



fit@hcmus

VNUHCM - UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

University of Science - VNU-HCM
Faculty of Information Science
Department of Computer Science

MTH083 - Advanced Programming for Artificial Intelligence

Slot 05- Tuple – Set – Dictionary

Advisor:

Dr. Nguyễn Tiến Huy

Dr. Lê Thanh Tùng

Part 1: 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
<code>len((1, 2, 3))</code>	3	Length
<code>(1, 2, 3) + (4, 5, 6)</code>	<code>(1, 2, 3, 4, 5, 6)</code>	Concatenation
<code>('Hi!') * 4</code>	<code>('Hi!', 'Hi!', 'Hi!', 'Hi!')</code>	Repetition
<code>3 in (1, 2, 3)</code>	True	Membership
<code>for x in (1, 2, 3): print x,</code>	1 2 3	Iteration

Basic Tuples Methods

Method	Description
<u>count()</u>	Returns the number of times a specified value occurs in a tuple
<u>index()</u>	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

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 count all prime number in a tuple t.

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:

```
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']`

Part 2: 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

```
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**

```
fruits = {"Apple", "Peach", "Mango"}  
  
# for loop to access each fruits  
for fruit in fruits:  
    print(fruit)
```

```
thisset = {"apple", "banana", "cherry"}  
  
print("banana" in thisset)
```

Add items

- Using `add()` method

```
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

```
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

```
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.

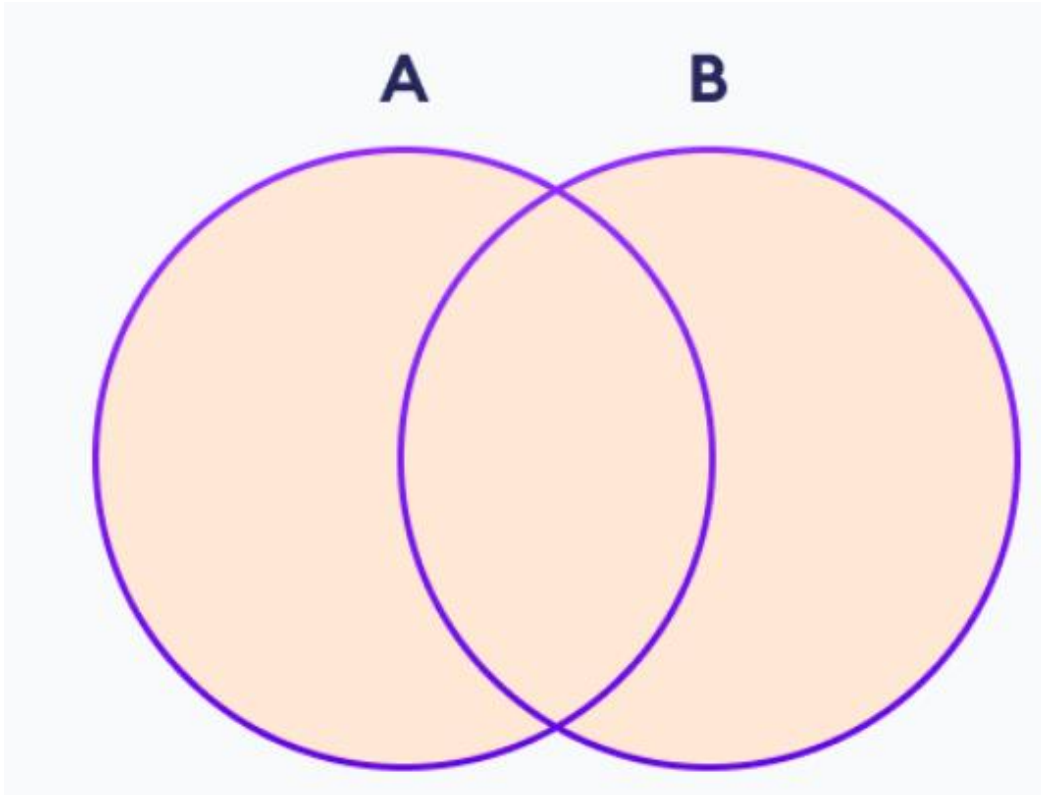
Remove item

- `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)

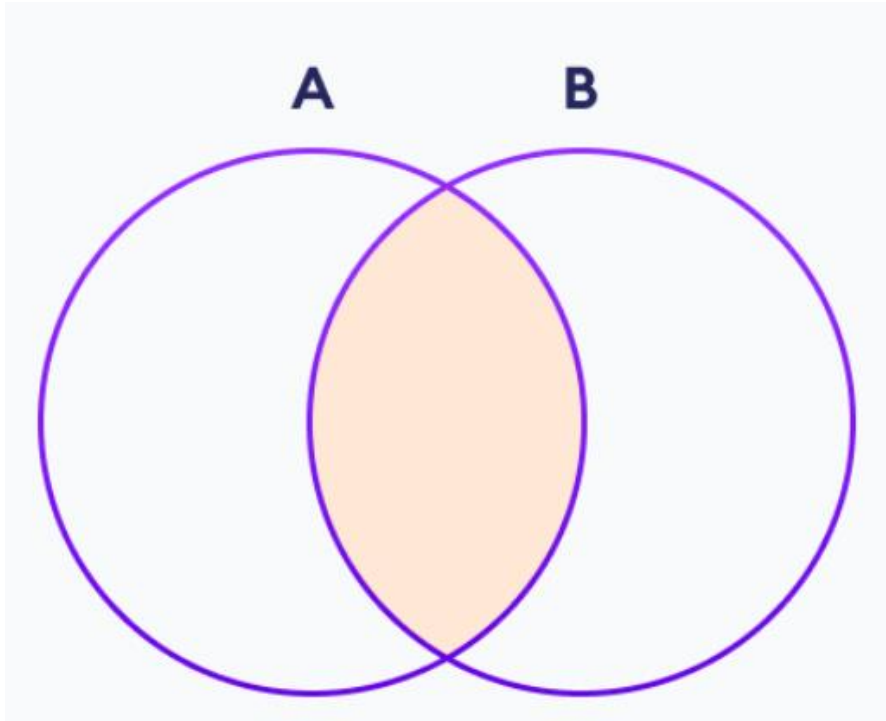
# perform union operation using union()
print('Union using union():', A.union(B))
```

```
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}

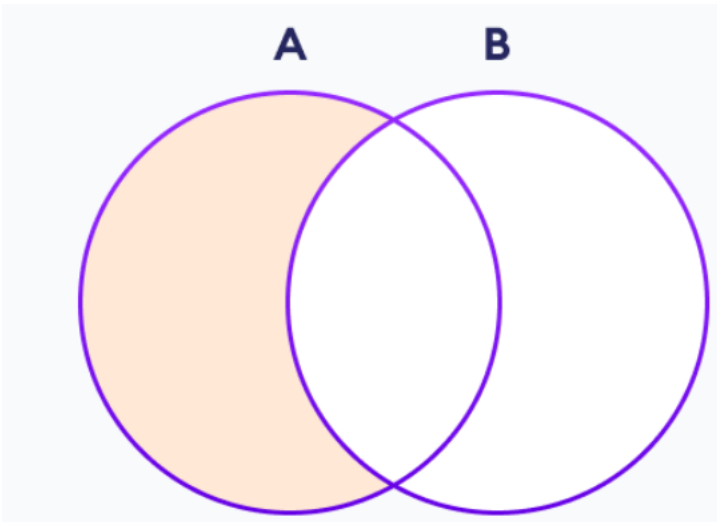
# perform intersection operation using &
print('Intersection using &:', A & B)

# 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}

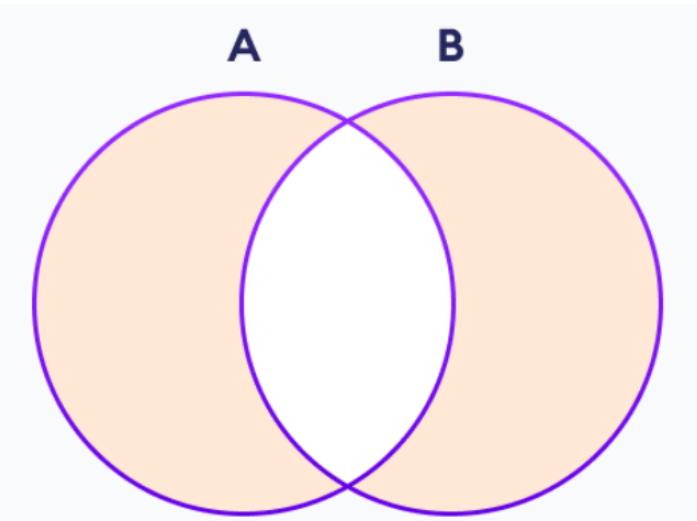
# perform difference operation using &
print('Difference using &:', A - B)

# 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}.

Part 3: 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  
}
```

- Access the items in Dictionary via key

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

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

Dictionary

```
# 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

```
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)

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

Part 4: JSON

JSON Format

- JSON (JavaScript Object Notation) is a popular data format used for representing structured data
- JSON format is quite similar dictionary in Python

```
{  "emp_details": [  
    {  
        "emp_name": "Shubham",  
        "email": "ksingh.shubh@gmail.com",  
        "job_profile": "intern"  
    },  
    {  
        "emp_name": "Gaurav",  
        "email": "gaurav.singh@gmail.com",  
        "job_profile": "developer"  
    },  
    {  
        "emp_name": "Nikhil",  
        "email": "nikhil@geeksforgeeks.org",  
        "job_profile": "Full Time"  
    }  
  ]  
}
```

Parse JSON in Python

- The **json** module makes it easy to parse JSON strings and files containing JSON object
- In Python, JSON exists as a string

```
p = '{"name": "Bob", "languages": ["Python", "Java"]}'
```

- It's also common to store a JSON object in a file whose content can be read as a string

Parse JSON in Python

- For example,

```
import json #import json library to parse JSON Format

person = '{"name": "Bob", "languages": ["English", "French"]}'
person_dict = json.loads(person)

print( person_dict)

# Output: {'name': 'Bob', 'languages': ['English', 'French']}

print(person_dict['languages'])

# Output: ['English', 'French']
```

Parse JSON in Python

- For example, parse json from file

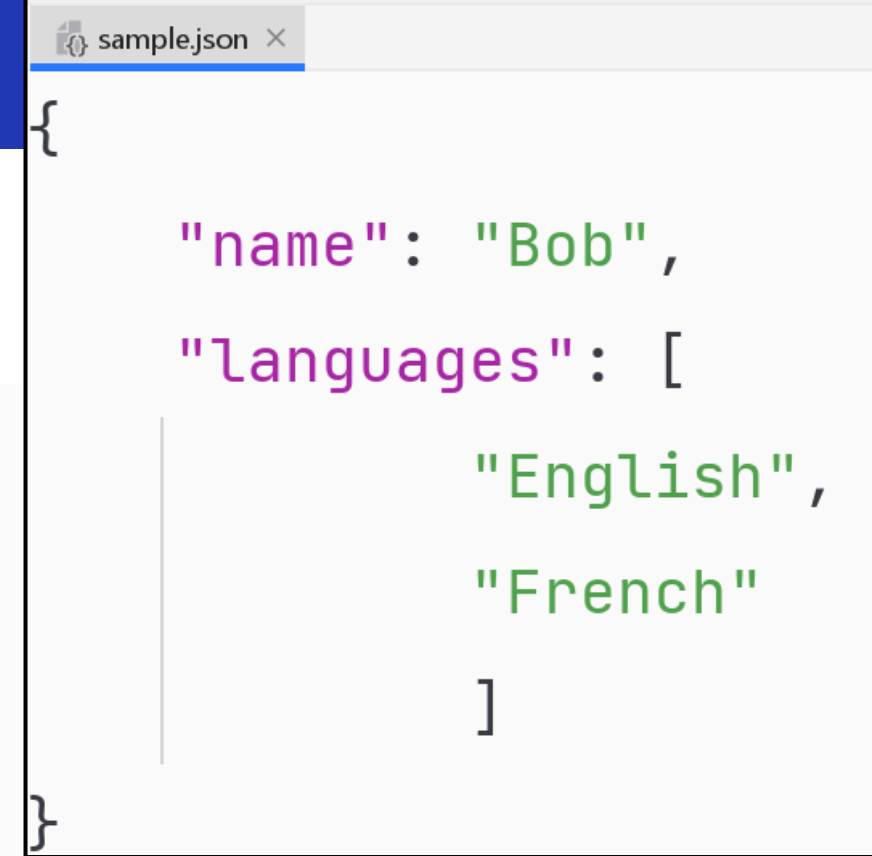
```
import json
```

```
inFile = open('sample.json', 'r')
```

```
data = json.load(inFile)
```

```
print(data)
```

```
# Output: {'name': 'Bob', 'languages': ['English', 'French']}
```



```
{  
    "name": "Bob",  
    "languages": [  
        "English",  
        "French"  
    ]  
}
```

Parse JSON in Python

- `json.load()` takes a file object and returns the json object
- `json.loads()` method can be used to parse a valid JSON string and convert it into a Python Dictionary

Convert to JSON string

- If you have a Python object, you can convert it into a JSON string by using the `json.dumps()` method
- By setting the `skipkeys` to `True`(default: `False`) we automatically skip the keys that are not of basic type

```
import json

Dictionary = {(1, 2, 3): 'Welcome',
              2: 'to', 3: 'Geeks', 4: 'for', 5: 'Geeks'}

json_string = json.dumps(Dictionary, skipkeys = True)
print(json_string)

# {"2": "to", "3": "Geeks", "4": "for", "5": "Geeks"}
```

Convert to JSON string

- If you have a Python object, you can convert it into a JSON string by using the `json.dumps()` method
- By setting the `skipkeys` to `True`(default: `False`) we automatically skip the keys that are not of basic type

```
import json

Dictionary = {(1, 2, 3): 'Welcome',
               2: 'to', 3: 'Geeks', 4: 'for', 5: 'Geeks'}

json_string = json.dumps(Dictionary, skipkeys = False)
# TypeError: keys must be str, int, float, bool or None, not tuple
print(json_string)
```

Convert to JSON string

- The **indent** parameter allows us to format the JSON array elements and object members in a more organized manner.
- A positive integer **indent** represents the number of spaces per level that should be used to indent the content.
- An indent level of 0 or negative will only insert newlines
- None (the default) selects the most compact representation.

Convert to JSON string

```
import json

content ={'name': 'Le Thanh Tung',
         'info': {
             'age': 18}
        }

json_string = json.dumps(content)
print(json_string)

# {"name": "Le Thanh Tung", "info": {"age": 18}}
```

```
import json

content ={'name': 'Le Thanh Tung',
         'info': {
             'age': 18}
        }

json_string = json.dumps(content, indent = 4)
print(json_string)

# {
#     "name": "Le Thanh Tung",
#     "info": {
#         "age": 18
#     }
# }
```

Convert to JSON string

- Write a python program to find the student who has the highest score from the student list in JSON file.
- The format of input file is presented as follows:

```
{  
    "2212001": {  
        "name": "Nguyen Van A",  
        "age": 18,  
        "score": 9.2  
    },  
    "2212002": {  
        "name": "Tran Thi B",  
        "age": 18,  
        "score": 10.0  
    }  
}
```


THANK YOU
for YOUR ATTENTION