VNUHCM - University of Science

fit@hcmus

Fundamentals of Programming for Artificial Intelligence

Session 05
2D List
Tuple/Set/Dictionary

Instructors:
Dr. Lê Thanh Tùng
Dr. Nguyễn Tiến Huy

Content

1. 2D List
2. Tuple
3. Set
4. Dictionary

1. 2D List

Review

- Happy numbers are defined through a process in which you replace the number by the sum of the squares of its digits. This process is repeated until the number becomes 1 (which makes it a happy number) or it loops endlessly in a cycle that does not include 1 (which makes it an unhappy number).

- For example,

$$19 \longrightarrow 1^2 + 9^2 = 1 + 81 = 82$$

$$82 \longrightarrow 8^2 + 2^2 = 64 + 4 = 68$$

$$68 \longrightarrow 6^2 + 8^2 = 36 + 64 = 100$$

$$100 \longrightarrow 1^2 + 0^2 + 0^2 = 1 + 0 + 0 = 1$$

$$4 \longrightarrow 4^2 = 16$$

$$16 \longrightarrow 1^2 + 6^2 = 1 + 36 = 37$$

$$37 \longrightarrow 3^2 + 7^2 = 9 + 49 = 58$$

$$58 \longrightarrow 5^2 + 8^2 = 25 + 64 = 89$$

$$89 \longrightarrow 8^2 + 9^2 = 64 + 81 = 145$$

$$145 \longrightarrow 1^2 + 4^2 + 5^2 = 1 + 16 + 25 = 42$$

$$42 \longrightarrow 4^2 + 2^2 = 16 + 4 = 20$$

$$20 \longrightarrow 2^2 + 0^2 = 4 + 0 = 4$$

Review

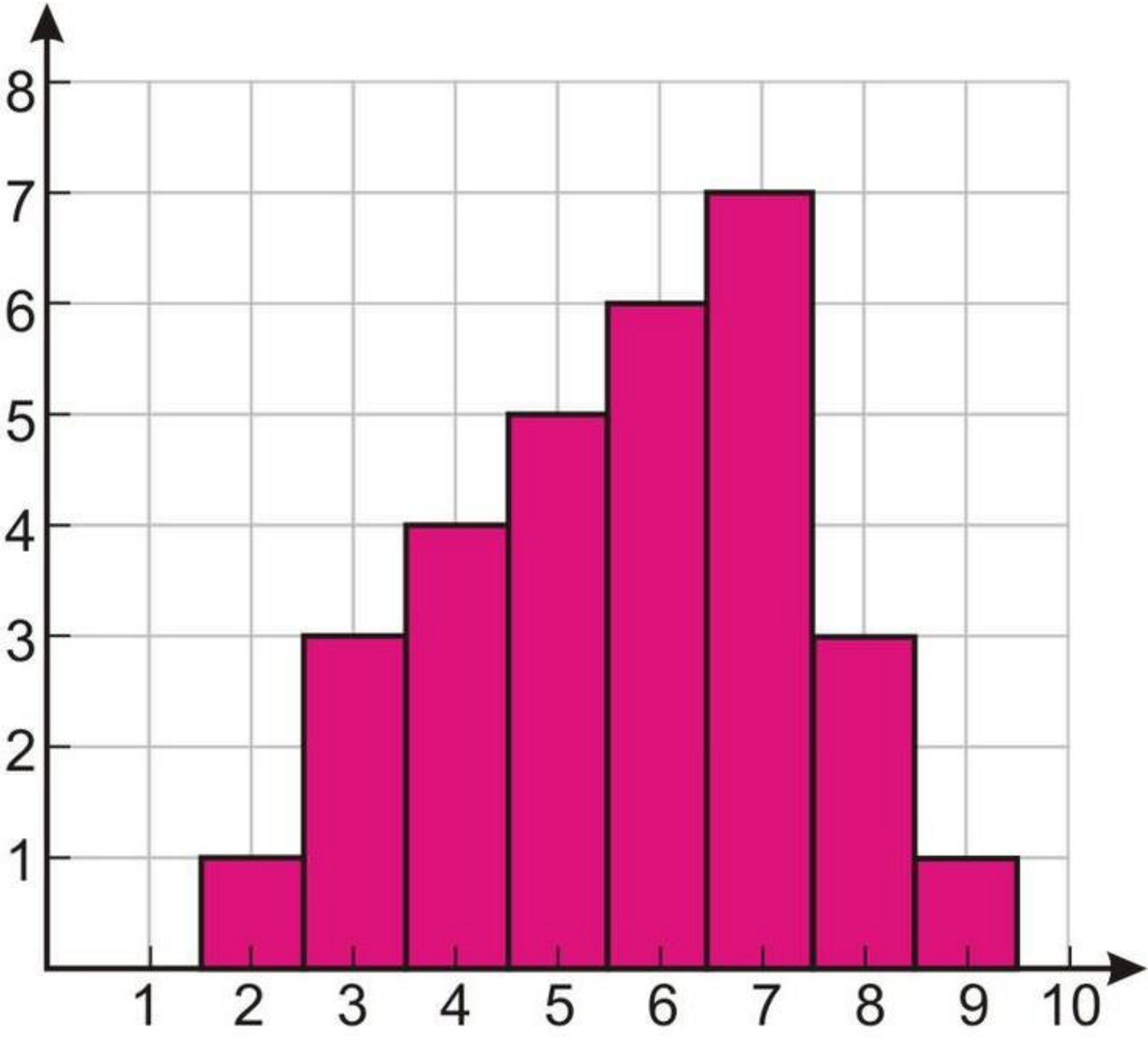
- Write a C++ program to perform a linear shift in a one-dimensional array. A linear shift is the process of rearranging the positions of elements in the array by a certain number of steps in either the left or right direction.
- **Input:**
  - {1, 2, 3, 4, 5}
  - Number of shift steps: 2
- **Output:** Array after the shift: {4, 5, 1, 2, 3}
- **Explanation:** The initial array is [1, 2, 3, 4, 5]. After shifting 2 steps to the right, we obtain a new array [4, 5, 1, 2, 3].

Review

- Consider A as a list of scores for n students, knowing that these scores are integers ranging from 0 to 100.
  - Determine the score that has the highest frequency of occurrence.
  - Determine the maximum number of consecutive students with scores above the average (≥ 50).

Review

- The freq array, also known as the frequency array, is used to track the frequency or count of each score in a given list of scores.
- Each **index** represents a **possible score**, and the corresponding **value** at that index represents the **frequency** or count of that score in the list



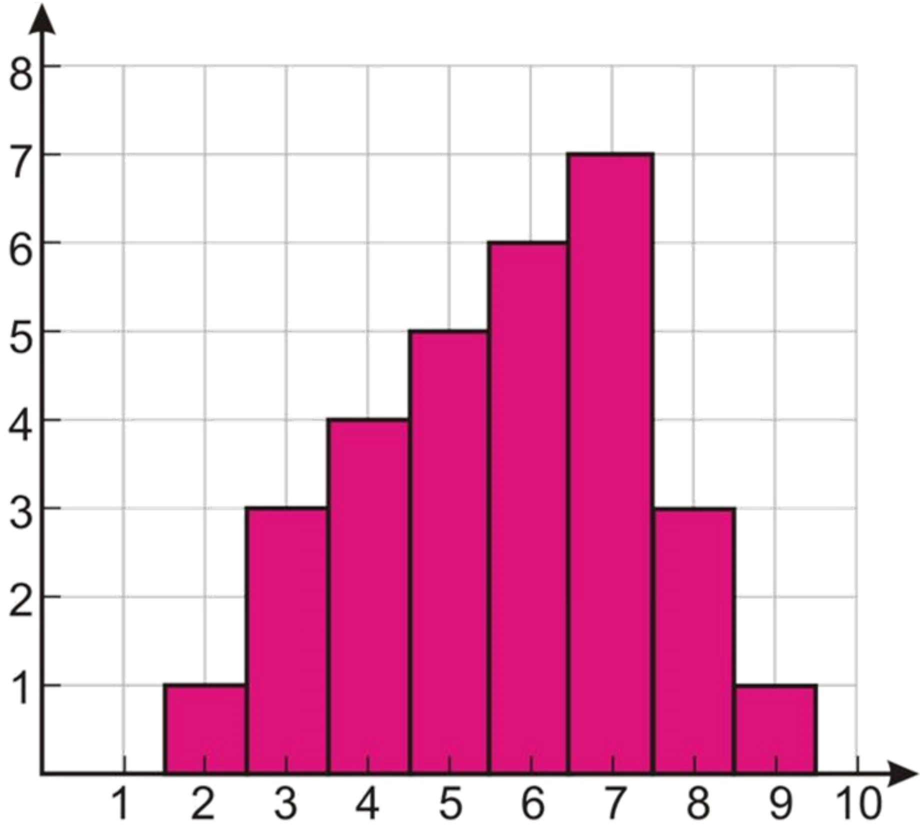
Review

```
1. Create an array freq[101] and initialize all elements to 0.
2. For each score in the list of scores:
   Increment freq[score] by 1
    for (int i = 0; i < nStudent; i++)
        freq[scores[i]] += 1

3. Find max frequency of score in the freq[]
    3.1: Set maxFreq = 0 and maxScore = -1.
    3.2: for (int i = 0; i < 101; i++)
            if (freq[i] > maxFreq){
                maxFreq = freq[i];
                maxScore = i;
            }

4. Return maxScore
```

Review

- Counting occurrences
- Limited range of values
- Efficient counting

- Application:
  - Identifying the maximum frequency
  - Tracking frequencies of different categories
  - Detecting duplicates

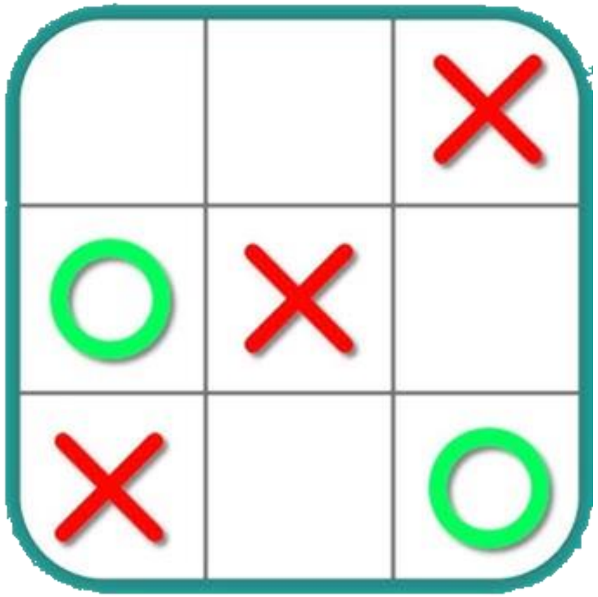


Review

- Consider A as a list of scores for n students, knowing that these scores are integers ranging from 0 to 100. Determine the maximum number of consecutive students with scores above 50 and print the last found consecutive sub-array

2D List

- A 2d list is a list that contains other lists as elements
  - lists of lists
- If we visualize the 2D list, we can think of each “sublist” as a row of a grid and each element of a “sublist” as a column in that row

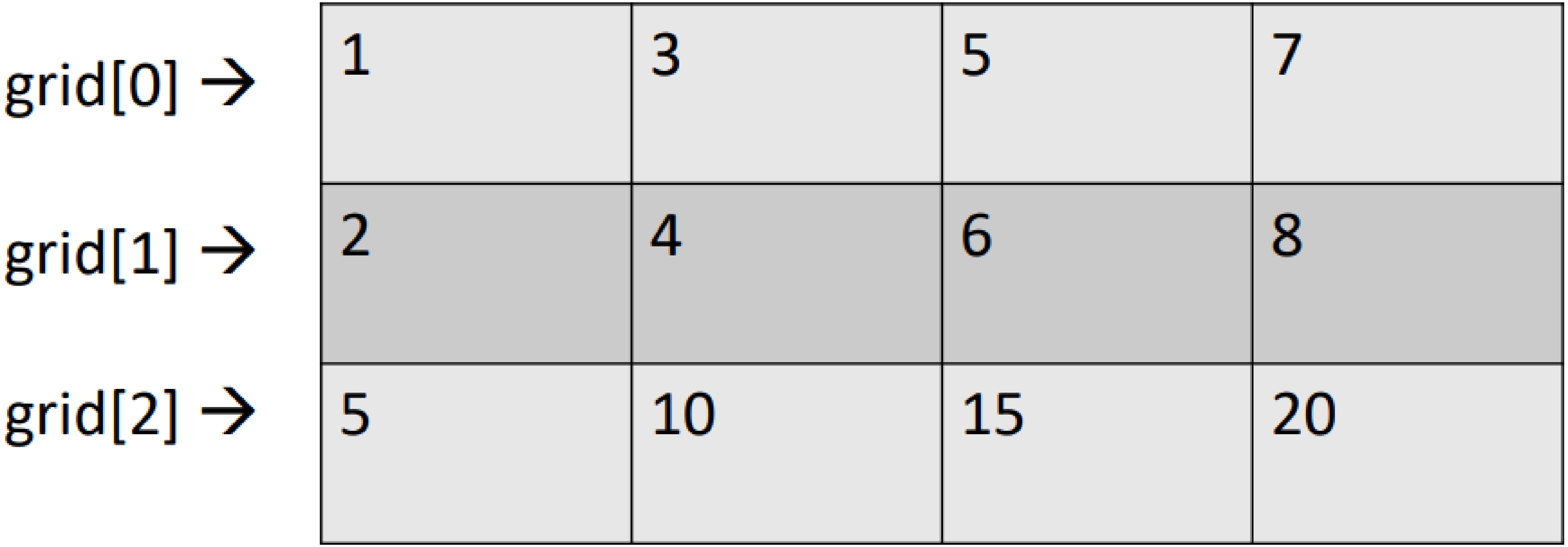


0	0	1
2	1	0
1	0	2

2D List

- Creating a matrix

```
grid = [[1, 3, 5, 7], [2, 4, 6, 8], [5, 10, 15, 20]]
```



grid[0] →	1	3	5	7
grid[1] →	2	4	6	8
grid[2] →	5	10	15	20

2D List

- Access element via its index

```
grid = [[1, 3, 5, 7], [2, 4, 6, 8], [5, 10, 15, 20]]
```

grid[0] →	1 grid[0][0]	3 grid[0][1]	5 grid[0][2]	7 grid[0][3]
grid[1] →	2 grid[1][0]	4 grid[1][1]	6 grid[1][2]	8 grid[1][3]
grid[2] →	5 grid[2][0]	10 grid[2][1]	15 grid[2][2]	20 grid[2][3]

2D List

- Using Slicing on 2D Lists

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10, 11, 12]
]
```

```
a = matrix[:1]
# [[1, 2, 3]]
b = matrix[::-1]
# [[10, 11, 12], [7, 8, 9], [4, 5, 6], [1, 2, 3]]
c = matrix[:][:-1]
# [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
d = matrix[:][::-1]
# [[10, 11, 12], [7, 8, 9], [4, 5, 6], [1, 2, 3]]
```

2D List

```
matrix = [
    [1, 2, 3],
    [4, 5, 6]
]
```

- Adding elements (a row) in 2D List

```
# Adding a row at the end of matrix
matrix.append([1, 1, 1])
# [[1, 2, 3], [4, 5, 6], [1, 1, 1]]
# Adding a row at the position pos
matrix.insert(1, [2, 2, 2])
# [[1, 2, 3], [2, 2, 2], [4, 5, 6], [1, 1, 1]]
# Add a matrix into matrix
matrix2 = [[3, 3, 3], [4, 4, 4]]
matrix.extend((matrix2))
```

2D List

- Adding elements (a column) in 2D List

```
matrix = [
    [1, 2, 3],
    [4, 5, 6]
]
for row in matrix:
    row.append(0)
print(matrix)
```

2D List

- Remove elements (a row) in 2D List

```
# del statement
del matrix[0]
# [[4, 5, 6], [7, 8, 9], [10, 11, 12]]

# remove the last row
matrix.pop()
# [[4, 5, 6], [7, 8, 9]]

# remove the i-th row
matrix.pop(1)
# [[4, 5, 6]]
```

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10, 11, 12]
]
```

2D List

- Iterating through a 2D List

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
]

for row in matrix:
    for element in row:
        print(element, end=" ")
    print()  # For a new line after each row
```

2D List

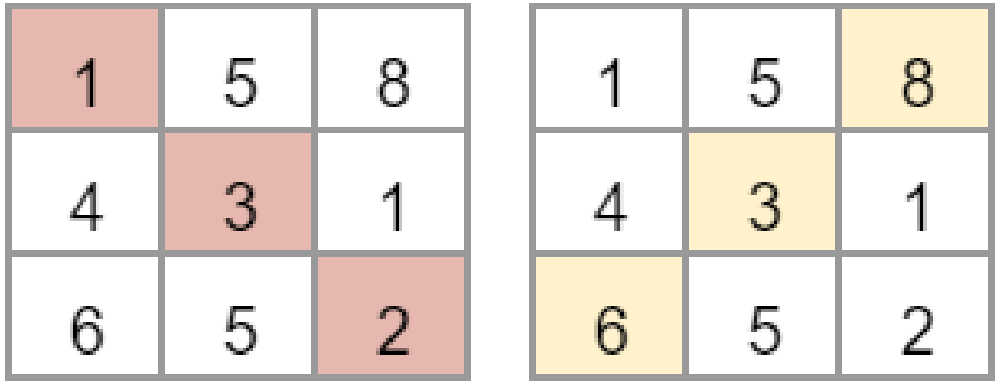
- Iterating through a 2D List

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
]

for row in matrix:
    for element in row:
        print(element, end=" ")
    print()  # For a new line after each row
```

Diagonal

- Main diagonal
  - `matrix[i][i]`

- Opposite diagonal
  - `matrix[i][Size - i - 1]`

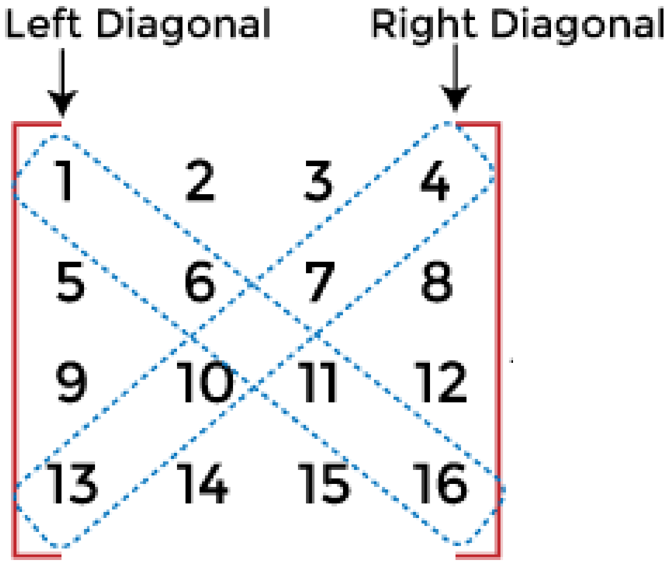


1	5	8
4	3	1
6	5	2

Principal Diagonal

1	5	8
4	3	1
6	5	2

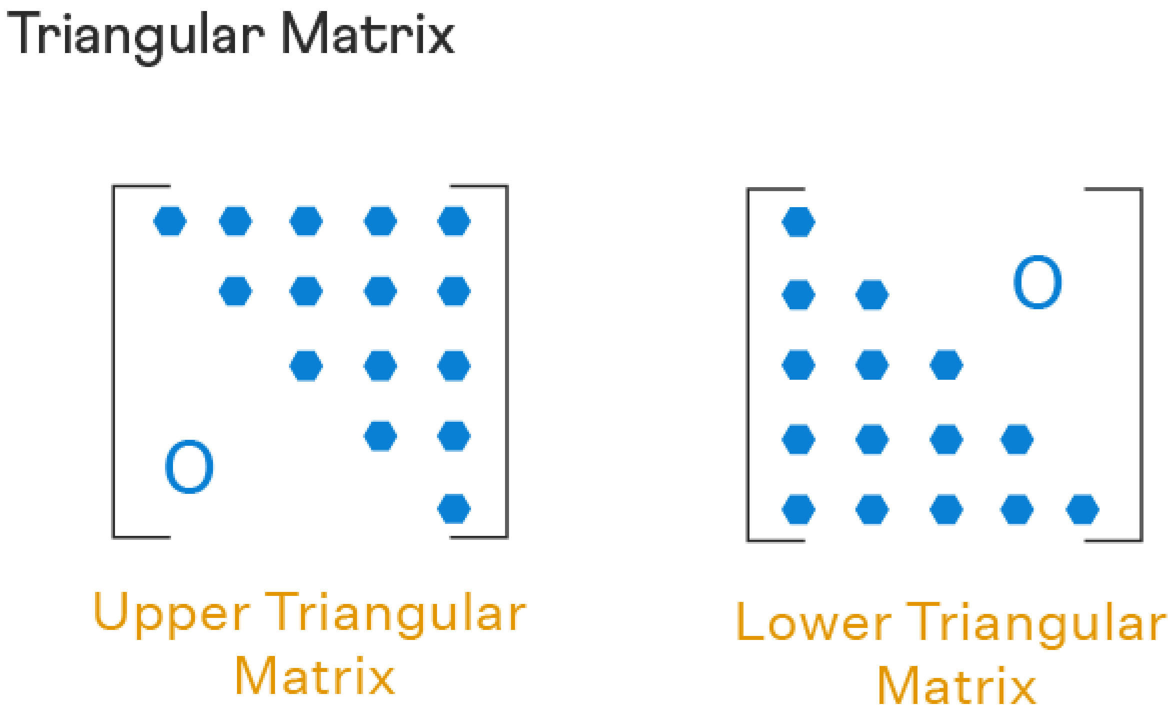
Anti-Diagonal



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Exercises

- Print the values in the main diagonal of a matrix `N x N`.
- Calculate the sum of Lower Triangular Matrix



Transpose a 2D Array

- The transpose of a matrix is a new matrix whose rows are the columns of the original.
  - This makes the columns of the new matrix the rows of the original.
  - The element at row `r` column `c` in the original is placed at row `c` column `r` of the transpose. The element `a[r][c]` of the original matrix becomes element `a[c][r]` in the transposed matrix.

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}_{2 \times 3} \qquad A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}_{3 \times 2}$$

Exercise

- Write a function to transpose the matrix (A, n, m) into the matrix (B, m, n)

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}_{2 \times 3}$$

$$A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}_{3 \times 2}$$

Exercise

- Write a function named Upper-half which takes a two-dimensional array A, with size N rows and N columns as argument and prints the upper half of the array.

```
2 3 1 5 0                      2 3 1 5 0
7 1 5 3 1                        1 5 3 1
2 5 7 8 1  Output will be:         7 8 1
0 1 5 0 1                            0 1
3 4 9 1 5                              5
```

Exercise

- Write a function which accepts a 2D array of integers and its size as arguments and displays the elements of middle row and the elements of middle column. Assuming the 2D Array to be a square matrix with odd dimension i.e. 3x3, 5x5, 7x7 etc...
- Example, if the array contents is

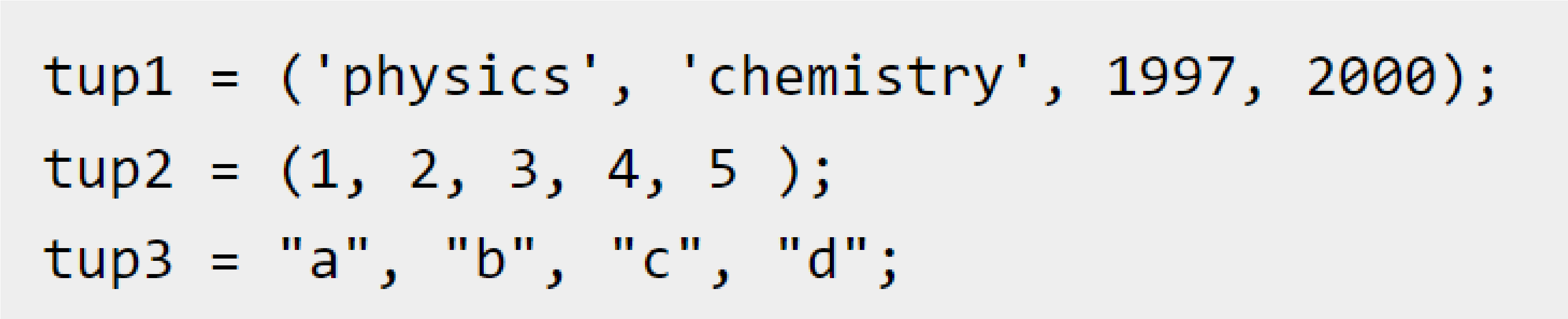
3	5	4
7	6	9
2	1	8

- Output through the function should be :
  - Middle Row : 7 6 9
  - Middle column : 5 6 1

2. Tuple

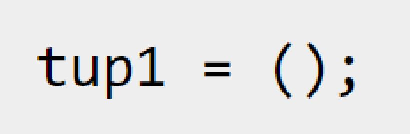
Tuple

- A tuple is a collection which is ordered and **unchangeable**
- Declare and create a tuple



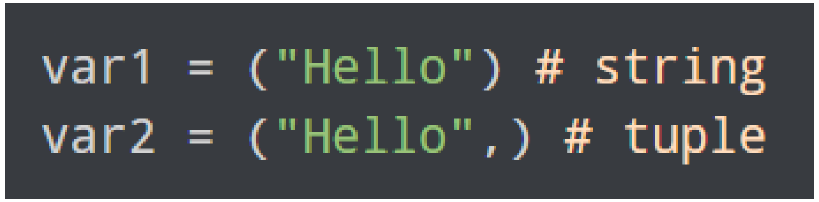
```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

- Empty tuple:



```
tup1 = ();
```

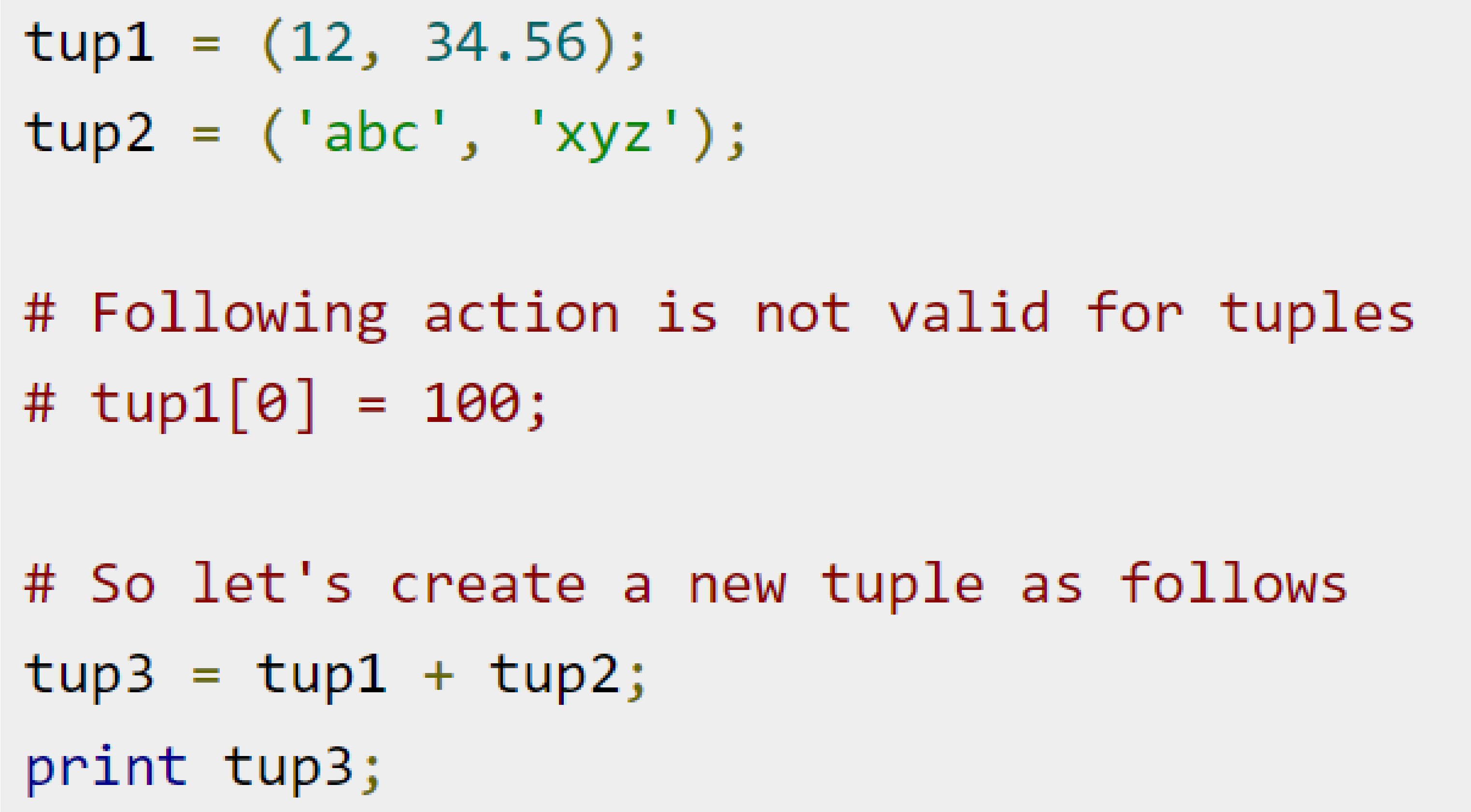
- Tuple with one element:



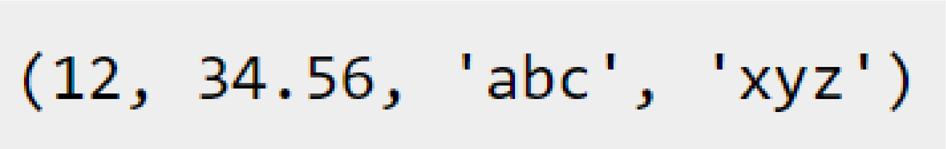
```
var1 = ("Hello") # string
var2 = ("Hello",) # tuple
```

Updating Tuples

- Tuples are immutable which means you cannot update or change the values of tuple elements



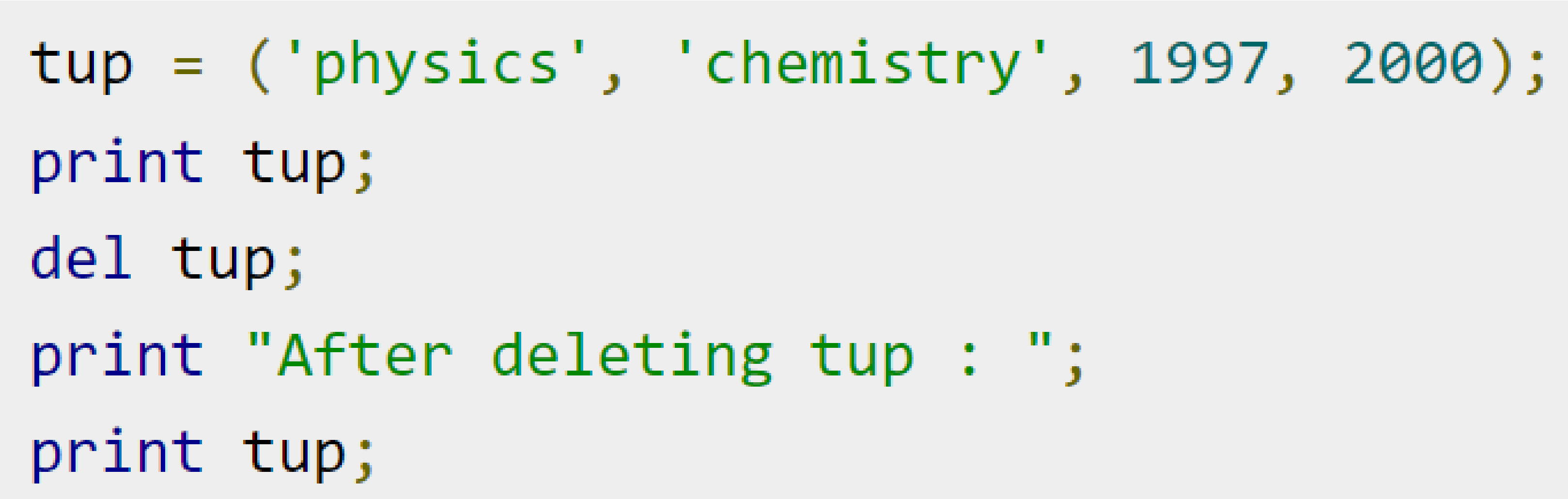
```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3;
```



```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

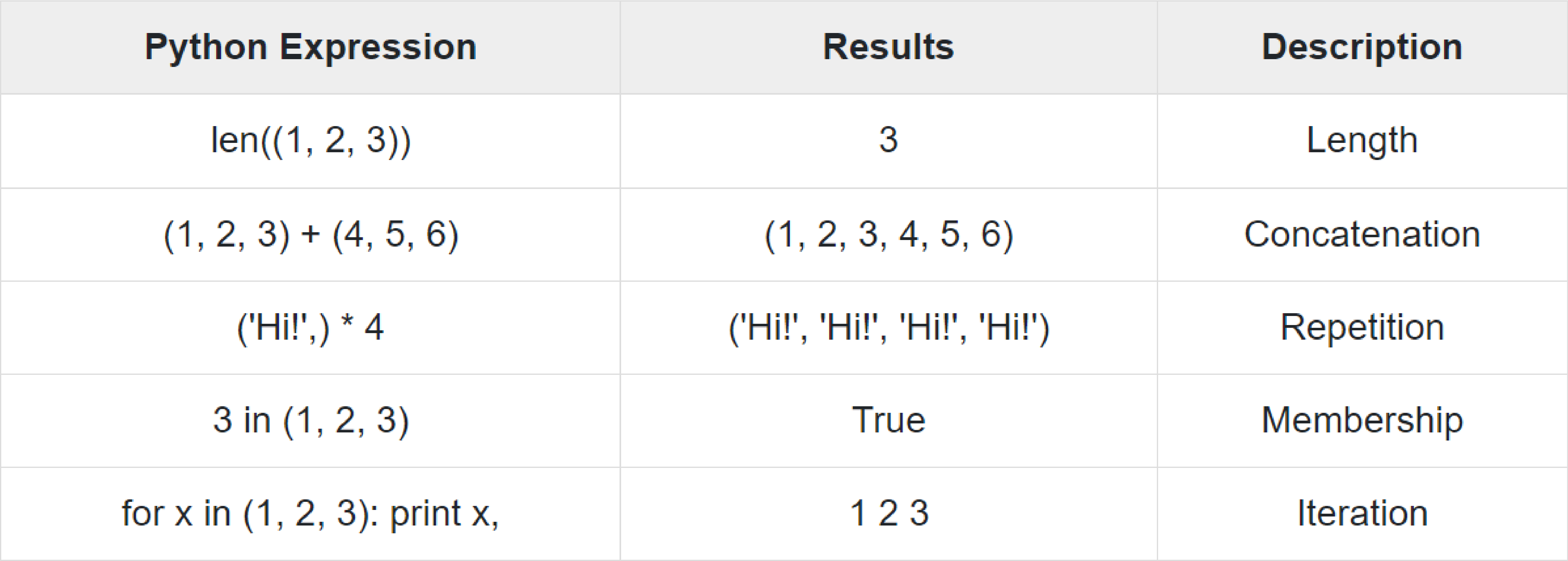
- Removing individual tuple elements is not possible
  - Removing by putting together another tuple with the undesired elements discarded is possible
  - Using **del** statement



```
tup = ('physics', 'chemistry', 1997, 2000);
print tup;
del tup;
print "After deleting tup : ";
print tup;
```

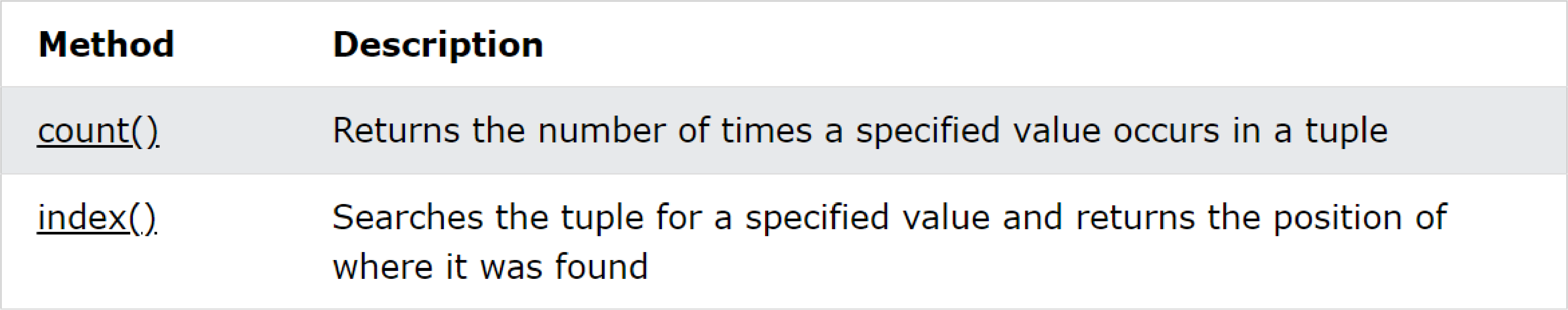
- Note an exception raised, this is because after **del tup** tuple does not exist any more

Basic Tuples Operations



Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Basic Tuples Methods

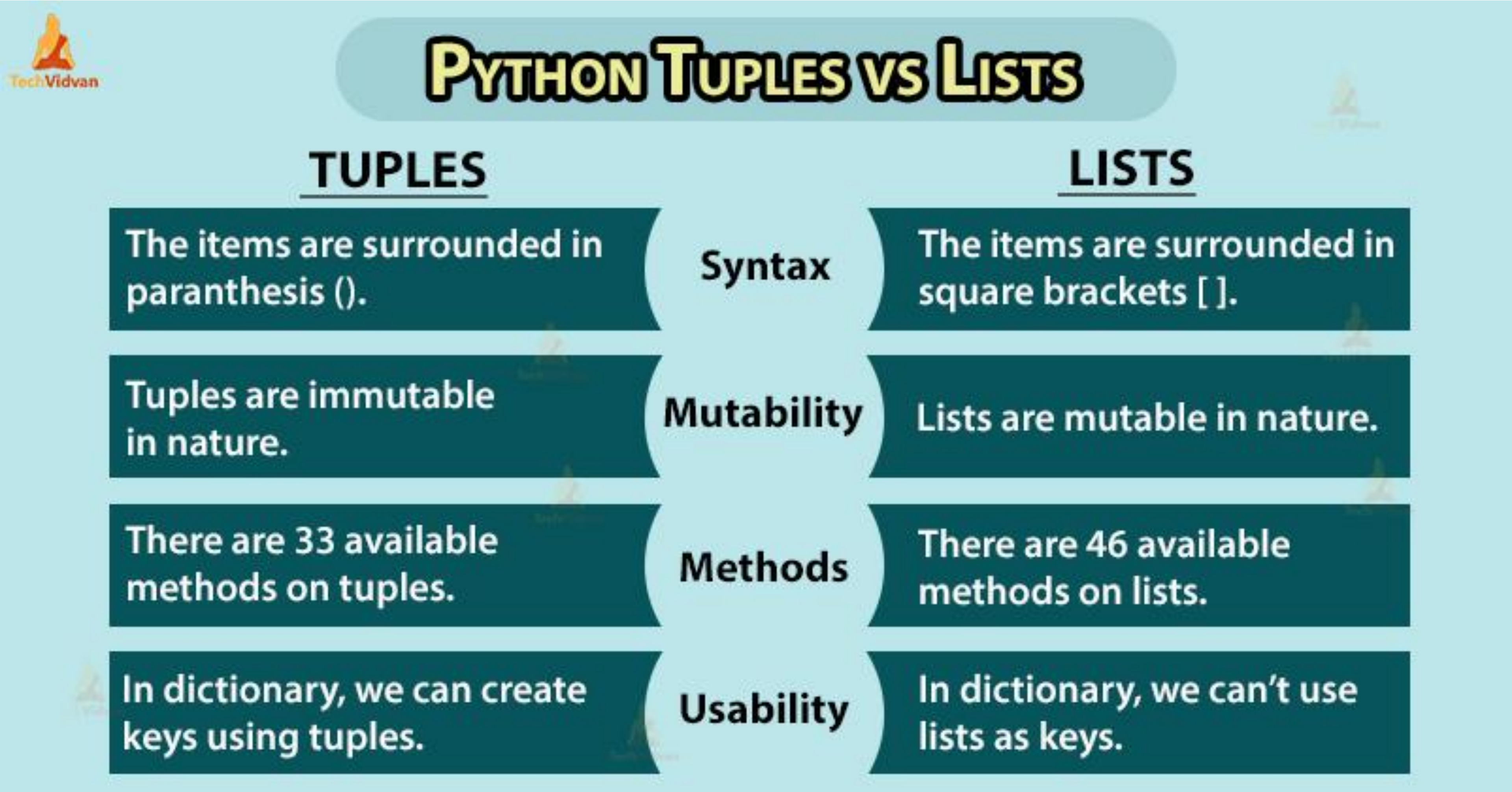


Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Attributes of Tuple item

- Ordered
  - When we say that tuples are ordered, it means that the items have a defined order, and that order will not change
- Unchangeable
  - Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created
- Allow Duplicates
  - Since tuples are indexed, they can have items with the same value

List Vs Tuple

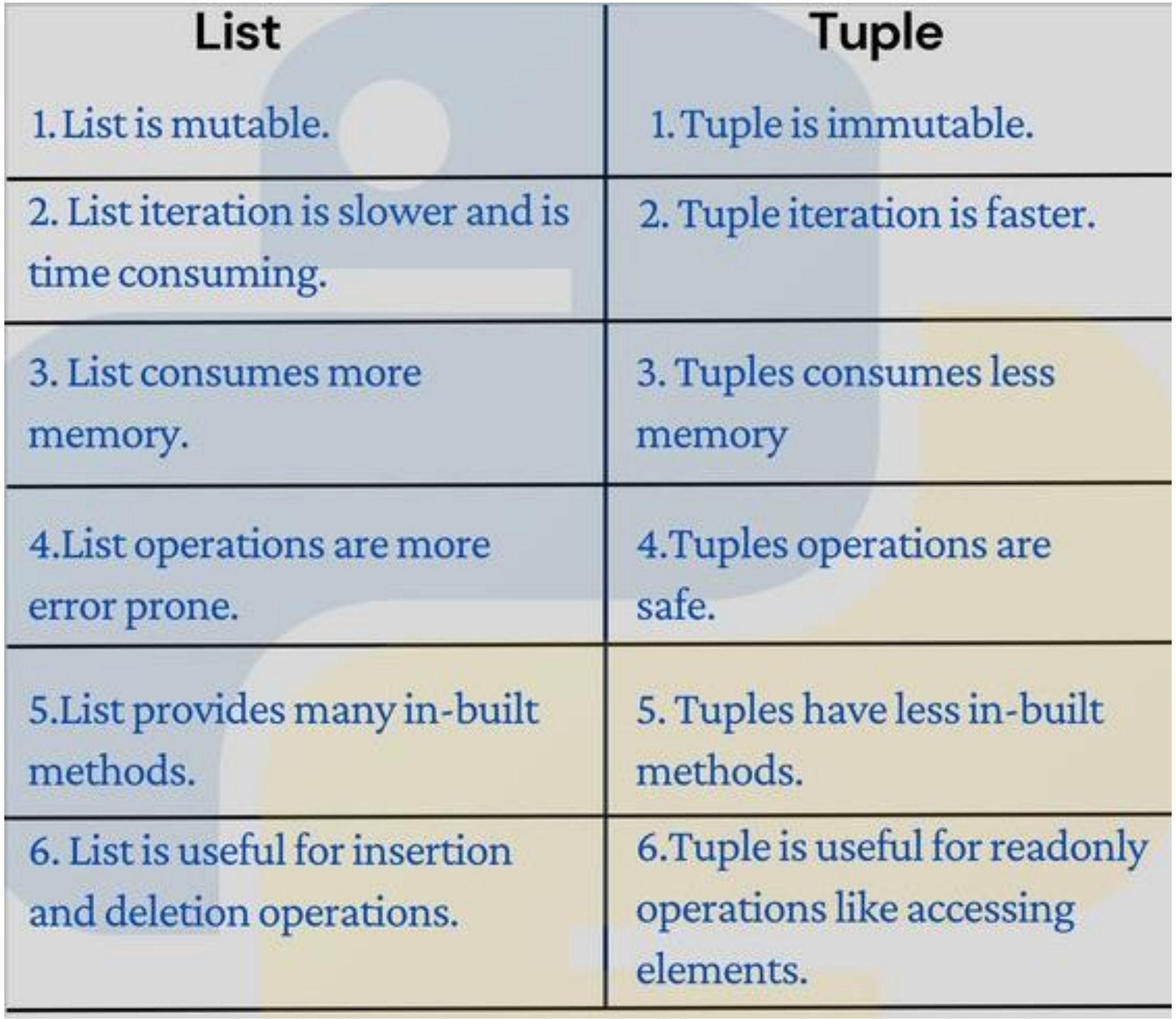


TechVidvan

PYTHON TUPLES VS LISTS

TUPLES		LISTS
The items are surrounded in paranthesis ().	Syntax	The items are surrounded in square brackets [ ].
Tuples are immutable in nature.	Mutability	Lists are mutable in nature.
There are 33 available methods on tuples.	Methods	There are 46 available methods on lists.
In dictionary, we can create keys using tuples.	Usability	In dictionary, we can't use lists as keys.

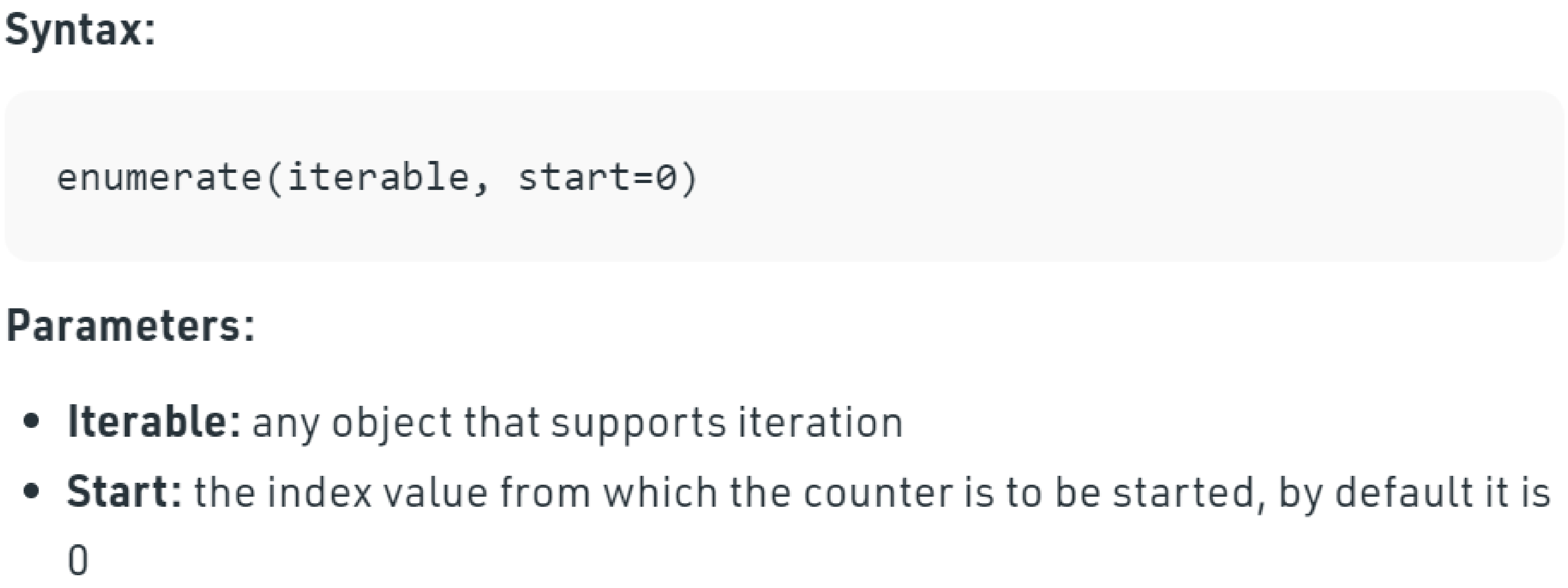
List Vs Tuple



List	Tuple
1. List is mutable.	1. Tuple is immutable.
2. List iteration is slower and is time consuming.	2. Tuple iteration is faster.
3. List consumes more memory.	3. Tuples consumes less memory
4.List operations are more error prone.	4.Tuples operations are safe.
5.List provides many in-built methods.	5. Tuples have less in-built methods.
6. List is useful for insertion and deletion operations.	6.Tuple is useful for readonly operations like accessing elements.

enumerate

- Enumerate() method adds a counter to an iterable and returns it in a form of enumerating object



Syntax:

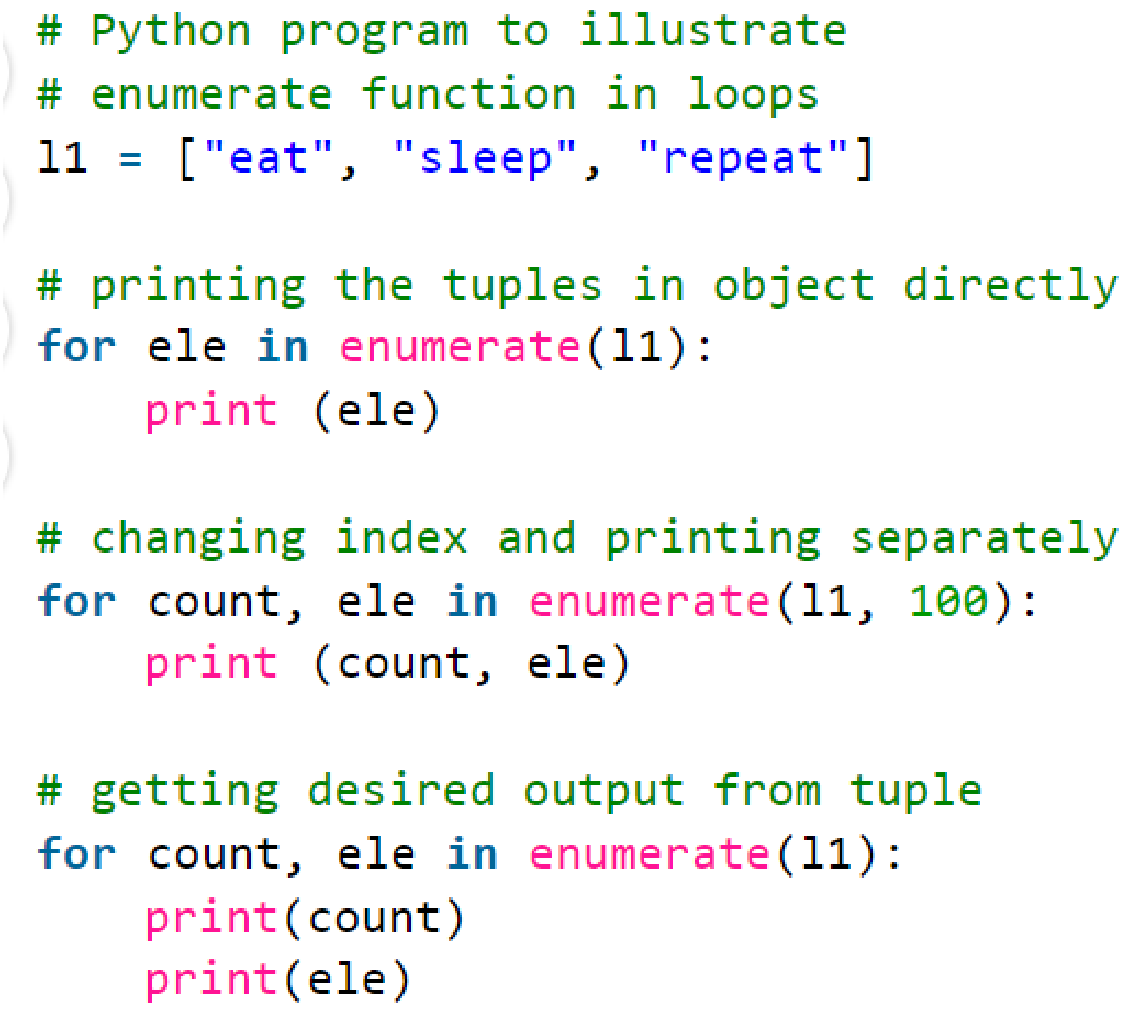
```
enumerate(iterable, start=0)
```

Parameters:

- **Iterable:** any object that supports iteration
- **Start:** the index value from which the counter is to be started, by default it is 0

enumerate

- Using it in loop to control the index and value



```
# Python program to illustrate
# enumerate function in loops
l1 = ["eat", "sleep", "repeat"]

# printing the tuples in object directly
for ele in enumerate(l1):
    print (ele)

# changing index and printing separately
for count, ele in enumerate(l1, 100):
    print (count, ele)

# getting desired output from tuple
for count, ele in enumerate(l1):
    print(count)
    print(ele)
```

Exercise

- Write a function to find all repeated elements in a tuple t. Return value is a list of repeated elements

Exercise

- Write a function to find a second maximum value of an integer tuple. Return value type is int

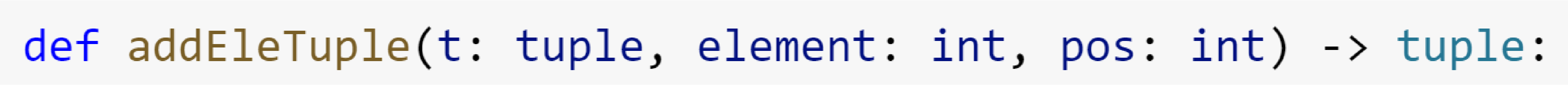
Exercise

- Write a function to to count how many prime numbers are in a given tuple t in Python

Exercise

- Write a function to add an integer n into an integer tuple at the position p
  - For example, we have the tuple t = (1, 2, 3, 4, 5)
  - And put n = 7 at position 3, the result is: t = (1, 2, 3, 7, 4, 5)

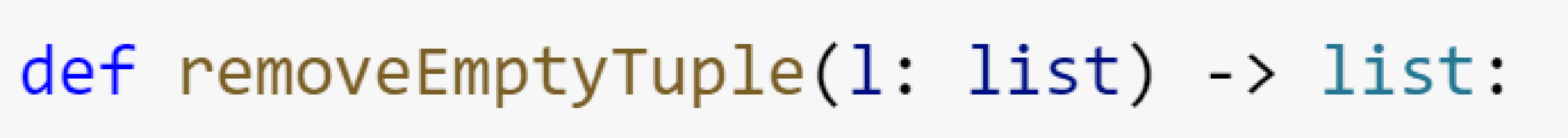
  The prototype of this function should be:



```python
def addEleTuple(t: tuple, element: int, pos: int) -> tuple:
```

Exercise

- Write a Python program to remove an empty tuple(s) from a list of tuples



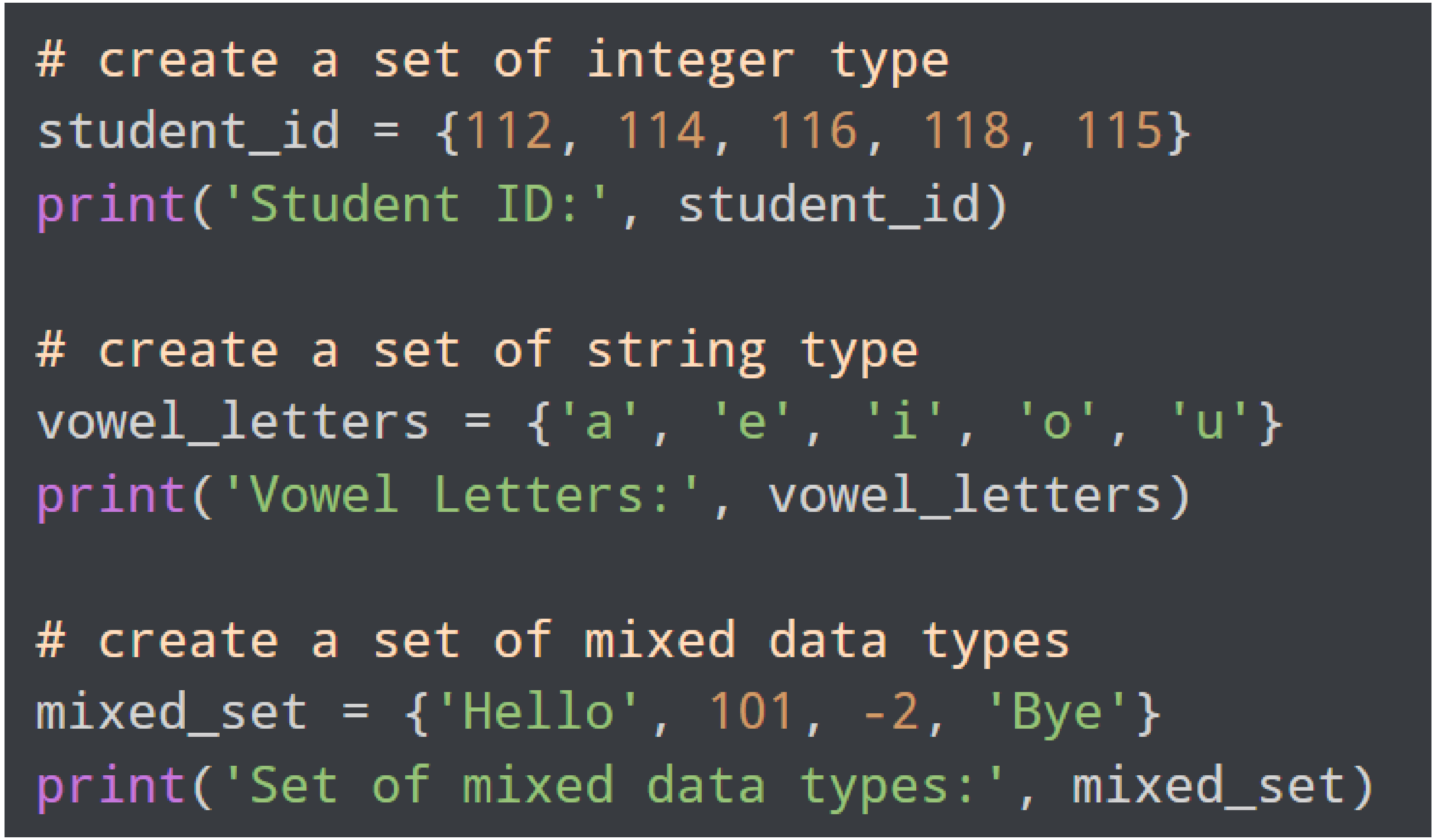
```
def removeEmptyTuple(l: list) -> list:
```

- With the input: [(), (), ('',), ('a', 'b'), ('a', 'b', 'c'), ('d’)]
- The expected output: [('',), ('a', 'b'), ('a', 'b', 'c'), 'd']

3. Set

Set

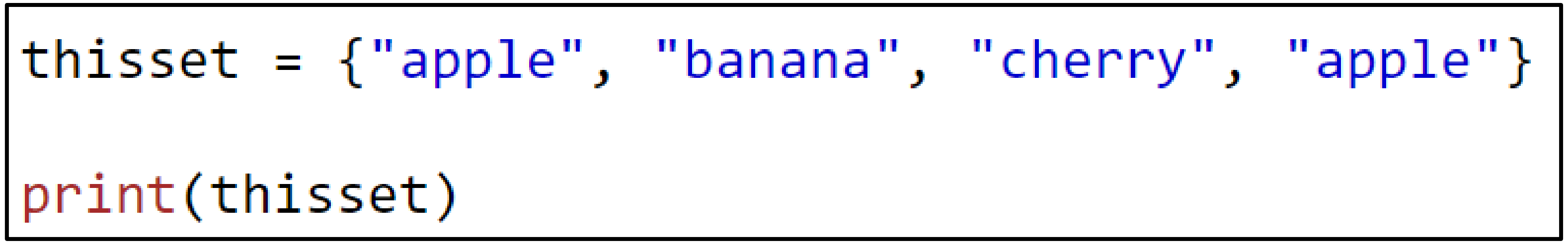
- A set is a collection which is unordered, unchangeable*, and unindexed
- Set items are unchangeable, but you can remove items and add new items



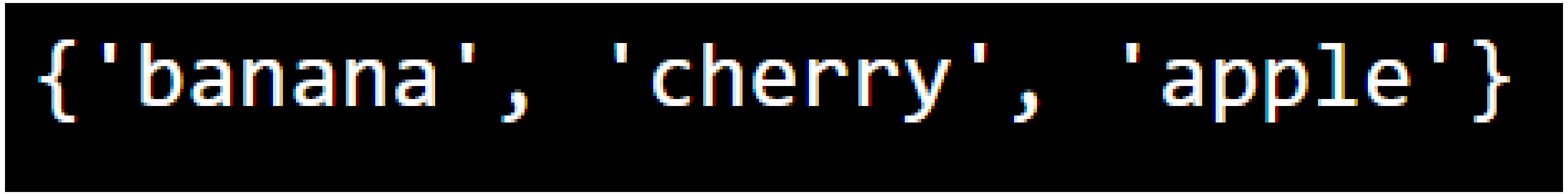
```
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)

# 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)
```

Set

- A set is a collection of unique data. Duplicate values will be ignored



```python
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

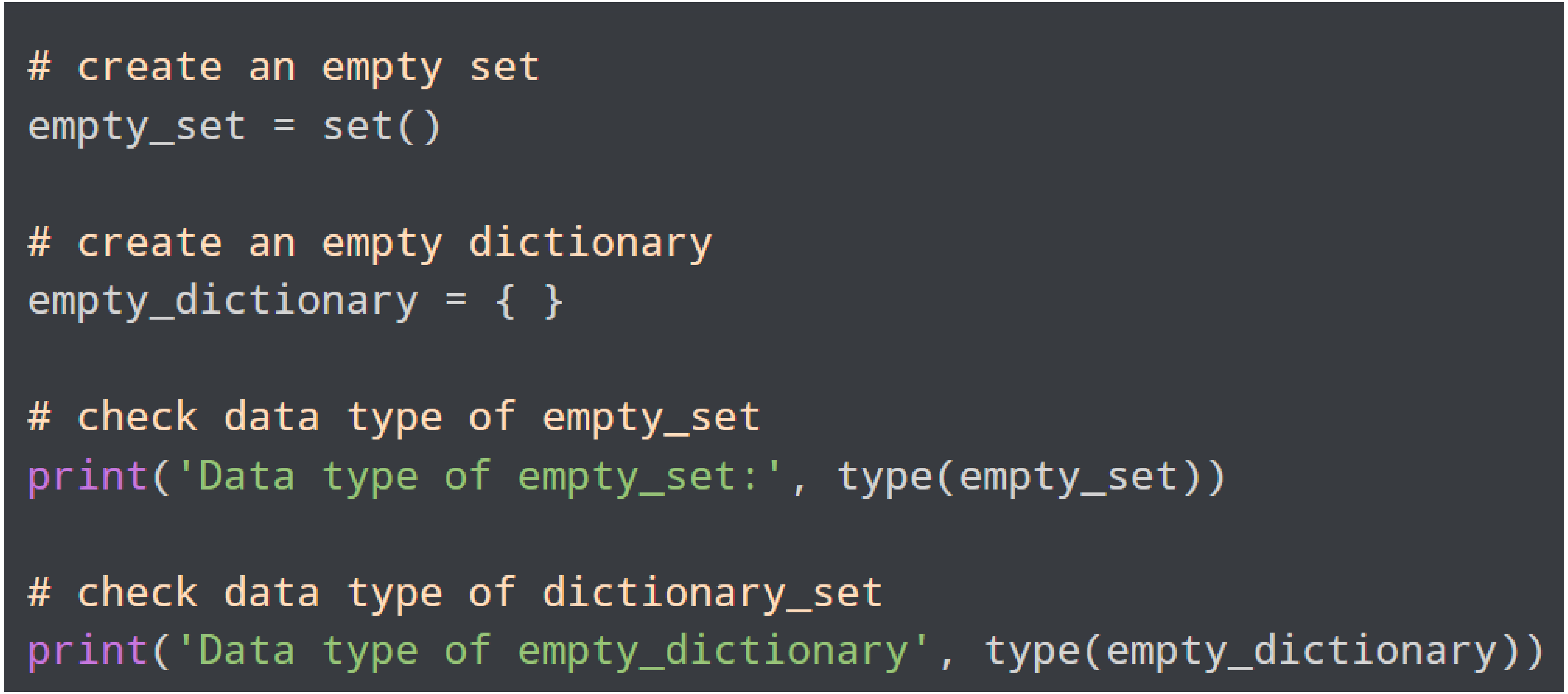


```
{'banana', 'cherry', 'apple'}
```

- Suppose we want to store information about student IDs. Since student IDs cannot be duplicate, we can use a set

Empty Set

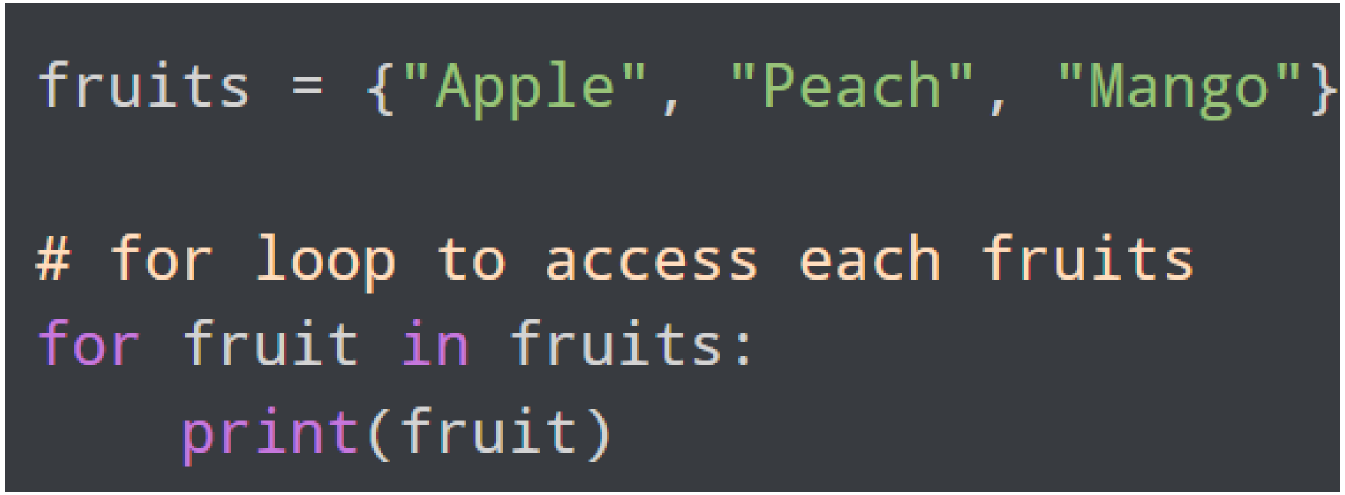
- To make a set without any elements, we use the set() function without any argument



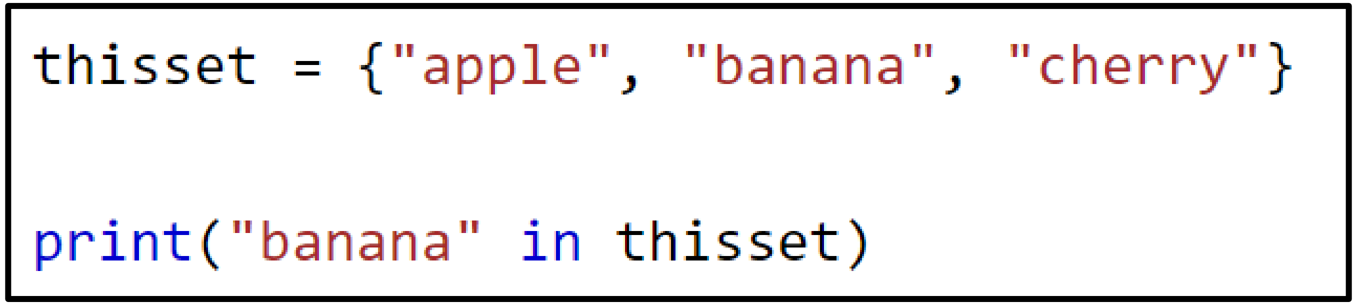
```
# create an empty set
empty_set = set()

# 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))
```

Access Items

- You cannot access items in a set by referring to an index or a key
- Using **for-loop** and **in**



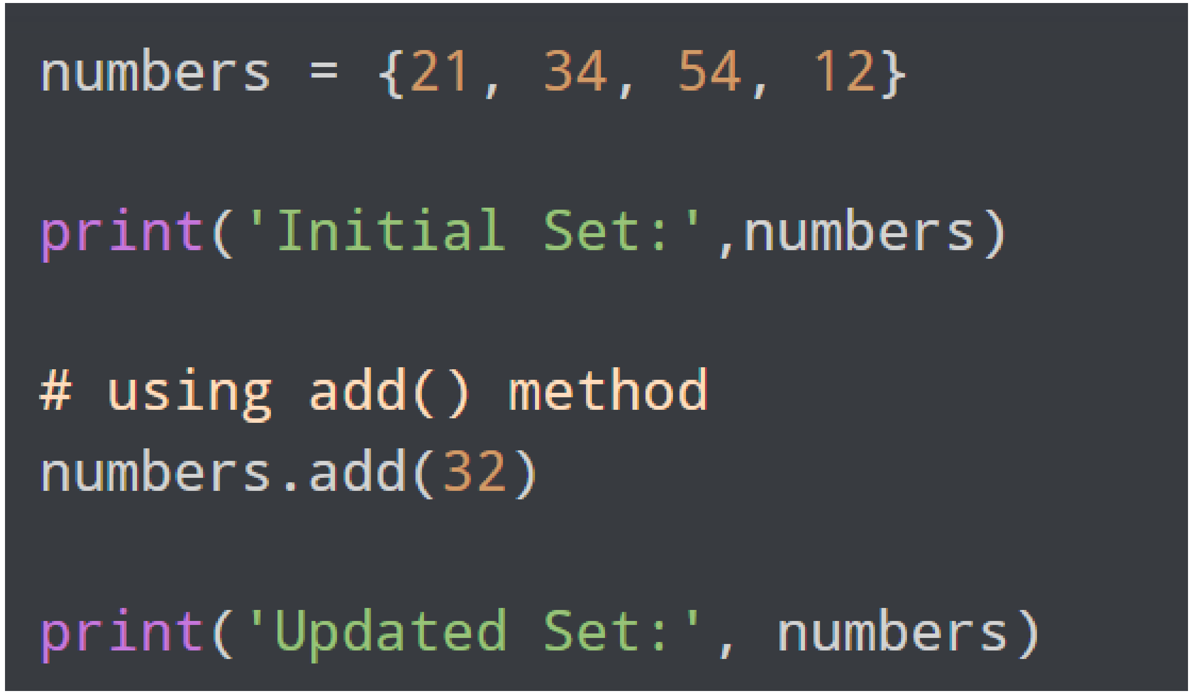
```python
fruits = {"Apple", "Peach", "Mango"}

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



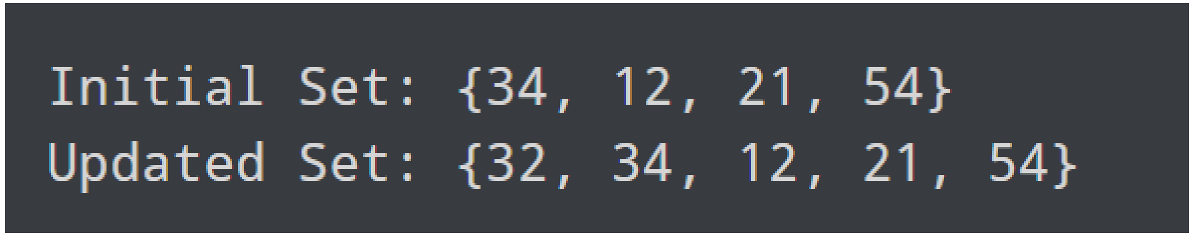
```python
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Add items

- Using **add()** method



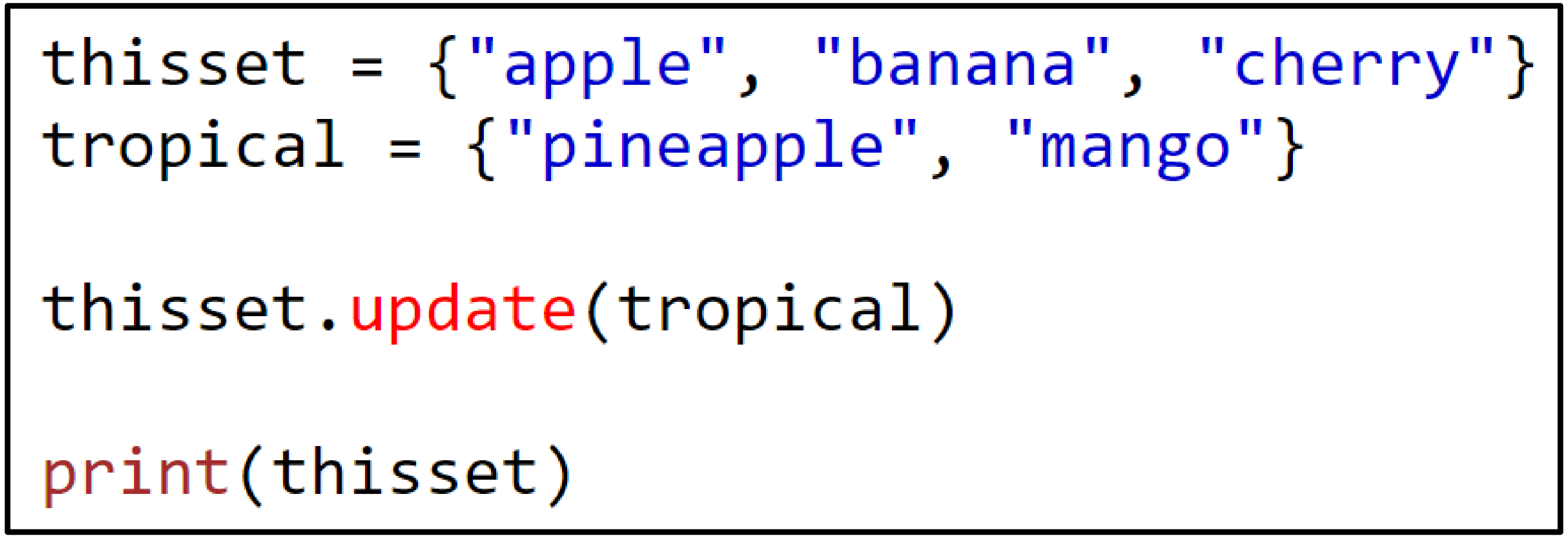
```python
numbers = {21, 34, 54, 12}

print('Initial Set:',numbers)

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

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



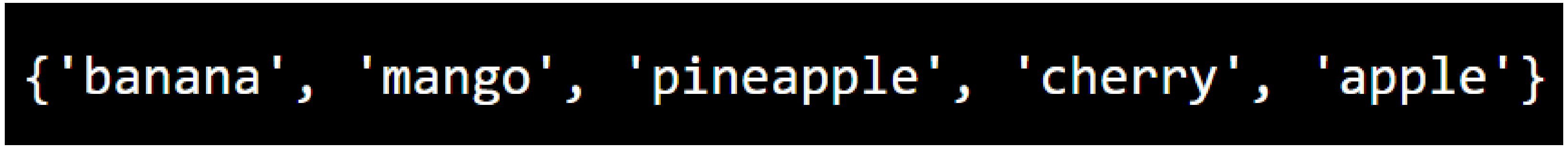
```
Initial Set: {34, 12, 21, 54}
Updated Set: {32, 34, 12, 21, 54}
```

Add items

- To add items from another set into the current, use the **update()** method



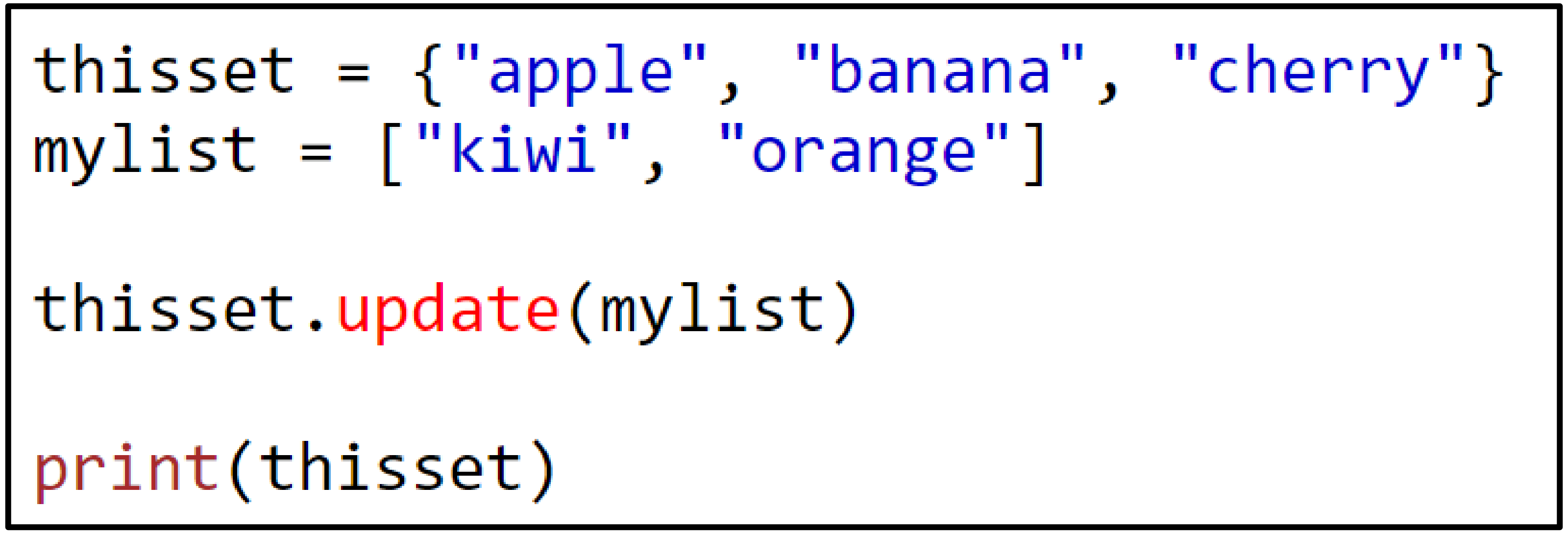
```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango"}

thisset.update(tropical)

print(thisset)
```



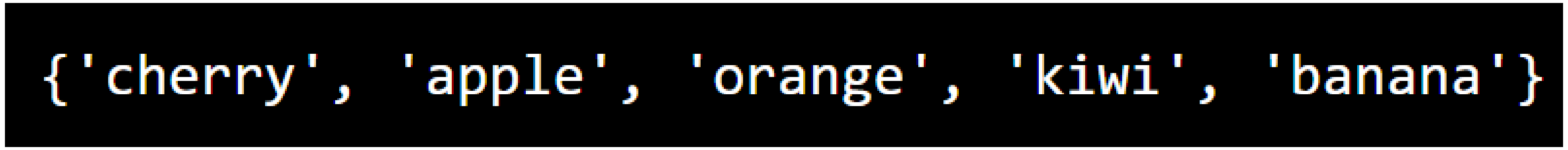
```
{'banana', 'mango', 'pineapple', 'cherry', 'apple'}
```

Add items

- the **update()** method can be used for any iterable object (tuples, lists, dictionaries etc.)



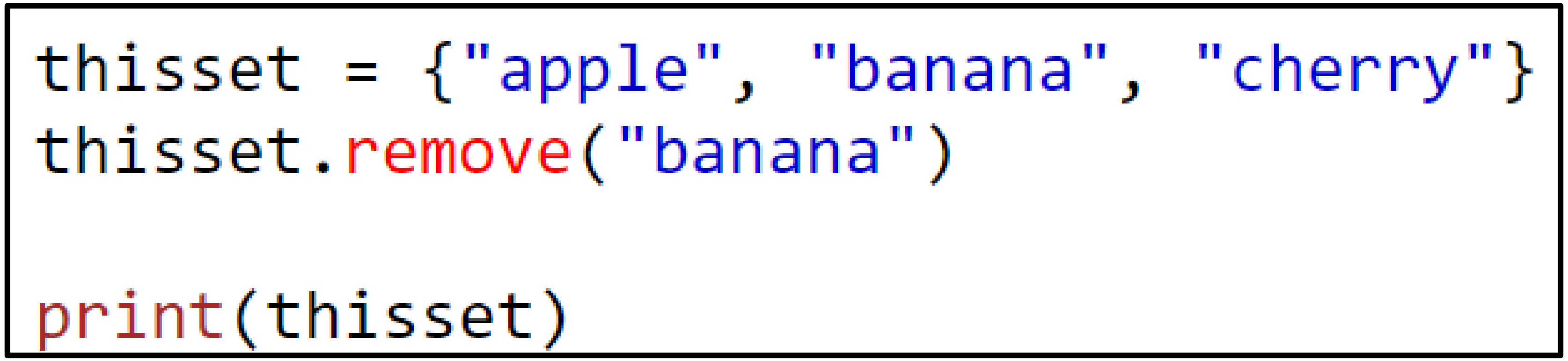
```
thisset = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)
```



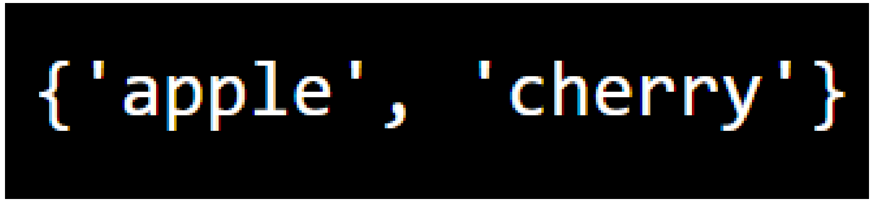
```
{'cherry', 'apple', 'orange', 'kiwi', 'banana'}
```

Remove item

- To remove an item in a set, use the **remove()**, or the **discard()** method



```python
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")

print(thisset)
```

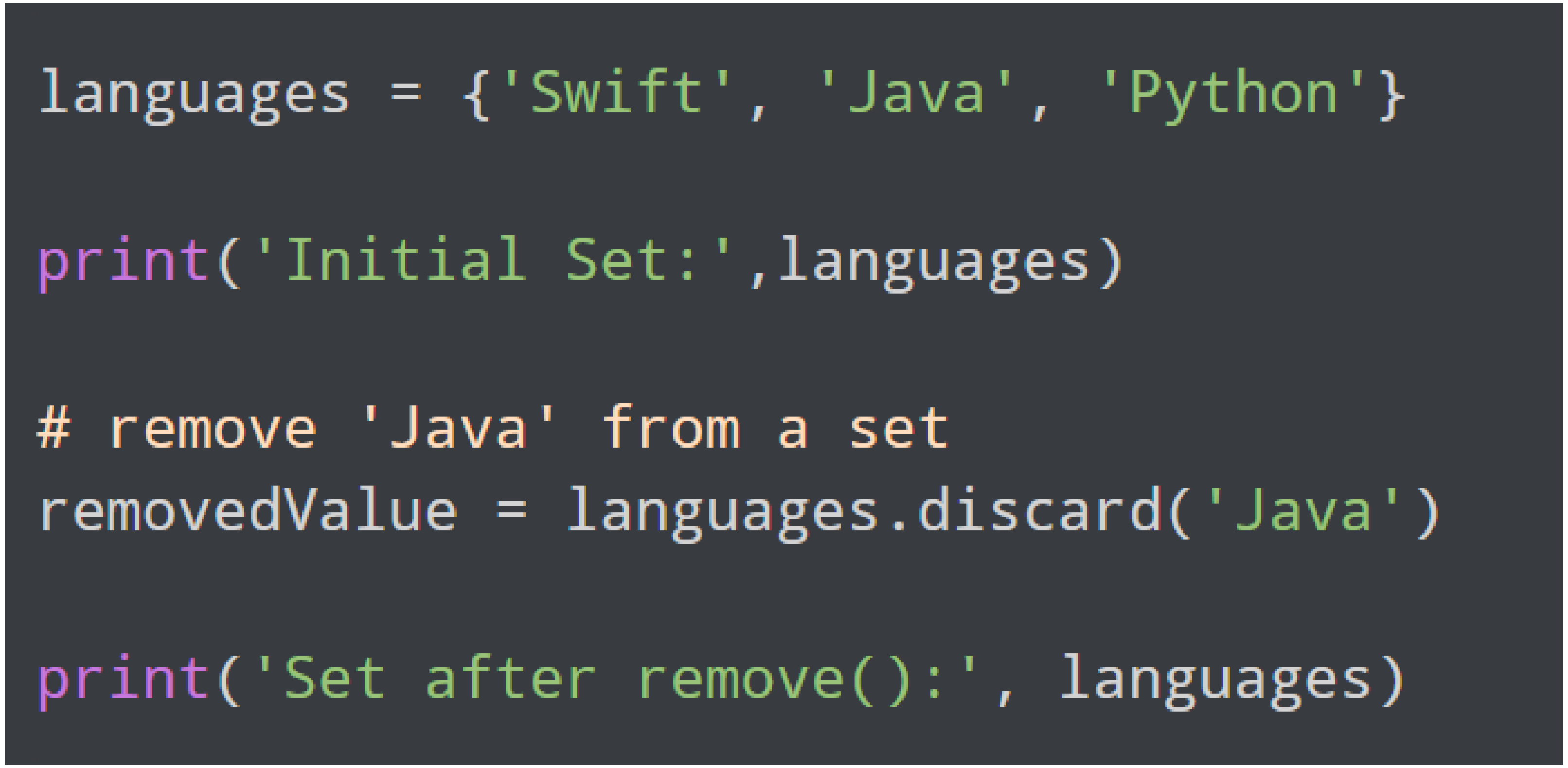


```
{'apple', 'cherry'}
```

- Note: If the item to remove does not exist, **remove()** will raise an error

Remove item

- To remove an item in a set, use the **remove()**, or the **discard()** method

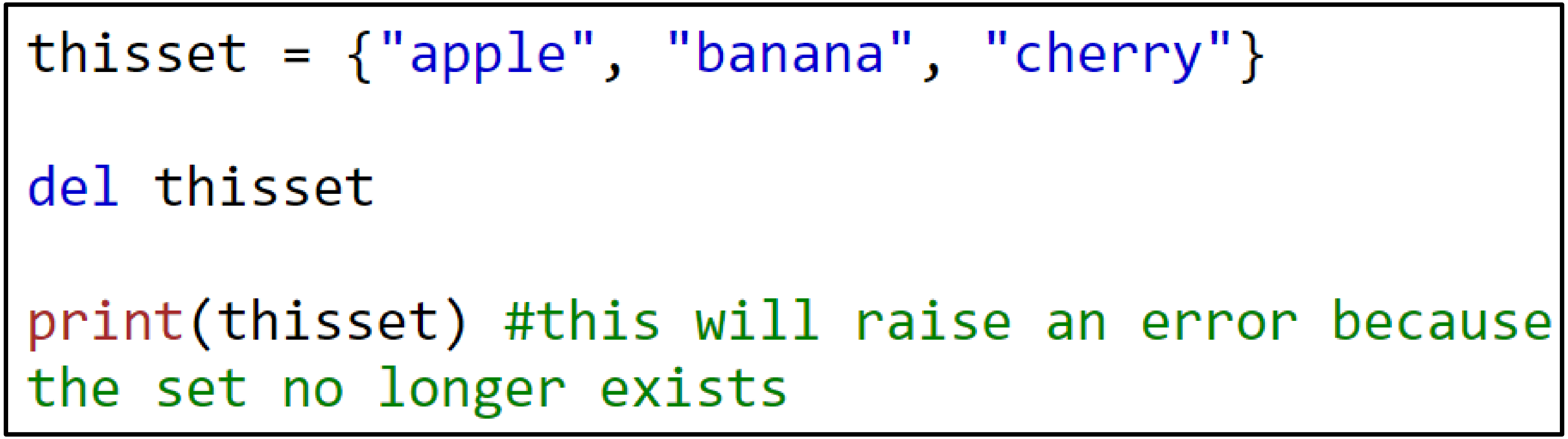


```python
languages = {'Swift', 'Java', 'Python'}

print('Initial Set:',languages)

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

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

- Note: If the item to remove does not exist, **discard()** will **NOT** raise an error.

Remove item

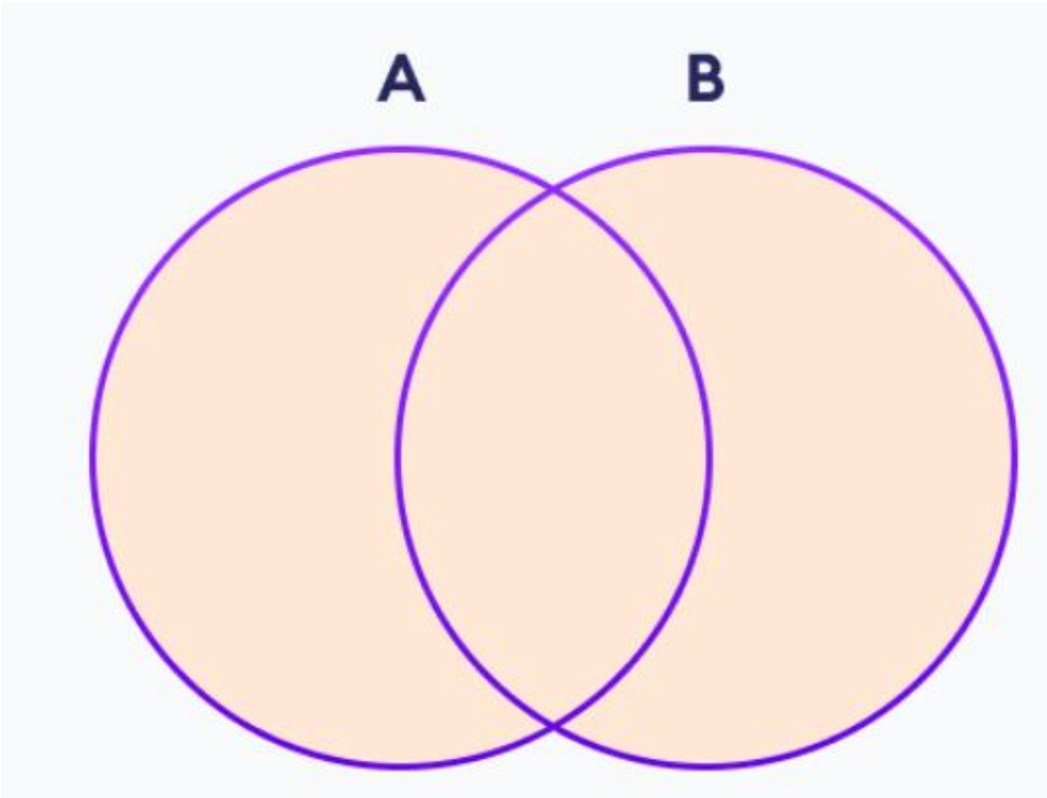
- pop() will remove a random item because set is unordered
- clear() will remove all items and return an empty set
- The del keyword will delete the set completely and can not use it to remove the specified element in set

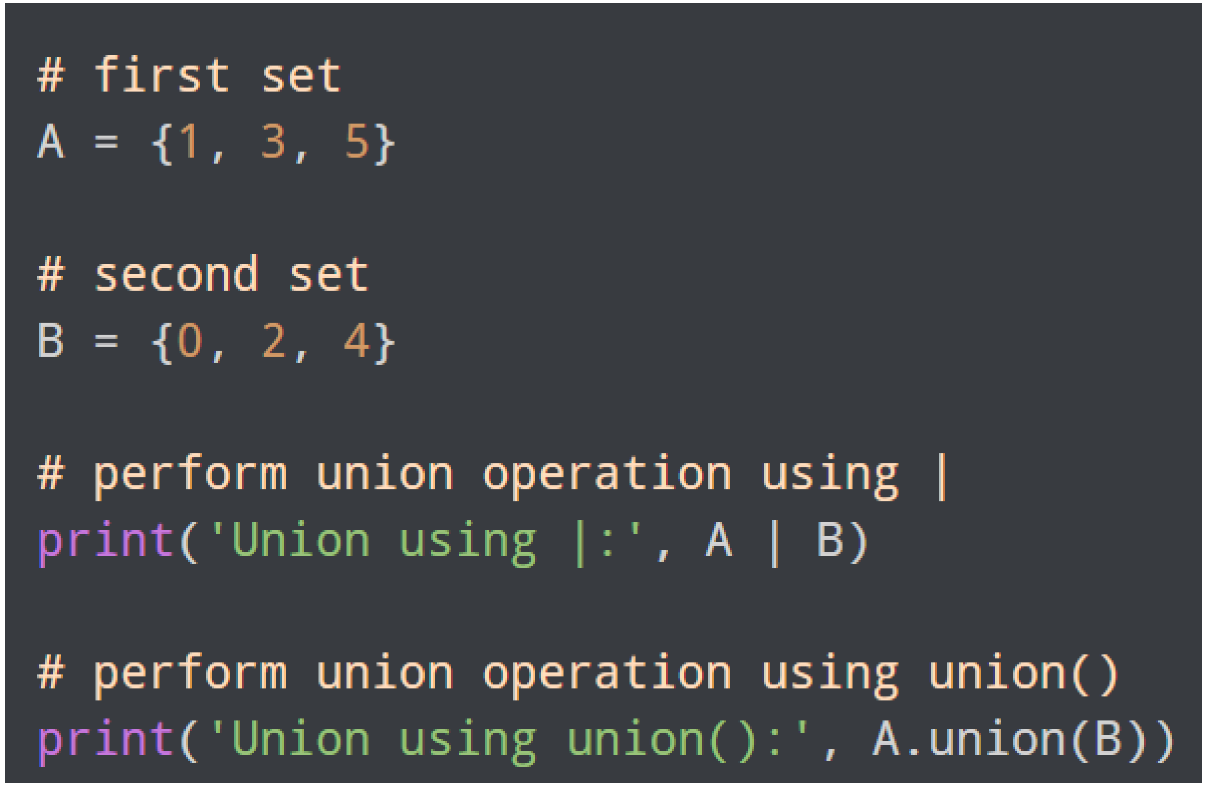


```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset) #this will raise an error because the set no longer exists
```

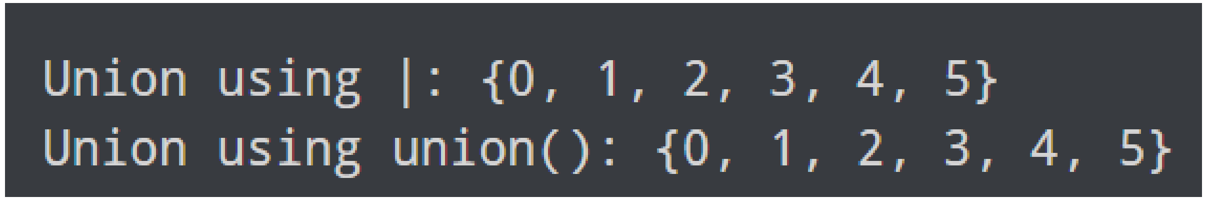
Set Operations: Union

- The union of two sets A and B include all the elements of set A and B.





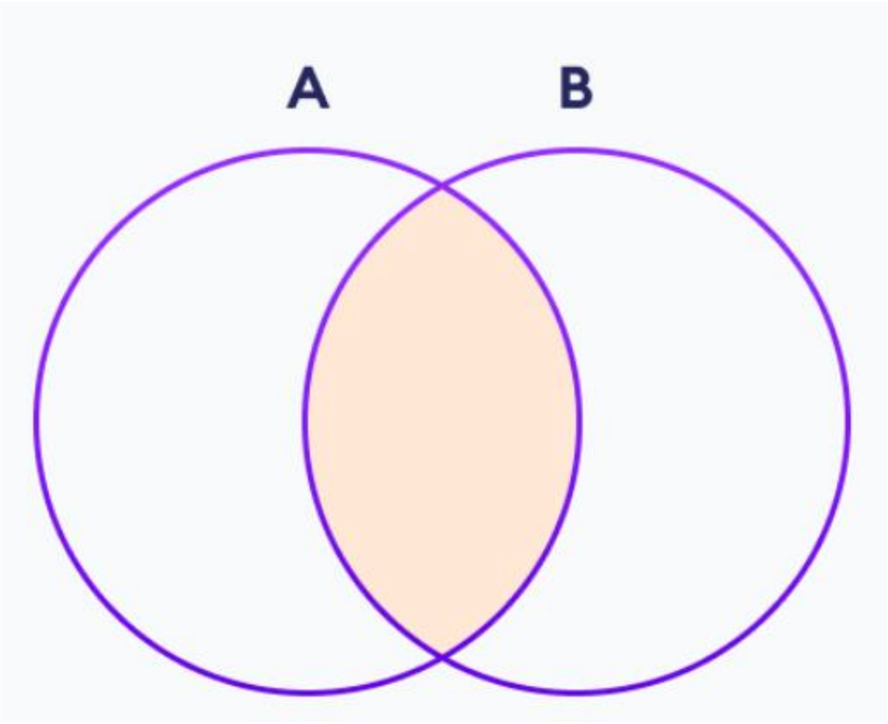
```
# first set
A = {1, 3, 5}

# 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))
```

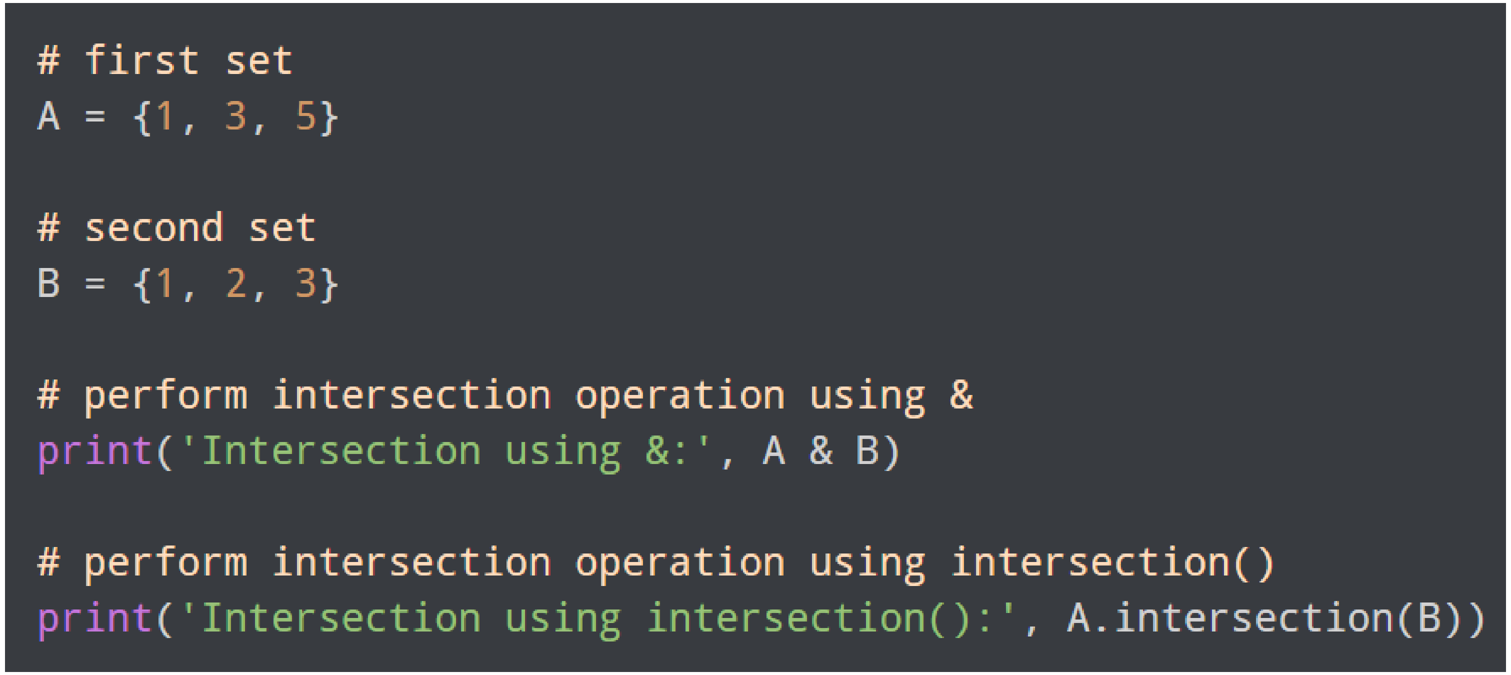


```
Union using |: {0, 1, 2, 3, 4, 5}
Union using union(): {0, 1, 2, 3, 4, 5}
```

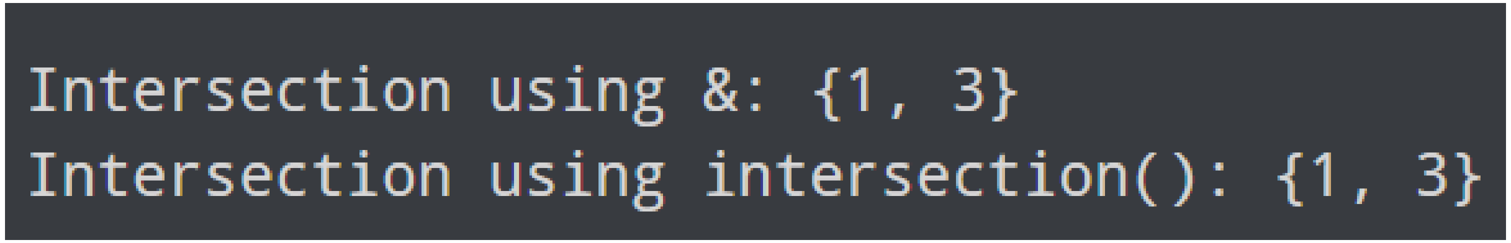
Set Operations: Intersection

- The union of two sets A and B include all the elements of set A and B.





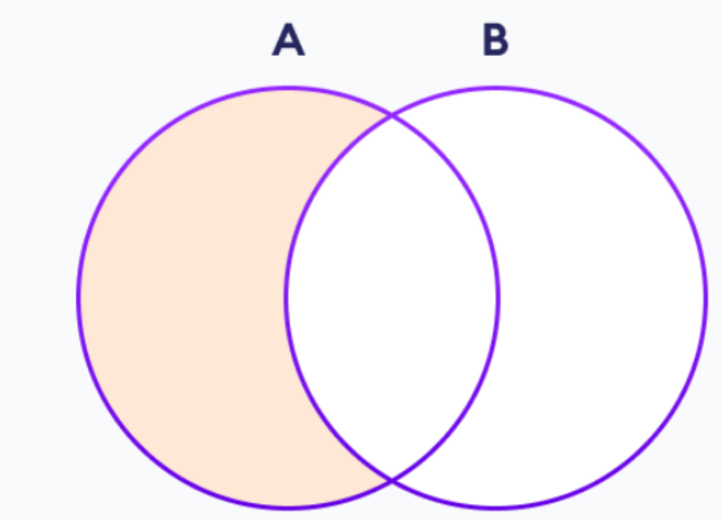
```
# first set
A = {1, 3, 5}

# 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))
```

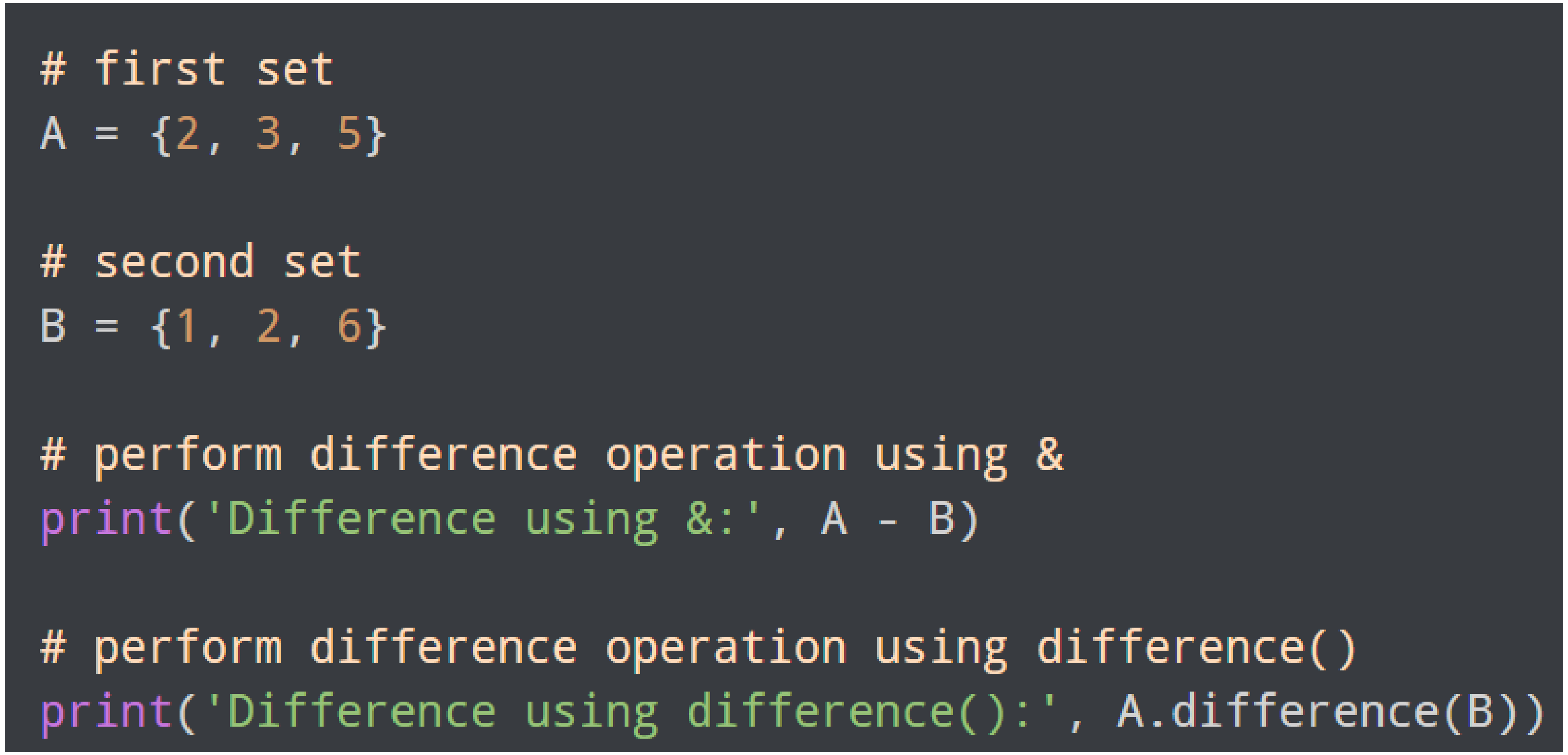


```
Intersection using &: {1, 3}
Intersection using intersection(): {1, 3}
```

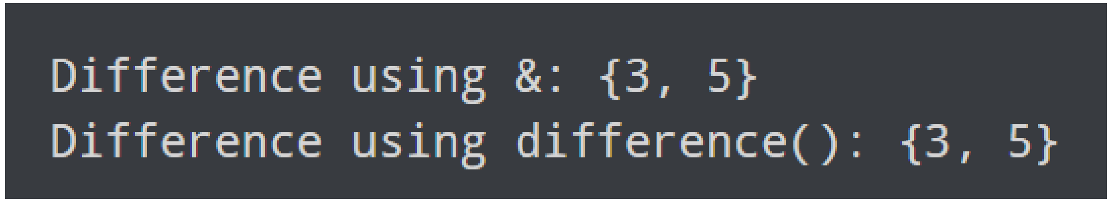
Set Operations: Difference

- The union of two sets A and B include all the elements of set A and B.





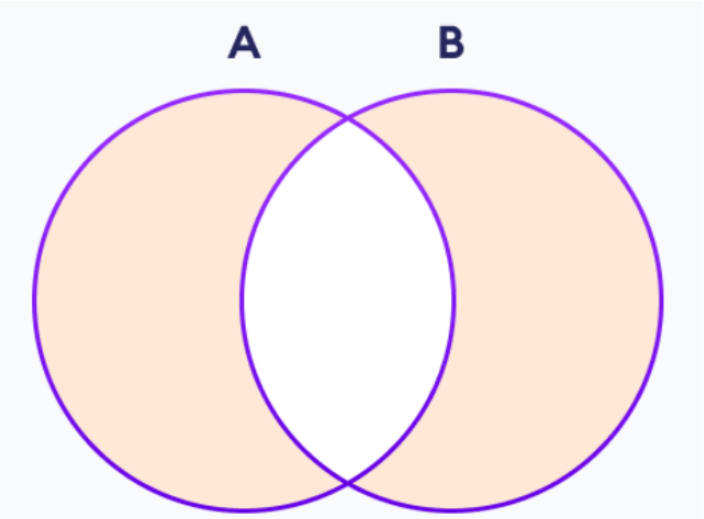
```
# first set
A = {2, 3, 5}

# 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))
```

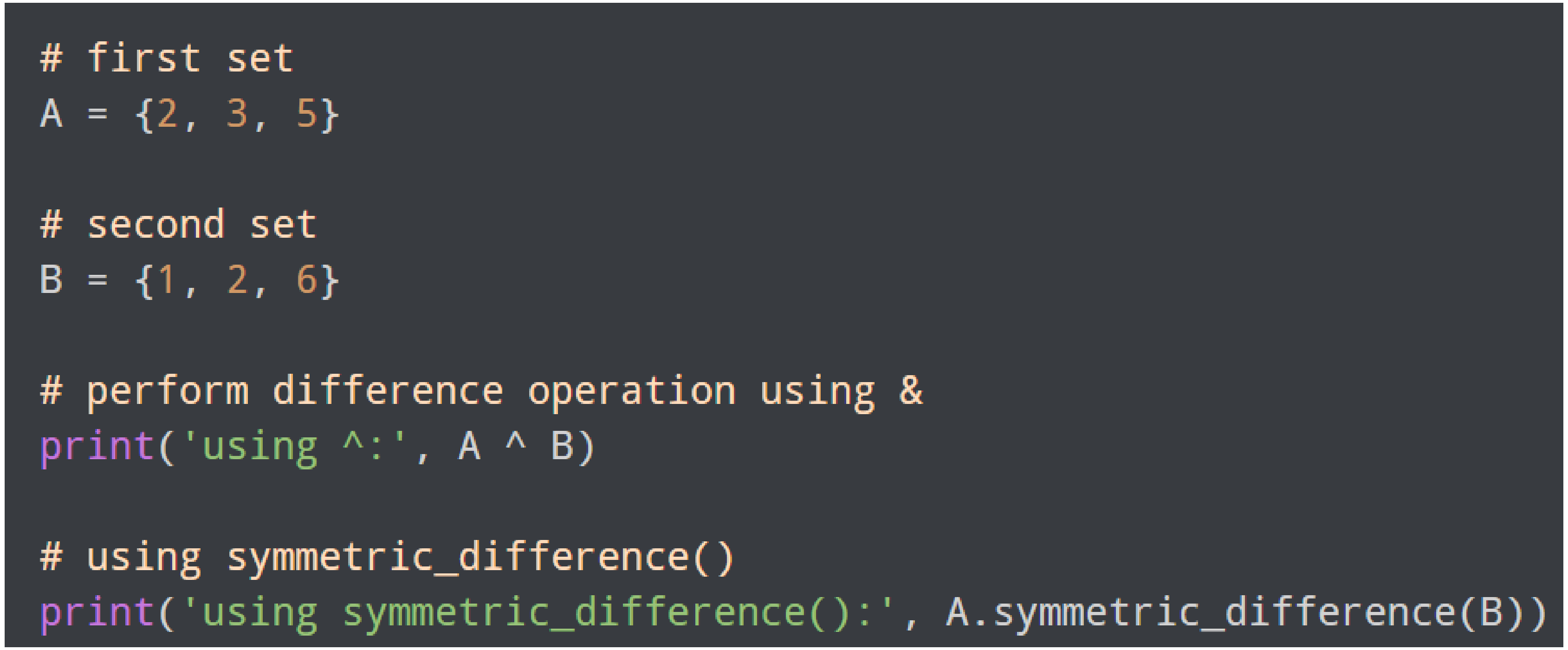


```
Difference using &: {3, 5}
Difference using difference(): {3, 5}
```

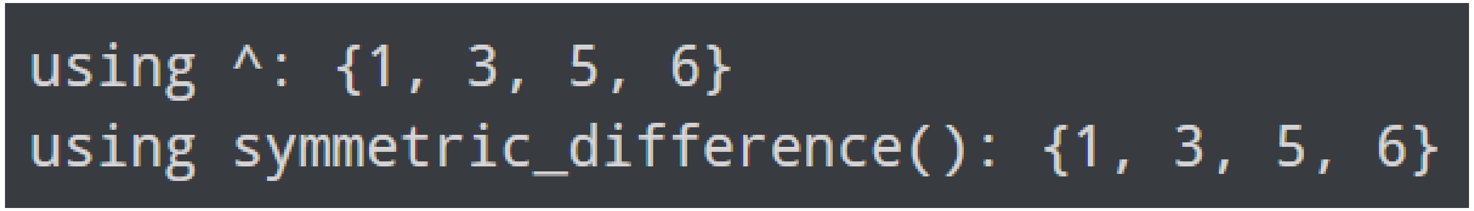
Set Operations: Symmetric Difference

- The union of two sets A and B include all the elements of set A and B.





```
# 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}
```

Exercise

- Given 2 sets, write a Python program to remove the intersection of two set from the original ones

- With the input: set1: {1, 2, 3, 4, 5}; set2: {4, 5, 6, 7, 8}
- The expected output: set1: {1, 2, 3} , set2: {6, 7, 8}

Exercise

- Write a Python program that takes a set as input and finds all of its subsets
- For example,
    - Input: set {1, 2, 3}
    - Output: the list of subsets – [{1}, {2}, {3}, {1, 2}, {2, 3}, {1, 3}, {1,2,3}]

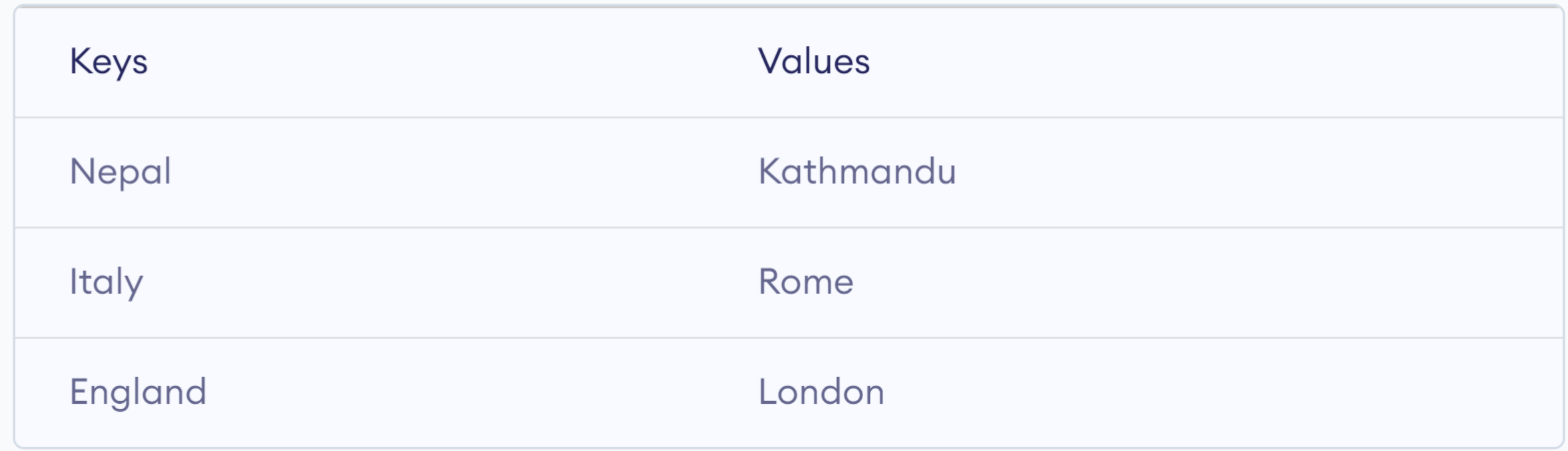
Exercise

- Set difference using bitwise operators: Write a Python function that takes in two sets as input and returns their difference as a new set using only bitwise operators (&, |, ~, ^). You cannot use the built-in set difference operator (-) or any other built-in set operations.
- For example, if the two input sets are {1, 2, 3} and {2, 3, 4}, the function should return {1, 4}.

4. Dictionary

Dictionary

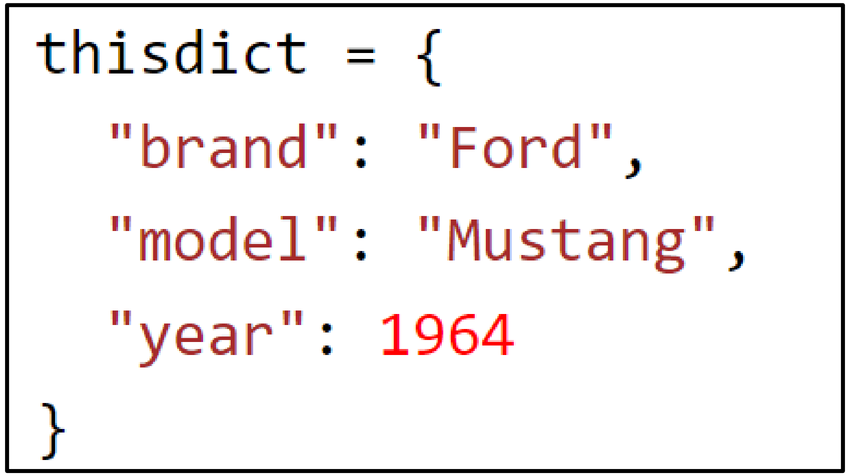
- Python dictionary is an *ordered* collection
- Dictionaries are used to store data values in **key : value** pairs
- Dictionaries are *ordered**, changeable and do not allow duplicates
- keys are **unique** identifiers that are associated with each value



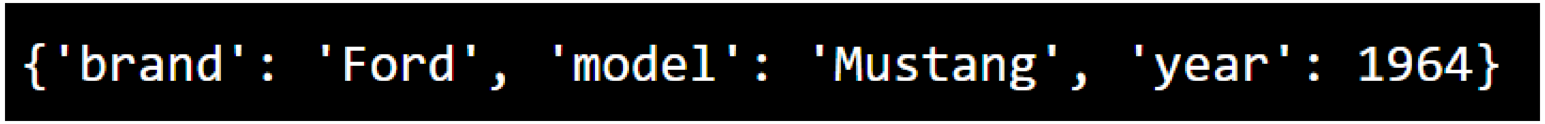
Keys	Values
Nepal	Kathmandu
Italy	Rome
England	London

Dictionary

- Create dictionary:

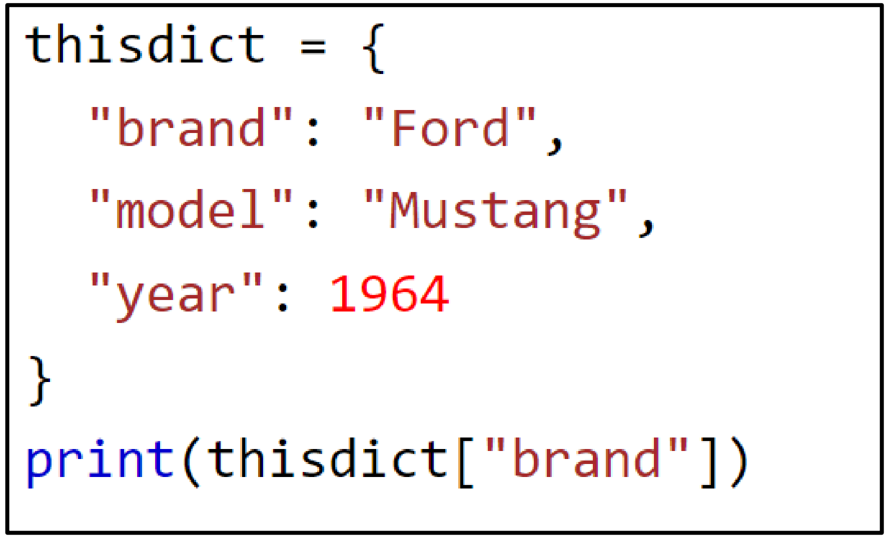


```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```



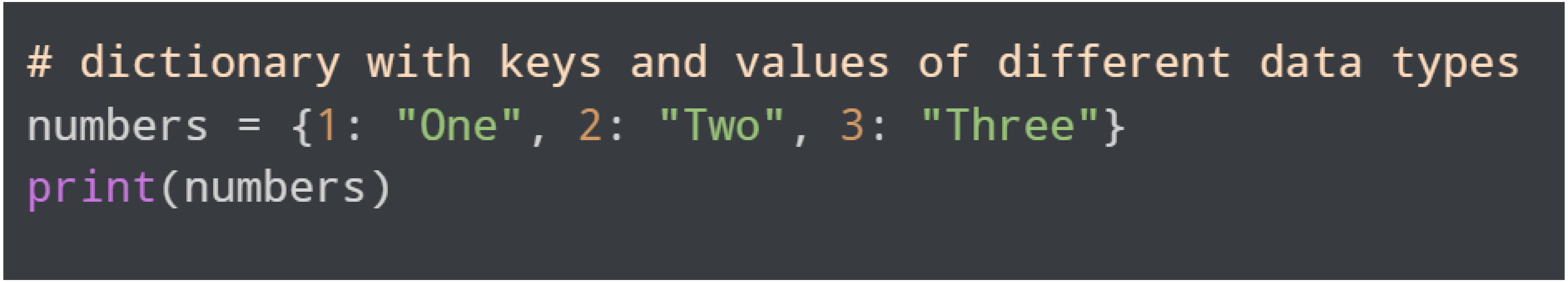
```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

- Access the items in Dictionary via key

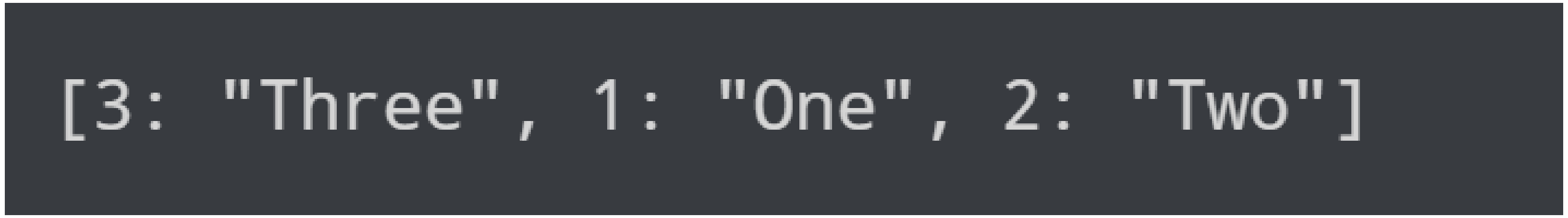


```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

Dictionary



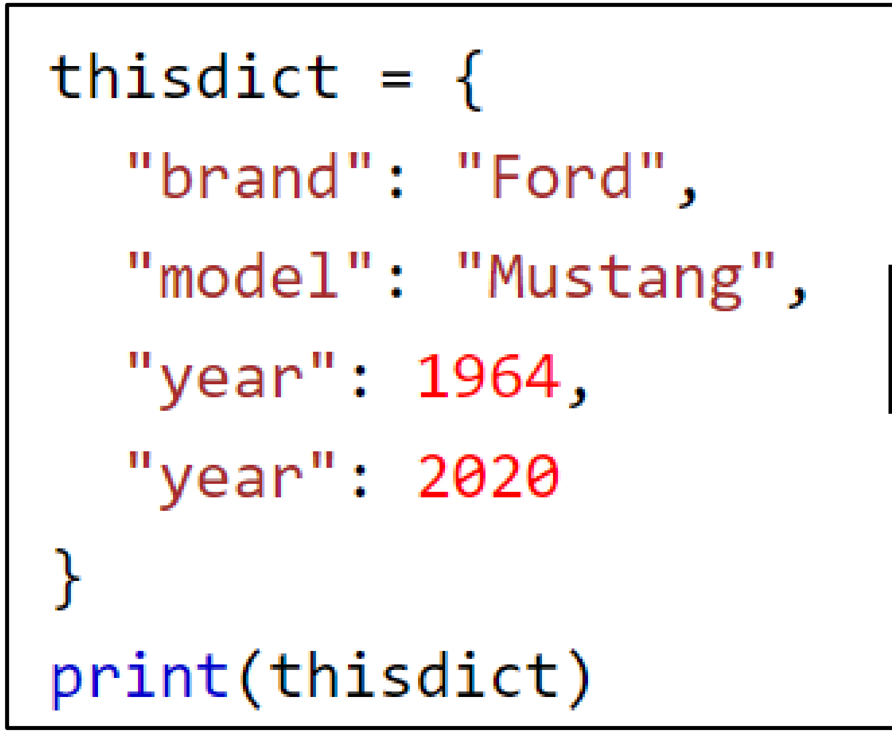
```python
# dictionary with keys and values of different data types
numbers = {1: "One", 2: "Two", 3: "Three"}
print(numbers)
```



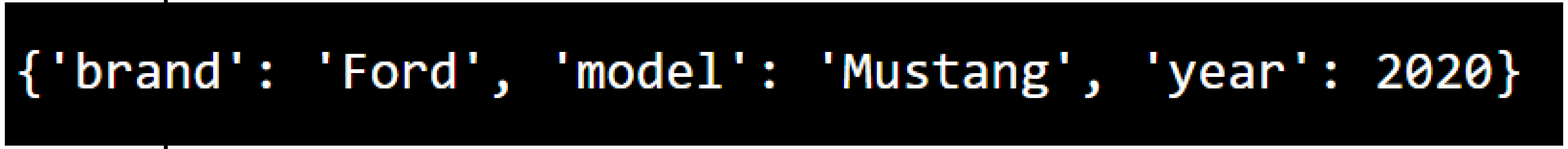
```
[3: "Three", 1: "One", 2: "Two"]
```

Duplicate keys

- Duplicate values will overwrite existing values



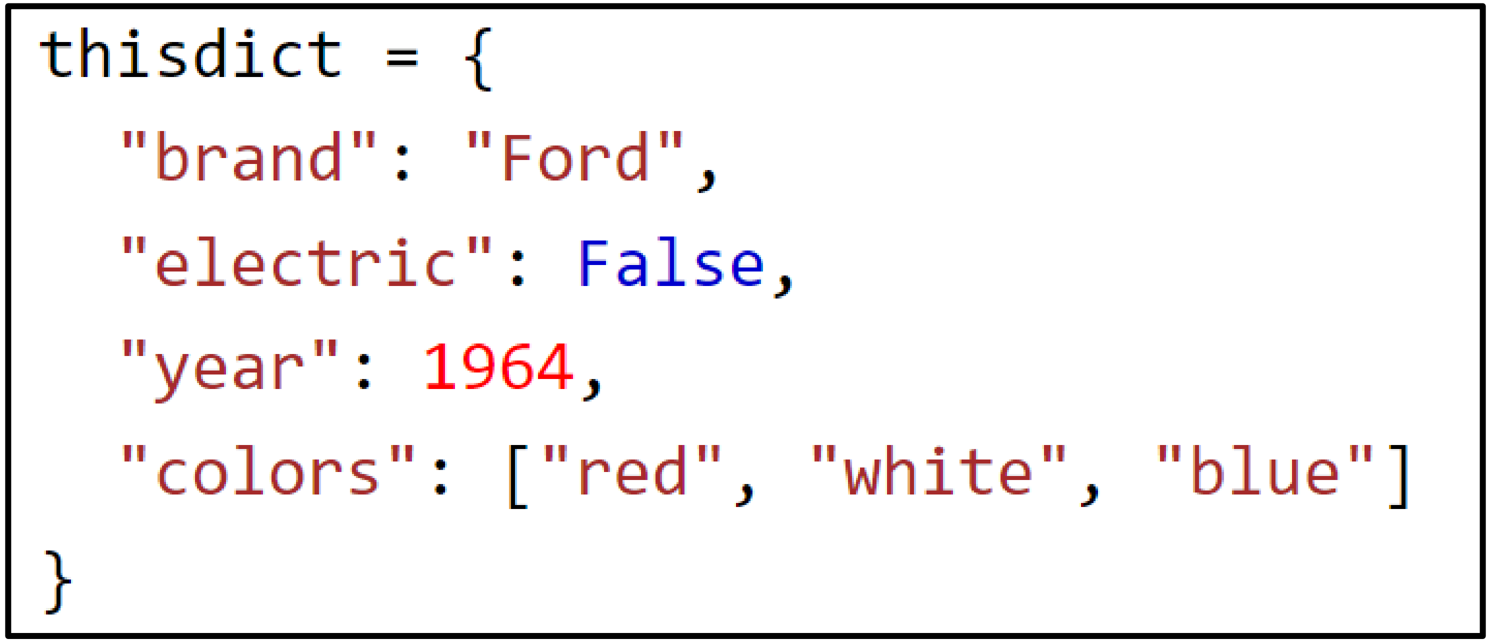
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```



```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Dictionary

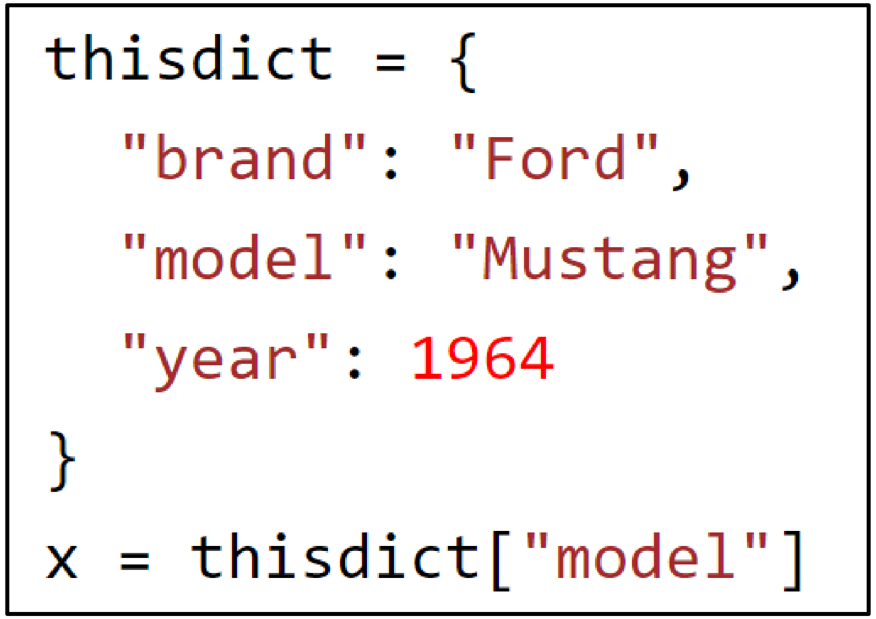
- Each item with key is a unique variable



```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

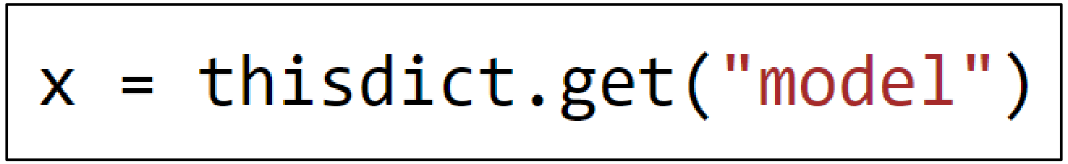
Accessing Items

- Access the items of a dictionary by referring to its key name



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

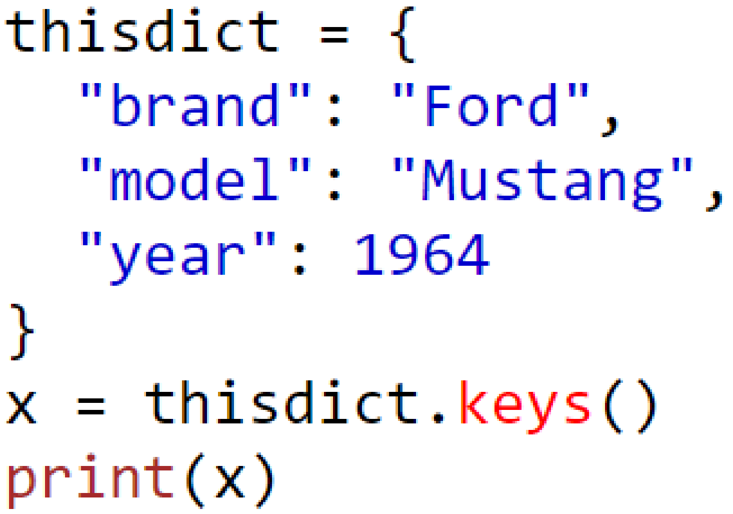
- Another way: through the method get()



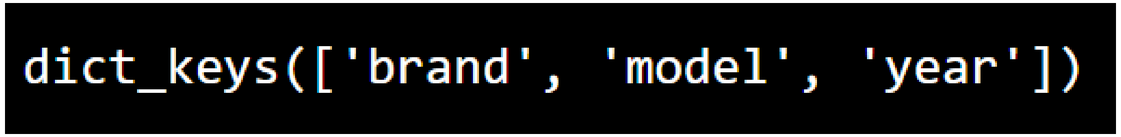
```
x = thisdict.get("model")
```

List of keys/values

- Method **keys()** will return a list of all the keys in the dictionary

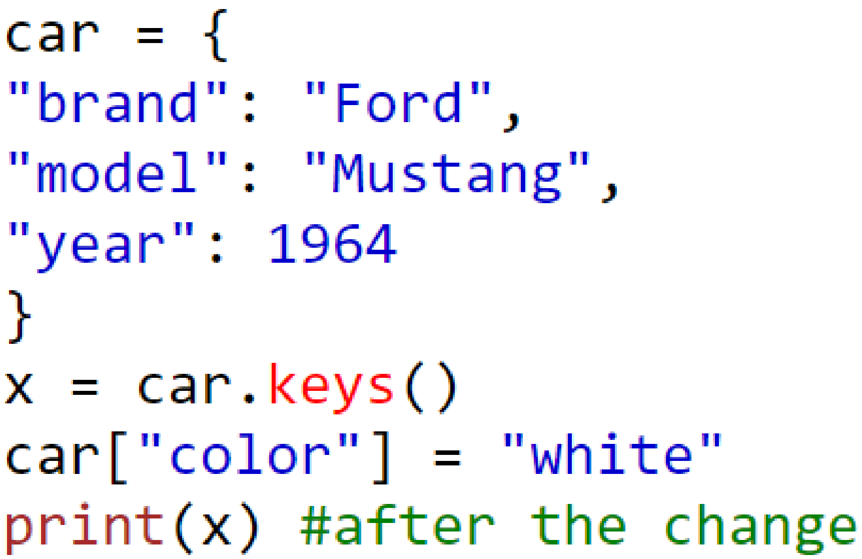


```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.keys()
print(x)
```

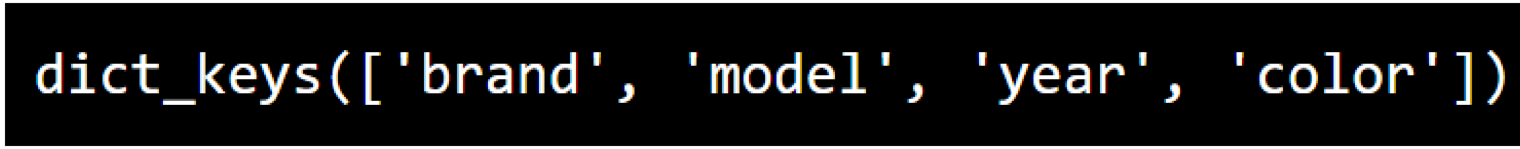


```
dict_keys(['brand', 'model', 'year'])
```

- List of the keys is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list



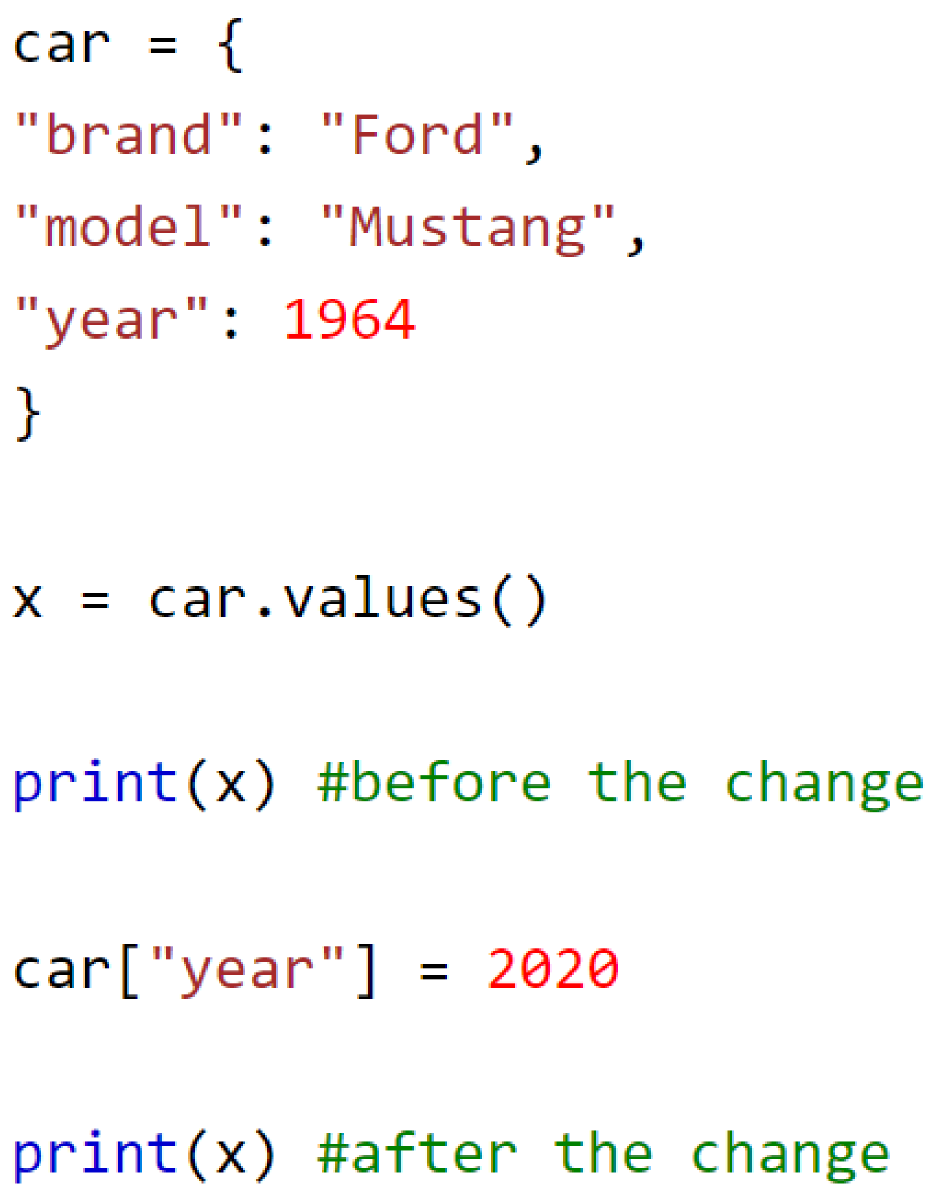
```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
x = car.keys()
car["color"] = "white"
print(x) #after the change
```



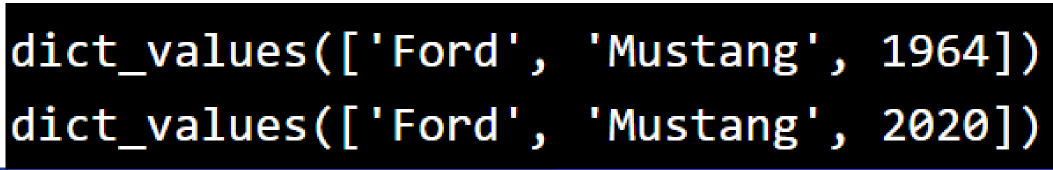
```
dict_keys(['brand', 'model', 'year', 'color'])
```

List of keys/values

- Method values() will return a list of all the values in the dictionary.
- It is a view of the dictionary



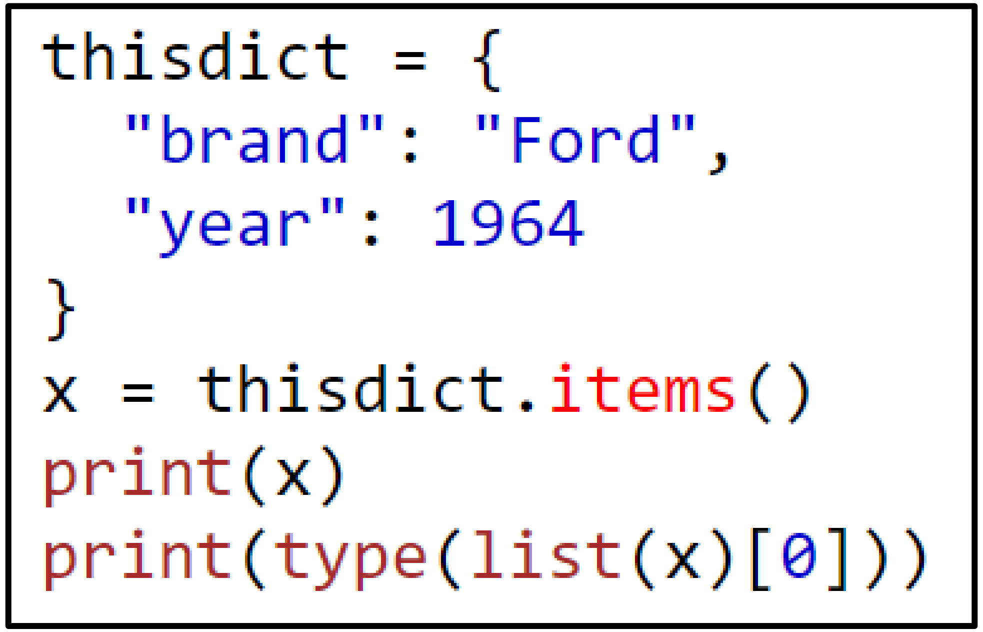
```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```



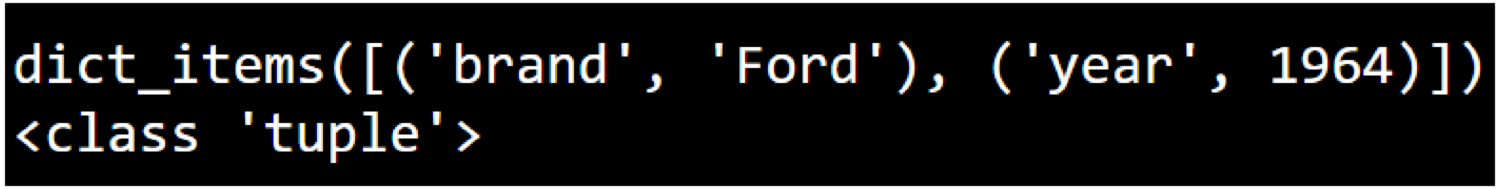
```
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 2020])
```

Get items

- Method items() return each item in a dictionary, as **tuples in a list**
- returned list is a **view** of the items of the dictionary

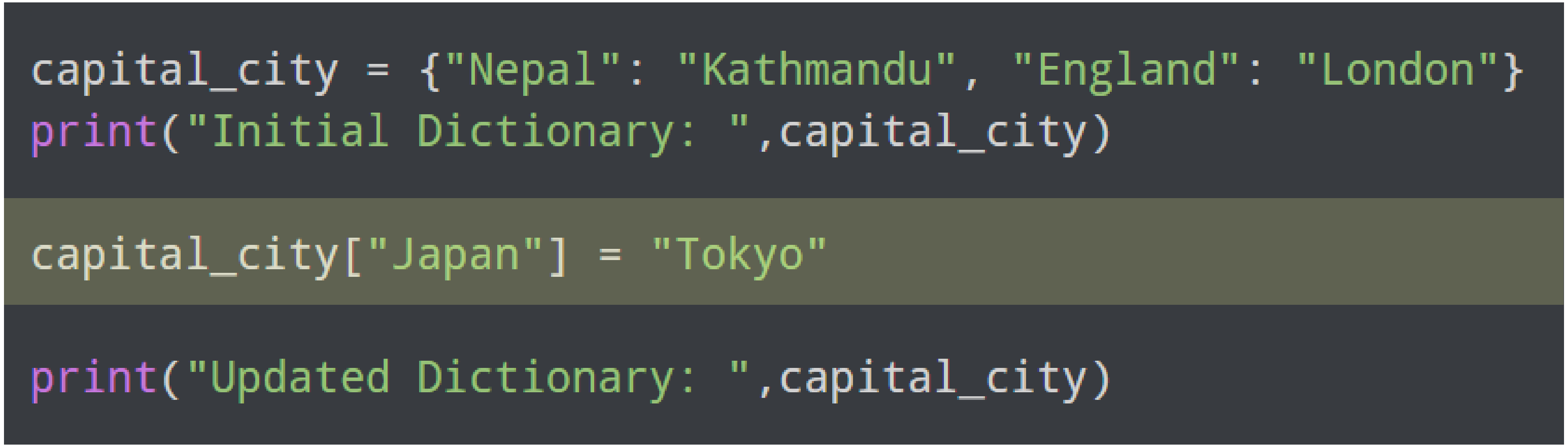


```
thisdict = {
  "brand": "Ford",
  "year": 1964
}
x = thisdict.items()
print(x)
print(type(list(x)[0]))
```

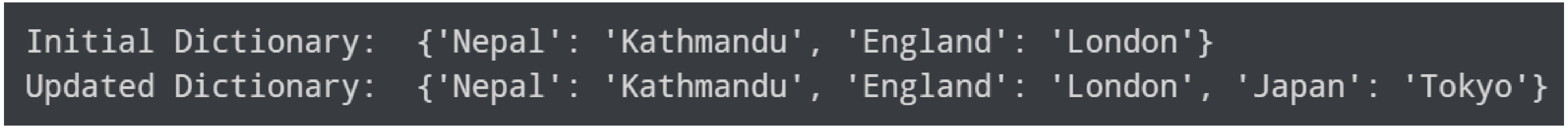


```
dict_items([('brand', 'Ford'), ('year', 1964)])
<class 'tuple'>
```

Add elements



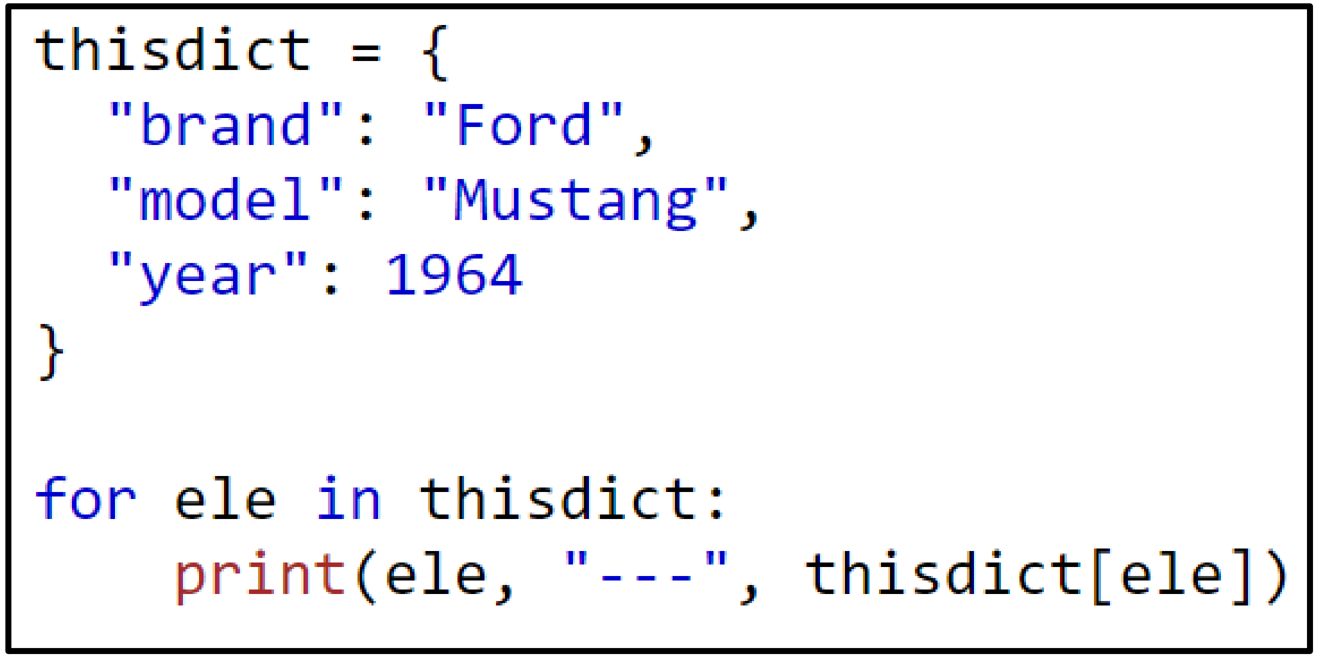
```python
capital_city = {"Nepal": "Kathmandu", "England": "London"}
print("Initial Dictionary: ",capital_city)

capital_city["Japan"] = "Tokyo"

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



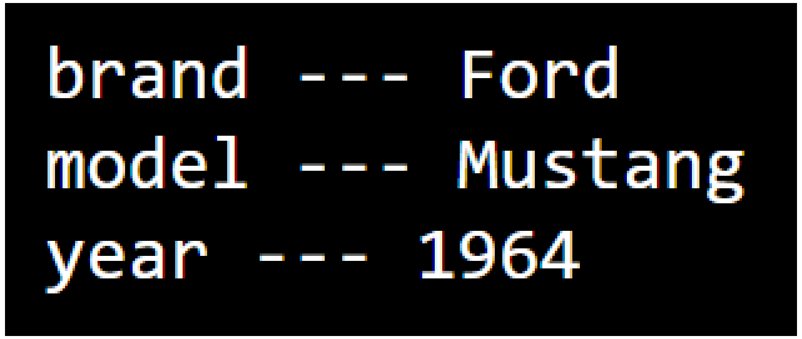
```
Initial Dictionary:  {'Nepal': 'Kathmandu', 'England': 'London'}
Updated Dictionary:  {'Nepal': 'Kathmandu', 'England': 'London', 'Japan': 'Tokyo'}
```

Loop in Dictionary

- The representative object in Dictionary is key



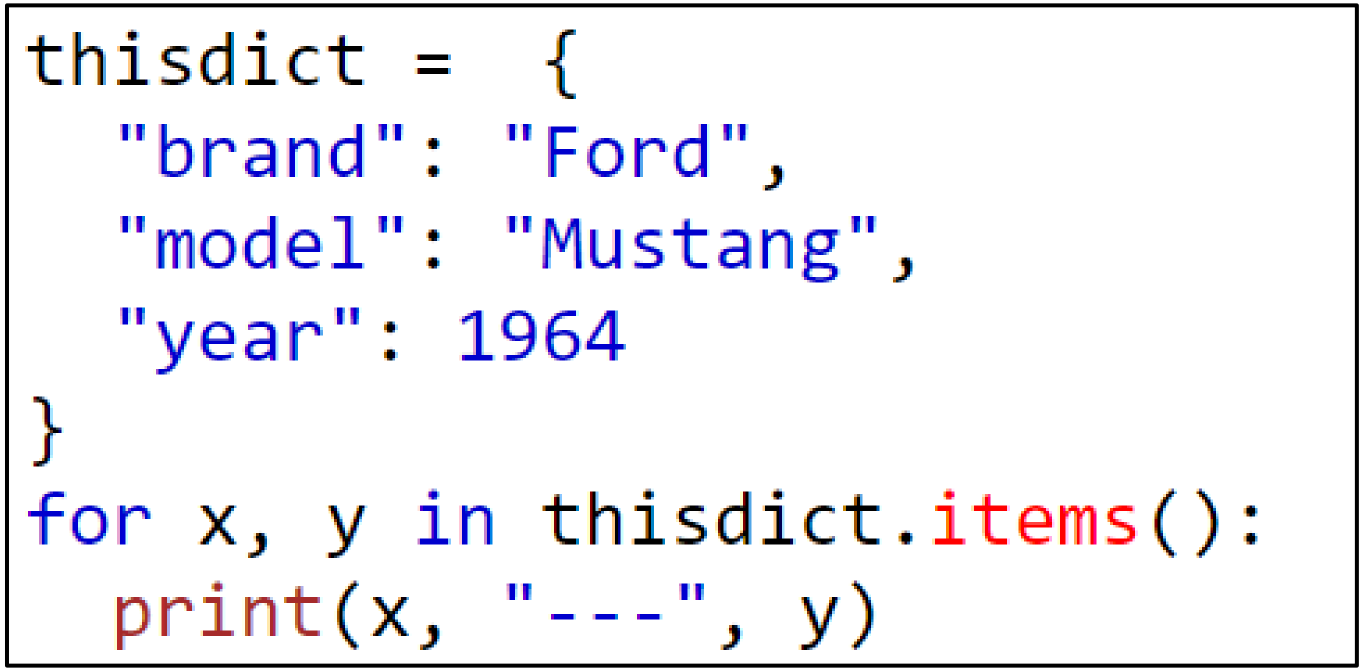
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

for ele in thisdict:
    print(ele, "---", thisdict[ele])
```



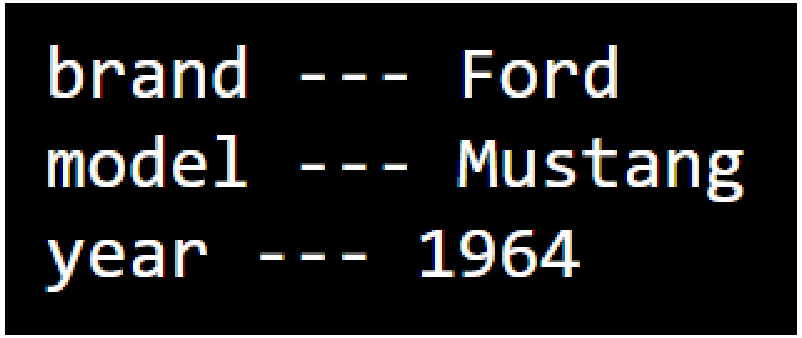
```
brand --- Ford
model --- Mustang
year --- 1964
```

Loop in Dictionary

- Loop through both keys and values, by using the **items()** method:



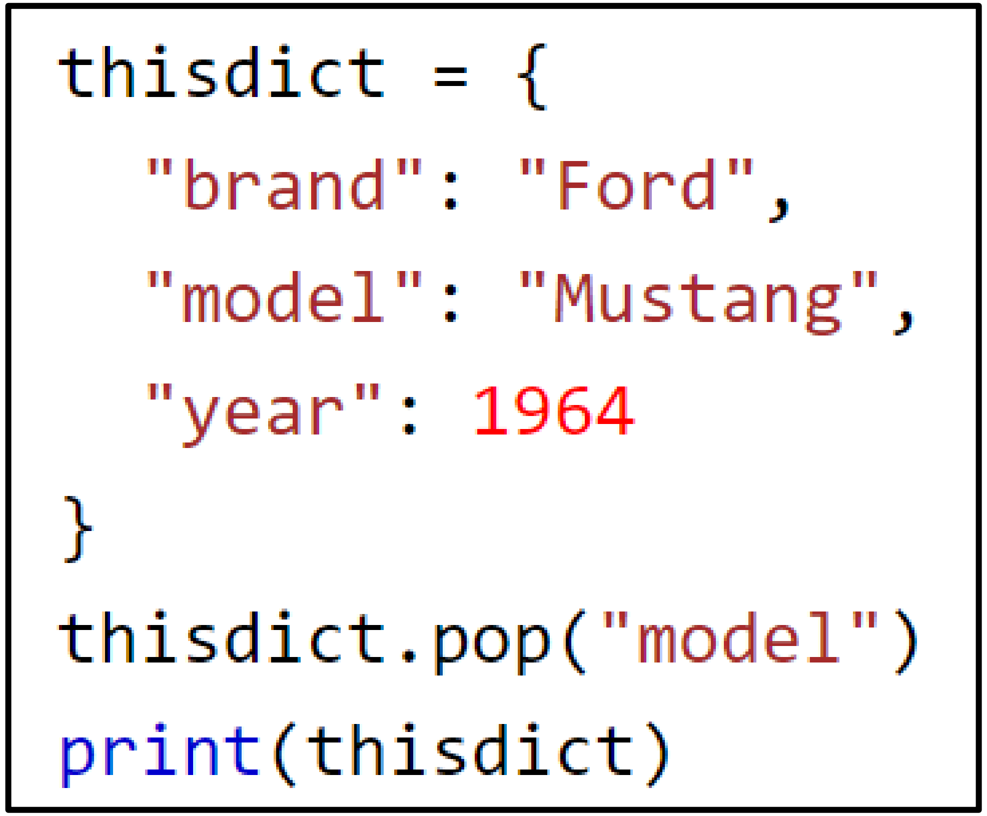
```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, "---", y)
```



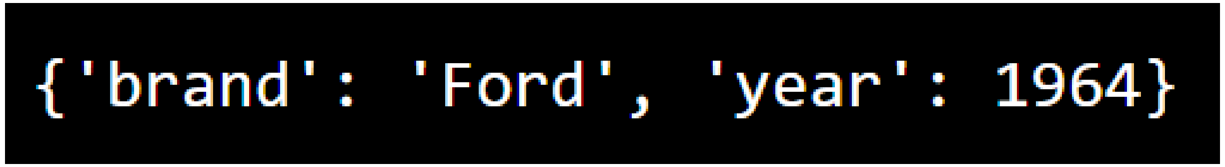
```
brand --- Ford
model --- Mustang
year --- 1964
```

Remove Items

- pop() method removes the item with the specified key name



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

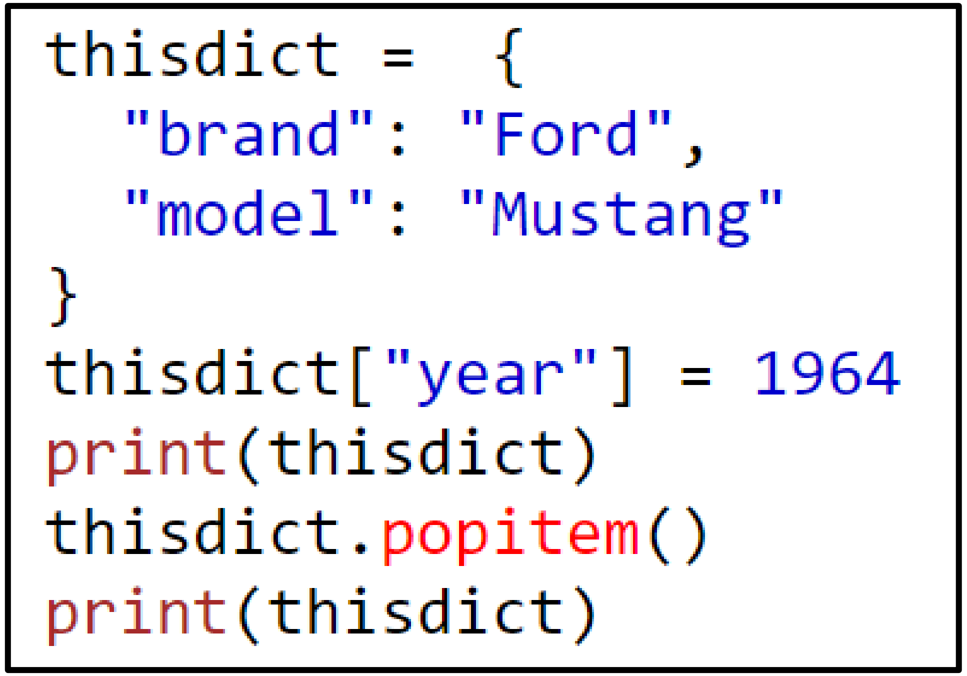


```
{'brand': 'Ford', 'year': 1964}
```

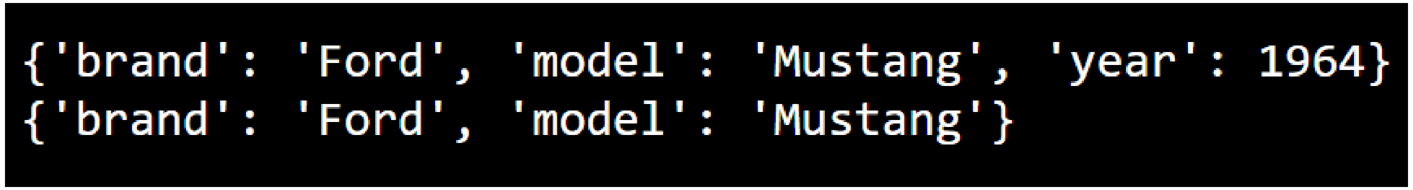
- If key name is not in Dictionary, raise error

Remove Items

- popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead)



```python
thisdict =  {
  "brand": "Ford",
  "model": "Mustang"
}
thisdict["year"] = 1964
print(thisdict)
thisdict.popitem()
print(thisdict)
```

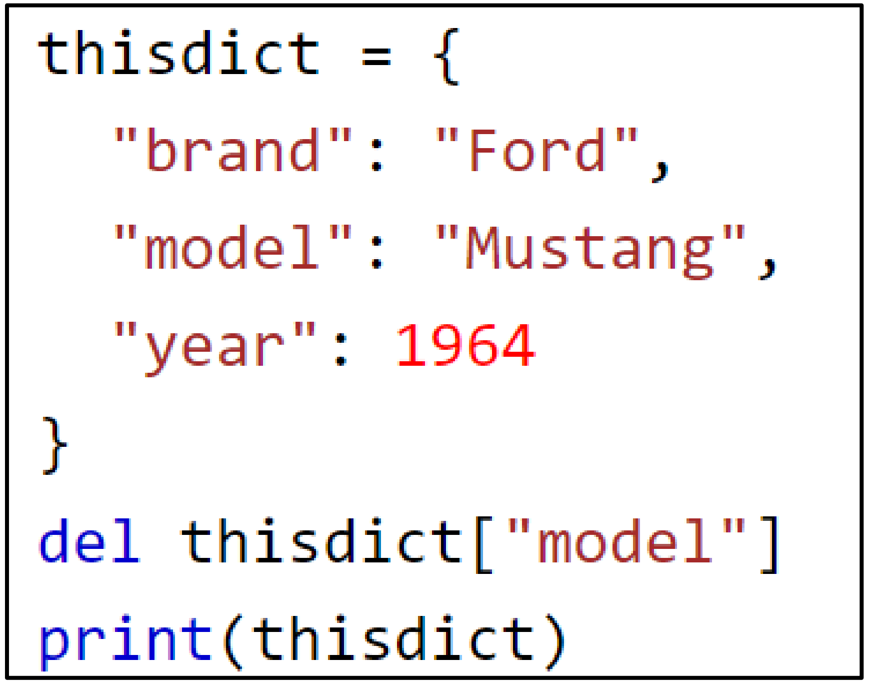


```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
{'brand': 'Ford', 'model': 'Mustang'}
```

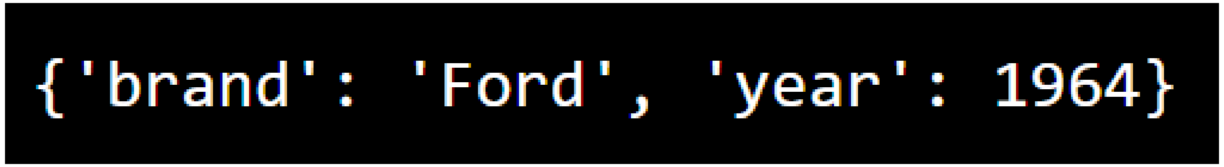
- If the dictionary is empty, raise error
- From Python 3.7, the dictionary is ordered

Remove Items

- del keyword removes the item with the specified key name
- clear() method empties the dictionary



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

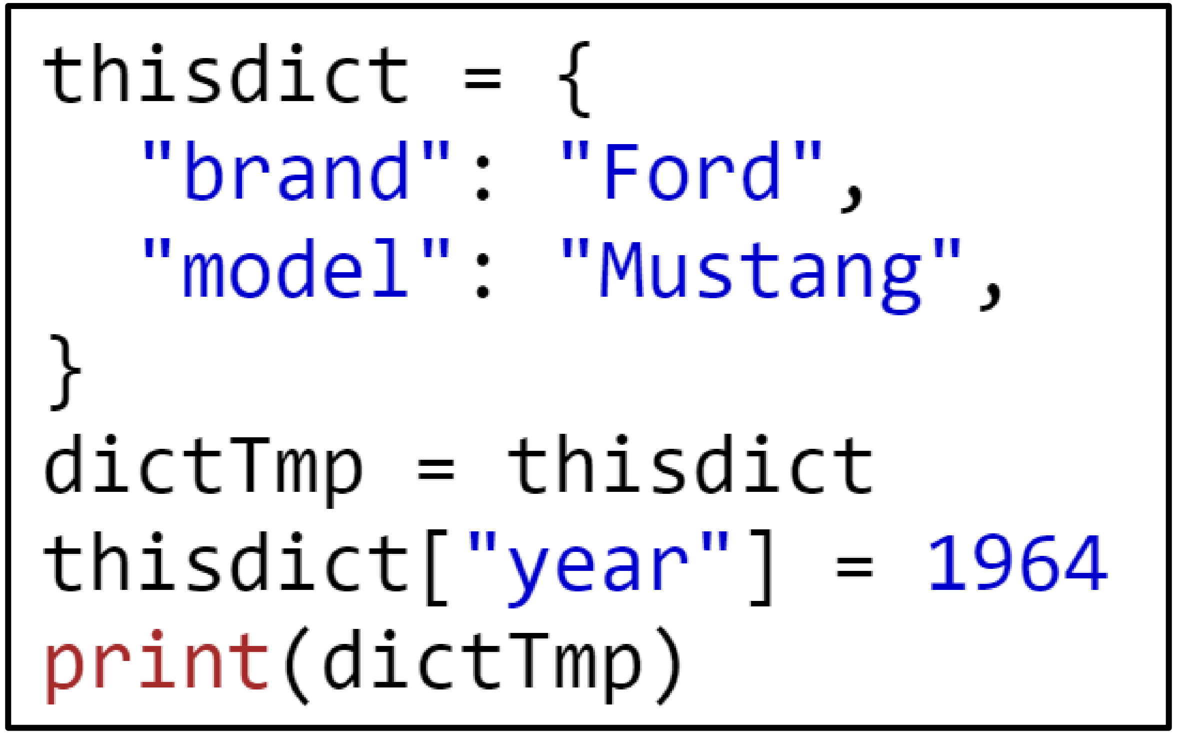


```
{'brand': 'Ford', 'year': 1964}
```

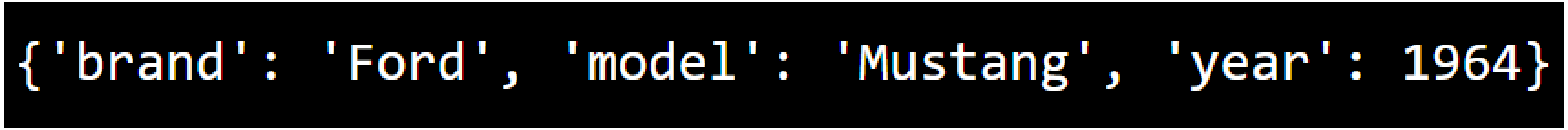
- The del keyword can also delete the dictionary completely

Copy a Dictionary

- Assignment “=“ in Dictionary is useless



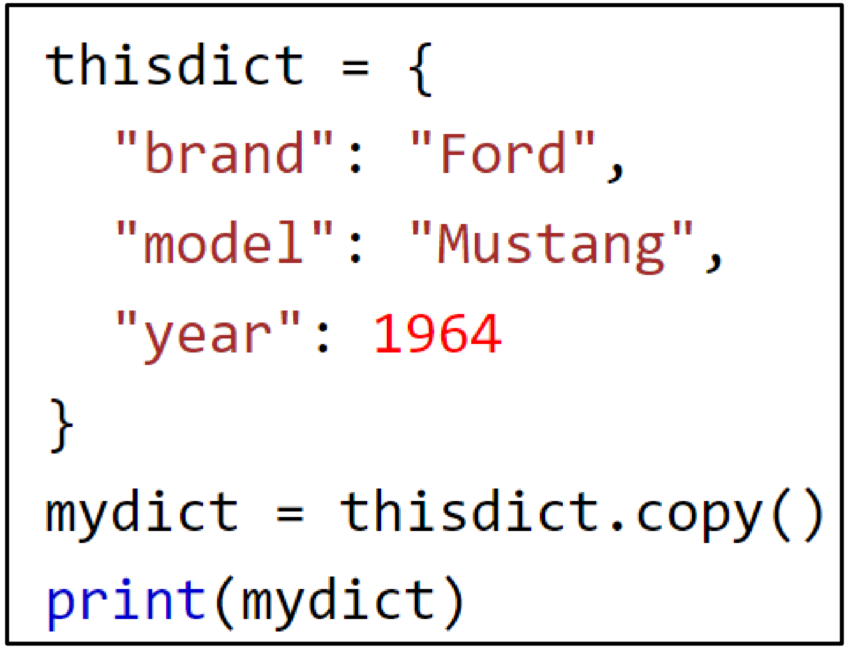
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
}
dictTmp = thisdict
thisdict["year"] = 1964
print(dictTmp)
```



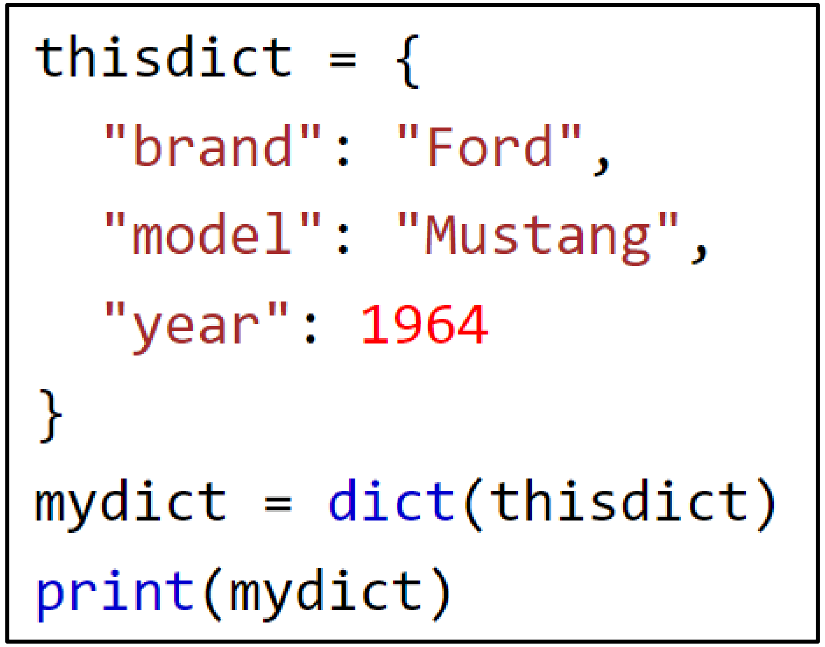
```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Copy a Dictionary

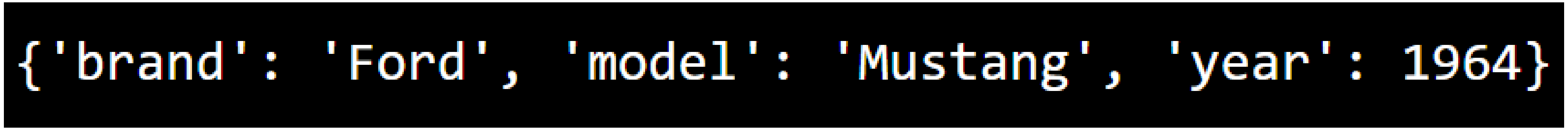
- To make a copy of Dictionary, use the built-in Dictionary method copy()
- Another way to make a copy is to use the built-in function dict()



```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```



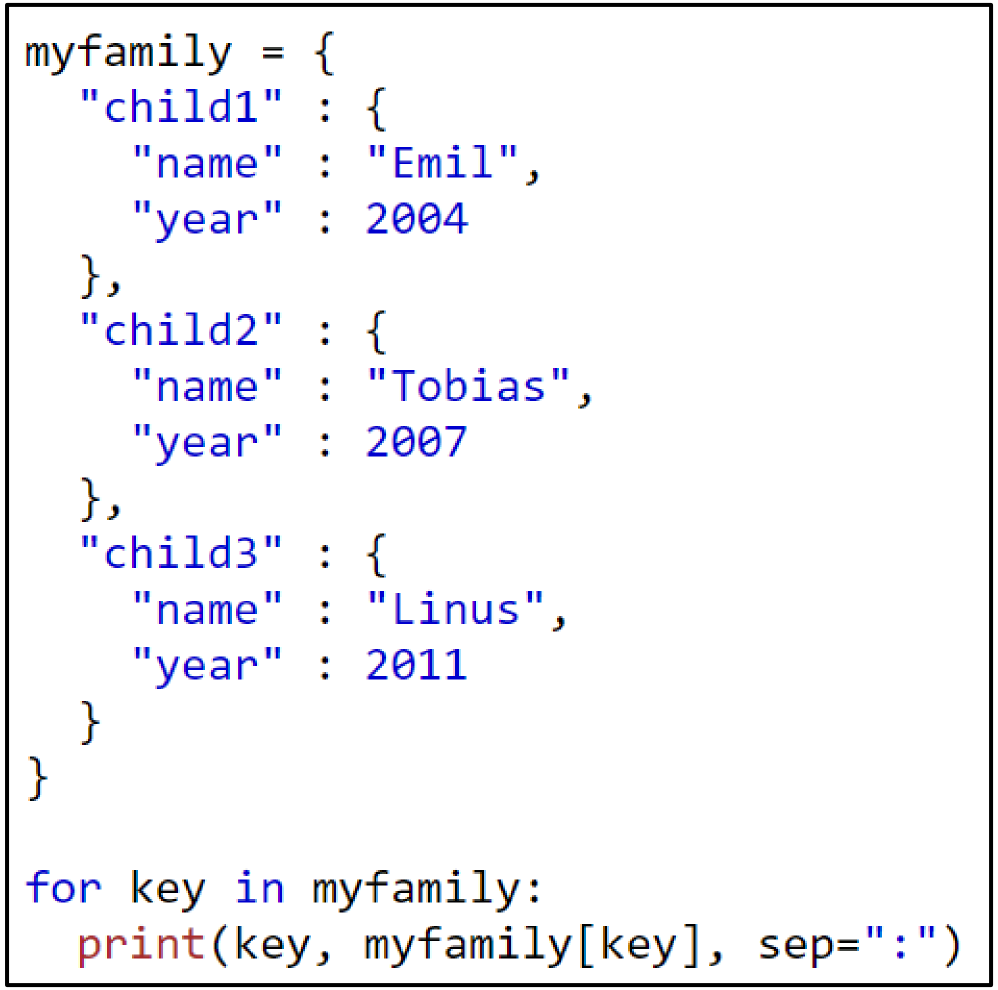
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```



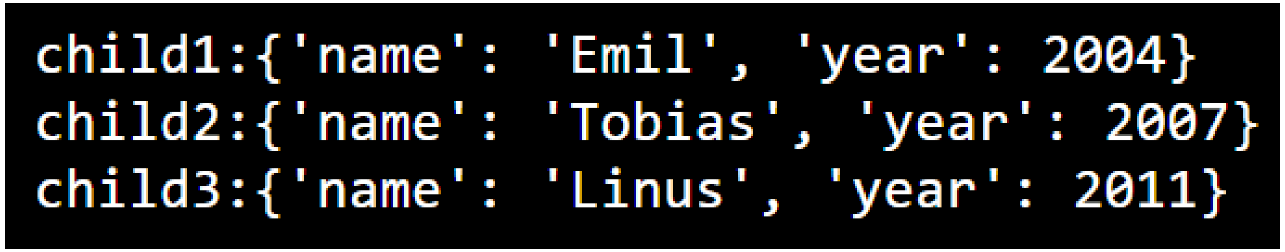
```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Nested Dictionary

- A dictionary can contain dictionaries

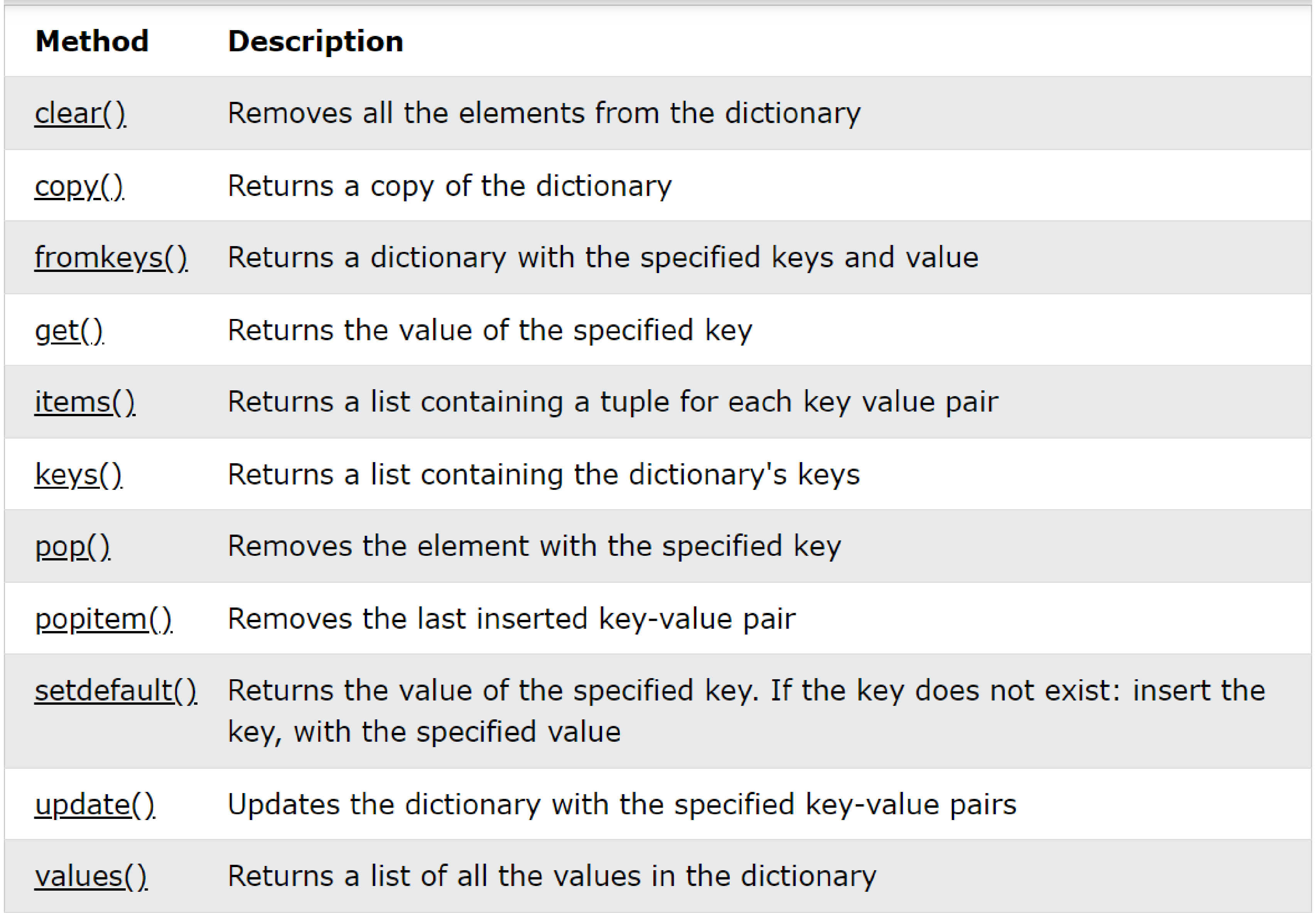


```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

for key in myfamily:
  print(key, myfamily[key], sep=":")
```



```
child1:{'name': 'Emil', 'year': 2004}
child2:{'name': 'Tobias', 'year': 2007}
child3:{'name': 'Linus', 'year': 2011}
```

Dictionary Methods



Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Exercise

- Given an array of names of candidates in an election. A candidate name in the array represents a vote cast to the candidate. Print the name of candidates received Max vote. If there is tie, print a lexicographically smaller name.
- **Input:** john/johnny/jackie/johnny/john/jackie/jamie/jamie/john/johnny/jamie/johnny/john
- **Output:** john

Exercise

- Write a function to print the number of days in a month with a specified month and year inputted from keyboards.

Exercise

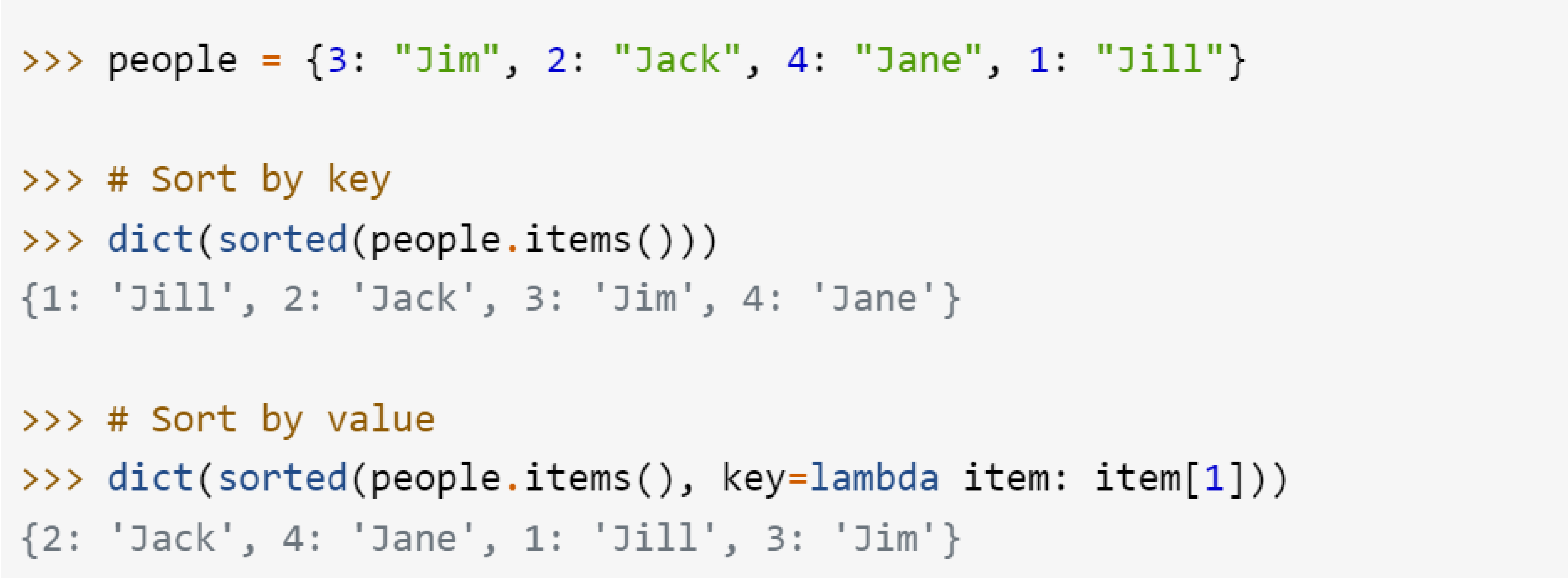
- Sort Dictionary via keys

Exercise

- Sort Dictionary via values

Sort Dictionary via values

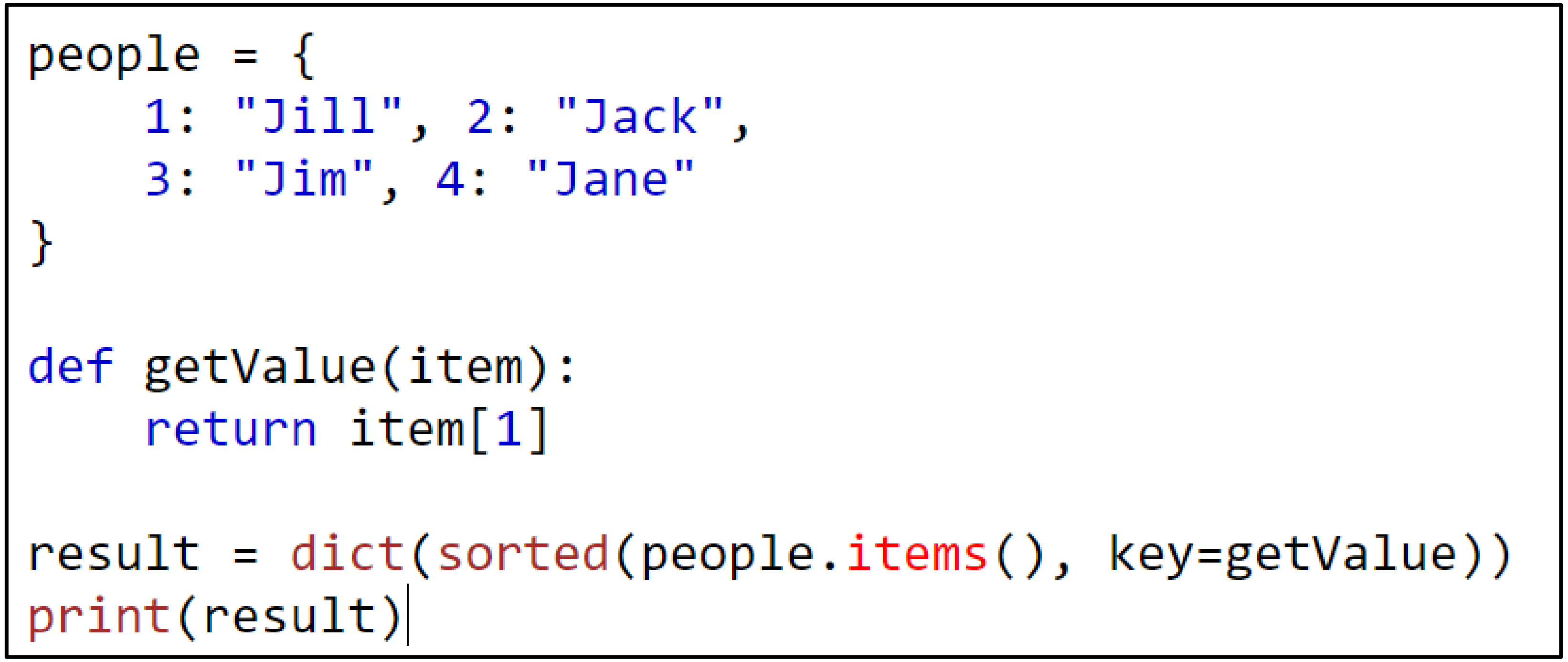
- Using **lambda** function



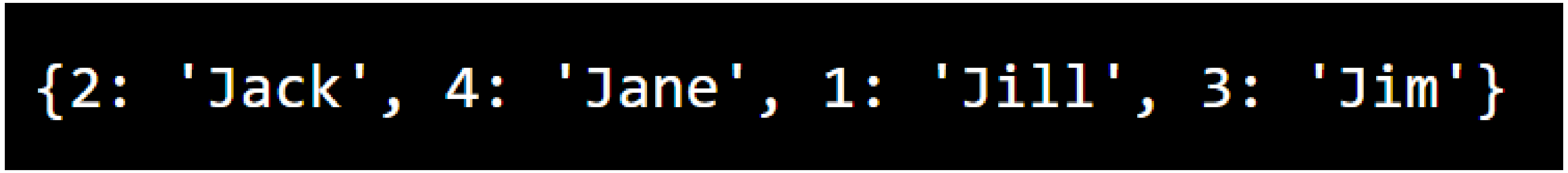
```
>>> people = {3: "Jim", 2: "Jack", 4: "Jane", 1: "Jill"}

>>> # Sort by key
>>> dict(sorted(people.items()))
{1: 'Jill', 2: 'Jack', 3: 'Jim', 4: 'Jane'}

>>> # Sort by value
>>> dict(sorted(people.items(), key=lambda item: item[1]))
{2: 'Jack', 4: 'Jane', 1: 'Jill', 3: 'Jim'}
```

Sort Dictionary via values

- Using user-defined functions



```
people = {
    1: "Jill", 2: "Jack",
    3: "Jim", 4: "Jane"
}

def getValue(item):
    return item[1]

result = dict(sorted(people.items(), key=getValue))
print(result)
```



```
{2: 'Jack', 4: 'Jane', 1: 'Jill', 3: 'Jim'}
```

THANK YOU

for YOUR ATTENTION