

Programming with Python


Sets and Dictionaries

Sets

- An unordered collection with no duplicate elements
- Supports
 - membership testing
 - eliminating duplicate entries
 - Set operations: union, intersection, difference, and symmetric difference.

Sets

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']  
>>> fruits = set(basket)
```



Create a set from
a sequence

Set Operations

```
>>> A=set('acads')
>>> B=set('institute')
>>> A
{'a', 's', 'c', 'd'}
>>> B
{'e', 'i', 'n', 's', 'u', 't'}
```

Set functions

- `add()` `# thisset.add("abc")`
- `update()` `# thisset.update(thatset)`
The object in the `update()` method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.)
- `remove()` `# thisset.remove("abc")`
- `discard()` `# thisset.discard("abc")`
- `thisset.pop()`
- `thisset.clear()`
- `del thisset`

Method	Description
<code>add()</code>	Adds an element to the set
<code><u>clear()</u></code>	Removes all the elements from the set
<code><u>copy()</u></code>	Returns a copy of the set
<code><u>difference()</u></code>	Returns a set containing the difference between two or more sets
<code><u>difference update()</u></code>	Removes the items in this set that are also included in another, specified set
<code><u>discard()</u></code>	Remove the specified item
<code><u>intersection()</u></code>	Returns a set, that is the intersection of two other sets
<code><u>intersection update()</u></code>	Removes the items in this set that are not present in other, specified set(s)
<code><u>isdisjoint()</u></code>	Returns whether two sets have a intersection or not
<code><u>issubset()</u></code>	Returns whether another set contains this set or not
<code><u>issuperset()</u></code>	Returns whether this set contains another set or not
<code><u>pop()</u></code>	Removes an element from the set
<code><u>remove()</u></code>	Removes the specified element
<code><u>symmetric difference()</u></code>	Returns a set with the symmetric differences of two sets
<code><u>symmetric difference update()</u></code>	inserts the symmetric differences from this set and another
<code><u>union()</u></code>	Return a set containing the union of sets
<code><u>update()</u></code>	Update the set with the union of this set and others

Which of the following are true for objects of Python's set type:

- A. The order of elements in a set is significant.
- B. Sets are mutable.
- C. A given element can't appear in a set more than once.
- D. A set may contain elements that are mutable.

You have a set `s` defined as follows: `s = {100, 200, 300}`

Which one of the following statements does **not** correctly produce the union of `s` and the set `{300, 400, 500}`:

- A. `s | set([300, 400, 500])`
- B. `s | {300, 400, 500}`
- C. `s.union({300, 400, 500})`
- D. `s.union([300, 400, 500])`
- E. `s | [300, 400, 500]`

What is the result of this statement:

```
{'b', 'a', 'r'} & set('qux')
```

A. {}

B. {'b', 'r', 'a'}

C. set()

D. {'q', 'r', 'x', 'u', 'b', 'a'}

What is the result of this statement:

$$\{1, 2, 3, 4, 5\} - \{3, 4\} \wedge \{5, 6, 7\}$$

A. $\{3, 4, 5, 6, 7\}$

B. `set()`

C. $\{1, 2, 6, 7\}$

D. $\{1, 2\}$

Suppose a set `s` is defined as follows:

```
s = {'foo', 'bar', 'baz', 'qux'}
```

Which of the following remove element 'bar' from `s`:

A. `s.difference_update({'bar'})`

B. `s.discard('bar')`

C. `s &= {'foo', 'baz', 'qux'}`

D. `del s['bar']`

E. `s -= {'bar'}`

F. `s.pop()`

Dictionaries

- Unordered set of *key:value* pairs,
- Keys have to be unique and immutable
- Key:value pairs enclosed inside curly braces {...}
- Empty dictionary is created by writing {}
- Dictionaries are mutable
 - add new key:value pairs,
 - change the pairing
 - delete a key (and associated value)

Operations on Dictionaries

Operation	Meaning
<code>len(d)</code>	Number of key:value pairs in d
<code>d.keys()</code>	List containing the keys in d
<code>d.values()</code>	List containing the values in d
<code>k in d</code>	True if key k is in d
<code>d[k]</code>	Value associated with key k in d
<code>d.get(k, v)</code>	If k is present in d, then d[k] else v
<code>d[k] = v</code>	Map the value v to key k in d (replace d[k] if present)
<code>del d[k]</code>	Remove key k (and associated value) from d
<code>for k in d</code>	Iterate over the keys in d

Operations on Dictionaries

```
>>> capital = {'India':'New Delhi', 'USA':'Washington DC', 'France':'Paris', 'Sri Lanka':'Colombo'}
```


Operations on Dictionaries

Dictionary Construction

- The **dict** constructor: builds dictionaries directly from *sequences of key-value pairs*

```
>>> airports=dict([('Mumbai', 'BOM'), ('Delhi', 'Del'), ('Chennai', 'MAA'), ('Kolkata', 'CCU')])
>>> airports
{'Kolkata': 'CCU', 'Chennai': 'MAA', 'Delhi': 'Del', 'Mumbai': 'BOM'}
```

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

Which of the following are true of Python dictionaries:

- A. All the keys in a dictionary must be of the same type.
- B. Dictionaries can be nested to any depth.
- C. Items are accessed by their position in a dictionary.
- D. Dictionaries are mutable.
- E. A dictionary can contain any object type except another dictionary.
- F. Dictionaries are accessed by key.

Write the Python code that prints out Captain Kirk's email address from the employee dictionary below:

```
employee = {  
    "id": 1701,  
    "name": "James T Kirk",  
    "email": "jtkirk@starfleet.com",  
    "position": "captain",  
}
```

Add your code below:

```
    print(employee["email"])
```

Which Python code could replace the ellipsis (...) below to get the following output? (Select all that apply.)

```
>>> captains = {  
>>>     "Enterprise": "Picard",  
>>>     "Voyager": "Janeway",  
>>>     "Defiant": "Sisko",  
>>> }
```

```
>>> ...
```

```
Enterprise Picard  
Voyager Janeway  
Defiant Sisko
```

- A.

```
for ship in captains:  
    print(ship, captains[ship])
```
- B.

```
for ship, captain in captains.items():  
    print(ship, captain)
```
- C.

```
for ship in captains:  
    print(ship, captains)
```

Which of the following statements about this nested dictionary are **true**? (Select all that apply.)

```
states = {  
    "California": {  
        "capital": "Sacramento",  
        "flower": "California Poppy"  
    },  
    "New York": {  
        "capital": "Albany",  
        "flower": "Rose"  
    },  
    "Texas": {  
        "capital": "Austin",  
        "flower": "Bluebonnet"  
    }  
}
```

- A. If you want to access the nested values, then you need to split the dictionary.
- B. You can access the nested values directly by chaining square brackets ([]).
- C. The values for "California", "New York", and "Texas" are all dictionaries.
- D. The syntax isn't valid.

Which of the following lines of code will create an empty dictionary named captains?

A. `type(captains)`

B. `captains = {dict}`

C. `captains = {}`

D. `captains.dict()`

Now you have your empty dictionary named `captains`. It's time to add some data!

Specifically, you want to add the key-value pairs `"Enterprise": "Picard"`, `"Voyager": "Janeway"`, and `"Defiant": "Sisko"`.

Which of the following code snippets will successfully add these key-value pairs to the existing `captains` dictionary?

- A.

```
captains["Enterprise"]: "Picard"  
captains["Voyager"]: "Janeway"  
captains["Defiant"]: "Sisko"
```
- B.

```
captains{"Enterprise" = "Picard"}  
captains{"Voyager" = "Janeway"}  
captains{"Defiant" = "Sisko"}
```
- C.

```
captains = {  
    "Enterprise": "Picard",  
    "Voyager": "Janeway",  
    "Defiant": "Sisko",  
}
```
- D.

```
captains["Enterprise"] = "Picard"  
captains["Voyager"] = "Janeway"  
captains["Defiant"] = "Sisko"
```