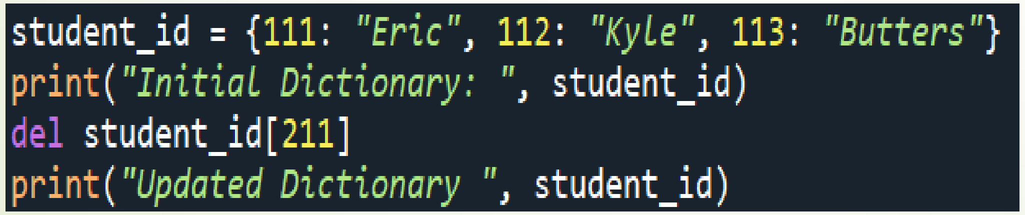
- We use the del statement to remove an element from the dictionary.



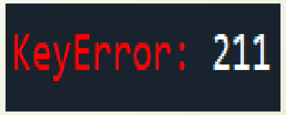
```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print("Initial Dictionary: ", student_id)
del student_id[111]
print("Updated Dictionary ", student_id)
```



```
Initial Dictionary:  {111: 'Eric', 112: 'Kyle', 113: 'Butters'}
Updated Dictionary  {112: 'Kyle', 113: 'Butters'}
```



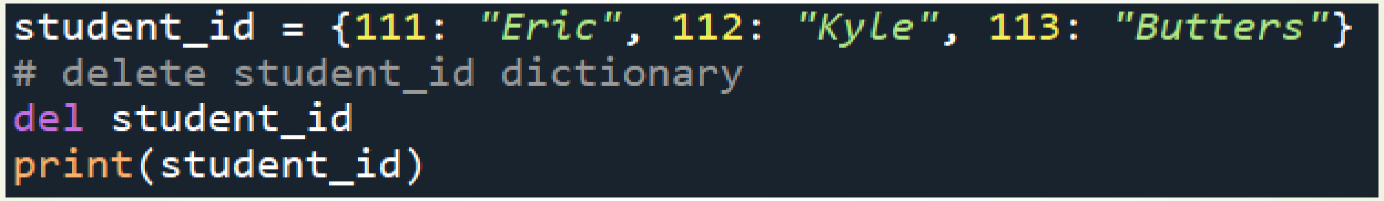
```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print("Initial Dictionary: ", student_id)
del student_id[211]
print("Updated Dictionary ", student_id)
```



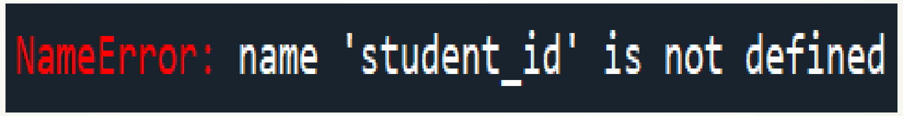
```
KeyError: 211
```

Removing elements from Dictionary:

- We can also delete the whole dictionary using the del statement.



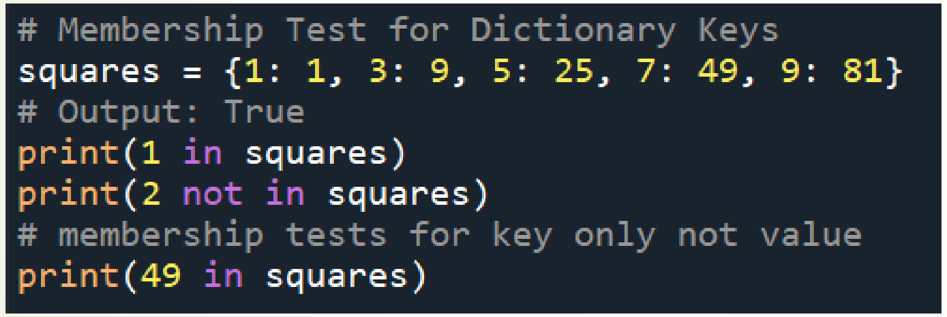
```python
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
# delete student_id dictionary
del student_id
print(student_id)
```



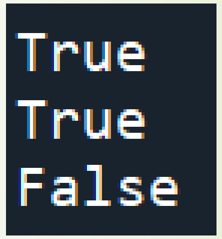
```
NameError: name 'student_id' is not defined
```

Dictionary Membership Test:

- We can test if a key is in a dictionary or not using the keyword in. Notice that the membership test is only for the keys and not for the values.



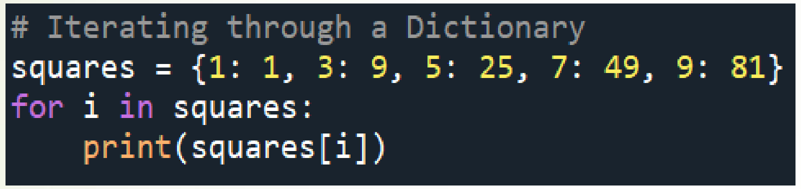
```
# Membership Test for Dictionary Keys
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
# Output: True
print(1 in squares)
print(2 not in squares)
# membership tests for key only not value
print(49 in squares)
```



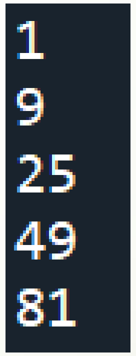
```
True
True
False
```

Iterating Through a Dictionary

- We can iterate through each key in a dictionary using a for loop.

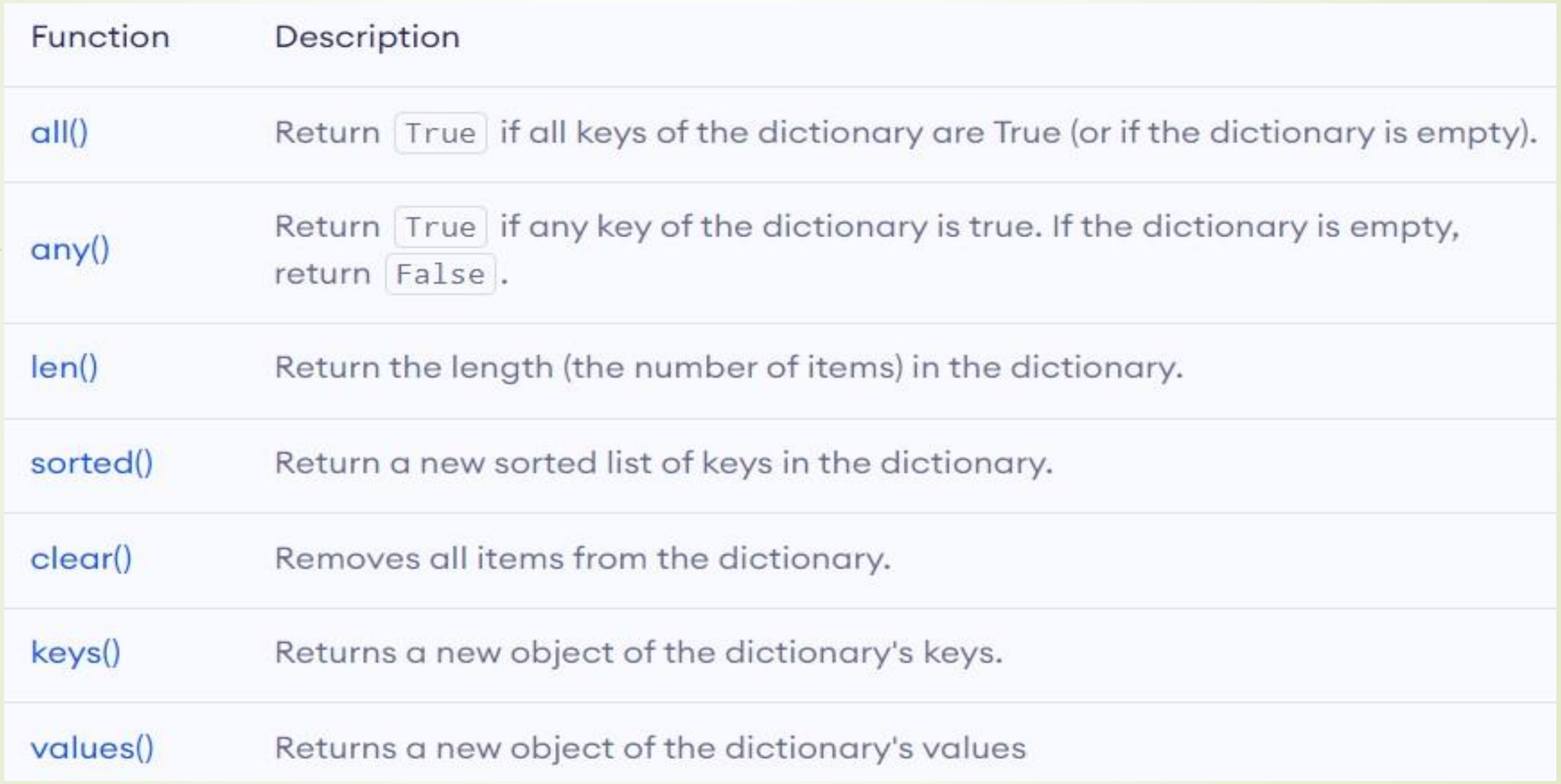


```
# Iterating through a Dictionary
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
    print(squares[i])
```



```
1
9
25
49
81
```

Python Dictionary Methods:



Function	Description
all()	Return `True` if all keys of the dictionary are True (or if the dictionary is empty).
any()	Return `True` if any key of the dictionary is true. If the dictionary is empty, return `False`.
len()	Return the length (the number of items) in the dictionary.
sorted()	Return a new sorted list of keys in the dictionary.
clear()	Removes all items from the dictionary.
keys()	Returns a new object of the dictionary's keys.
values()	Returns a new object of the dictionary's values

Thank You.