PYTHON COLLECTIONS

softserve

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

Lists

- Ordered collection of objects; array
- List items need not have the same type
- Lists are mutable; can be changed in-place
- Lists are dynamic; size may be changed
- Same operators as for strings
- Lists have a set of built-in methods

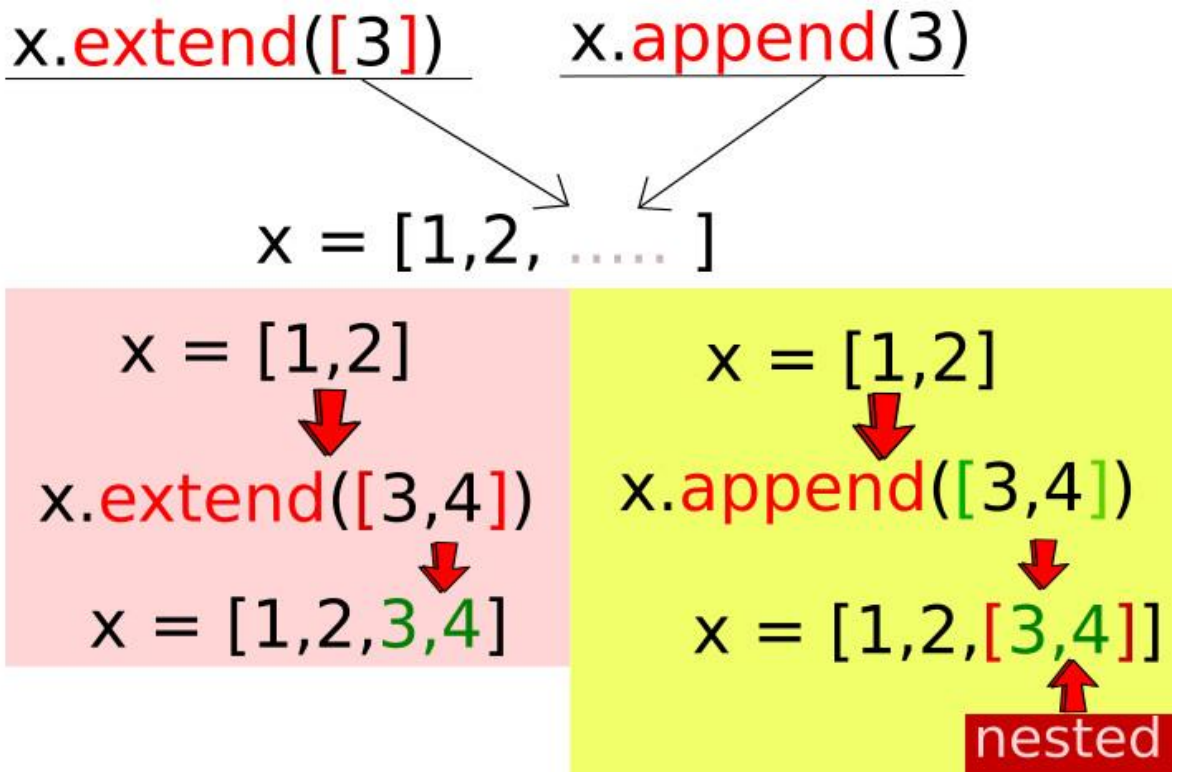
(some of them change the list in place): append(), insert(), pop(), reverse(), sort(), etc.

Python List Methods

Methods that are available with list object in Python programming are tabulated below. They are accessed as `list.method()`.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a shallow copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes and returns the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of items in the list
sort()	Sort items in a list in ascending order

Python List Methods



Built-in Functions with List

Built-in functions are commonly used with list to perform different tasks.

Function	Description
all()	Return True if all elements of the list are true (or if the list is empty).
any()	Return True if any element of the list is true. If the list is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of list as a tuple.
len()	Return the length (the number of items) in the list.
list()	Convert an iterable (tuple, string, set, dictionary) to a list.
max()	Return the largest item in the list.
min()	Return the smallest item in the list
sorted()	Return a new sorted list (does not sort the list itself).
sum()	Return the sum of all elements in the list.

Syntax

```
all(iterable)
```

The **all()** function returns **True** if **all items** in an iterable are **true**, otherwise it returns False.

If the iterable object is empty, the **all()** function also returns True.

Parameter	Description
iterable	An iterable object (list, tuple, dictionary)

```
mylist = [True, True, True]
x = all(mylist)
print(x)
######################
mylist = [0, 1, 1]
x = all(mylist)
print(x)
######################
mydict = {0 : "Apple", 1 : "Orange"}
x = all(mydict)
print(x)
```

```
True
```

```
False
```

```
False
```

Note: When used on a dictionary, the **all()** function checks if all the ***keys*** are **true**, **not the *values***.

softserve

Syntax

any(*iterable*)

The **any()** function returns **True** if **any item** in an iterable are **true**, otherwise it returns False.

If the iterable object is **empty**, the **any()** function will return **False**.

Parameter	**Description**
iterable	An iterable object (list, tuple, dictionary)

```
mylist = [False, True, False]
x = any(mylist)
print(x)
######################
mytuple = (0, 1, False)
x = any(mytuple)
print(x)
######################
mydict = {0 : "Apple", 1 : "Orange"}
x = any(mydict)
print(x)
```

```
True
```

```
True
```

```
True
```

Note: When used on a dictionary, the **any()** function checks if **any** of the ***keys*** are **true**, **not the *values***.

softserve

Syntax

```
enumerate(iterable, start)
```

The **enumerate()** function takes a collection (e.g. a tuple) and returns it as an enumerate object.

The **enumerate()** function adds a counter as the key of the enumerate object.

Parameter	Description
iterable	An iterable object
start	A Number. Defining the start number of the enumerate object. Default 0

```
x = ['apple', 'banana', 'cherry']
y = enumerate(x)
print(list(y))
```

```
[(0, 'apple'), (1, 'banana'), (2, 'cherry')]
```

softserve

List Comprehension: Elegant way to create new List

List comprehension consists of an expression followed by **for statement** inside square brackets **[]**

```
(values) = [ (expression) for (value) in (collection) ]

# Transforms into:

(values) = []
for (value) in (collection):
    (values).append( (expression) )
```

softserve

List Comprehension: Elegant way to create new List

List comprehension(filter) consists of an expression followed by **for statement** inside square brackets **[]**

```
values = [expression for value in collection if condition]

# Transforms into:

values = []
for value in collection:
    if condition:
        values.append(expression)
```

List Comprehension: Elegant way to create new List

List comprehension consists of an expression followed by **for statement** inside square brackets **[]**

```
pow2 = [2 ** x for x in range(10)]
print(pow2)

# Output: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

=

```
pow2 = []
for x in range(10):
   pow2.append(2 ** x)
```

```
>>> pow2 = [2 ** x for x in range(10) if x > 5]
>>> pow2
[64, 128, 256, 512]
>>> odd = [x for x in range(20) if x % 2 == 1]
>>> odd
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
>>> [x+y for x in ['Python ','C '] for y in ['Language','Programming']]
['Python Language', 'Python Programming', 'C Language', 'C Programming']
```

softserve

Iterating Through a List

```
for fruit in ['apple','banana','mango']:
        print("I like",fruit)
```

```
fruit = ['apple','banana','mango']
i = 0
for i in range(len(fruit)):
    print("I like",fruit[i])
```

Tuples

A tuple consists of a number of values separated by commas

Same as list, except immutable

Once created, can't be changed

Some functions return tuples

Tuples, like strings, are immutable: it is not possible to assign to the individual items of a tuple

```
>>> t = (1, 3, 2)
>>> t[1]
3
>>> (a, b, c) = t
>>> a
1
>>> b
3
>>> a, b, c
(1, 3, 2)
>>>
>>> a, b = b, a
>>> a, b
(3, 1)
```

```
>>> r = list(t)   # convert tuple to a list
>>> r
[1, 3, 2]
>>> tuple(r)      # reverse conversion
(1, 3, 2)
```

softserve

Python Tuple Methods

count(x)	Return the number of items that is equal to x
index(x)	Return index of first item that is equal to x

Syntax

list.count(*value*)

The **count()** method returns the number of elements with the specified value.

Parameter	Description
value	Required. Any type (string, number, list, tuple, etc.). The value to search for.

```
fruits = ("apple", "banana", "cherry“)
x = fruits.count("cherry")
print(x)
##########################
fruits = (1, 4, 2, 9, 7, 8, 9, 3, 1)
x = fruits.count(9)
print(x)
```

=

```
1
```

```
2
```

softserve

Syntax

list.index(*elment*)

The **index()** method returns the position at the first occurrence of the specified value.

Parameter	Description
elment	Required. Any type (string, number, list, etc.). The element to search for

```
fruits = ('apple', 'banana', 'cherry')
x = fruits.index("cherry")
print(x)
##########################
fruits = (4, 55, 64, 32, 16, 32)
x = fruits.index(32)
print(x)
```

=

```
2
```

```
3
```

Note: The **index()** method only returns the ***first*** occurrence of the value.

softserve

Built-in Functions with Tuple

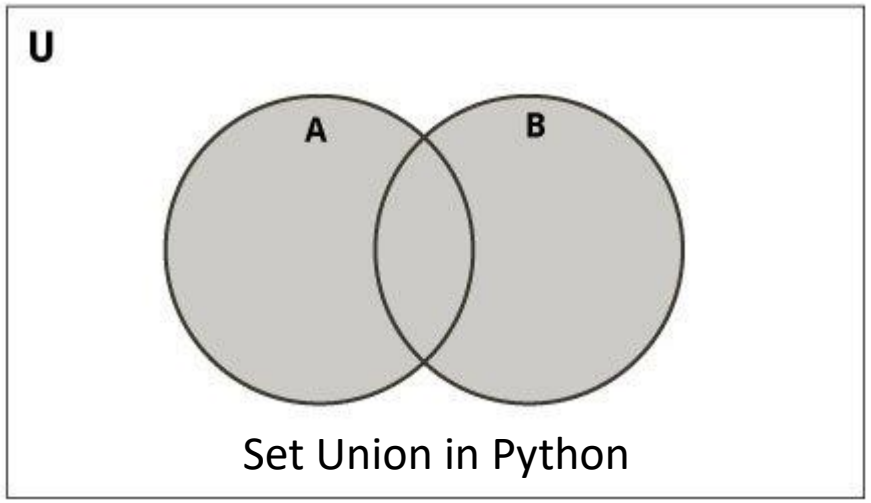
Function	Description
all()	Return True if all elements of the tuple are true (or if the tuple is empty).
any()	Return True if any element of the tuple is true. If the tuple is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of tuple as pairs.
len()	Return the length (the number of items) in the tuple.
max()	Return the largest item in the tuple.
min()	Return the smallest item in the tuple
sorted()	Take elements in the tuple and return a new sorted list (does not sort the tuple itself).
sum()	Retrun the sum of all elements in the tuple.
tuple()	Convert an iterable (list, string, set, dictionary) to a tuple.

softserve

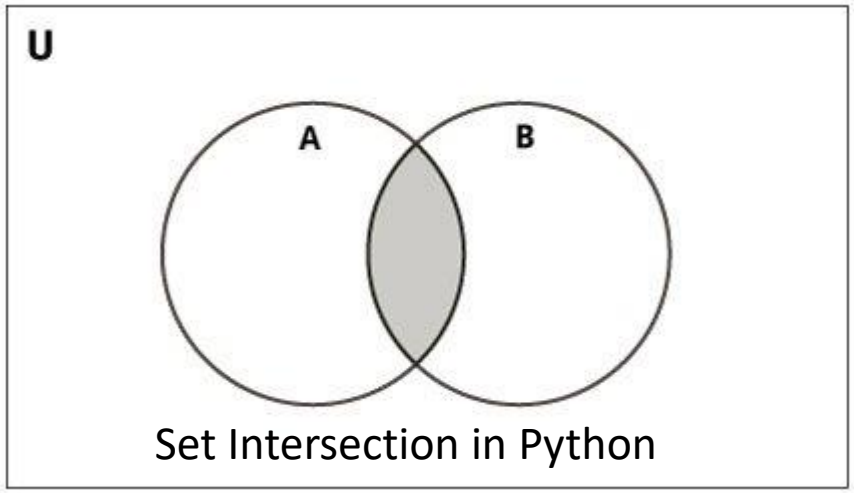
Set and Frozenset

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from the set if it is a member.
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns True if two sets have a null intersection
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
pop()	Removes and returns an arbitrary set element. Raises KeyError if the set is empty
remove()	Removes an element from the set. If the element is not a member, raises a KeyError
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others

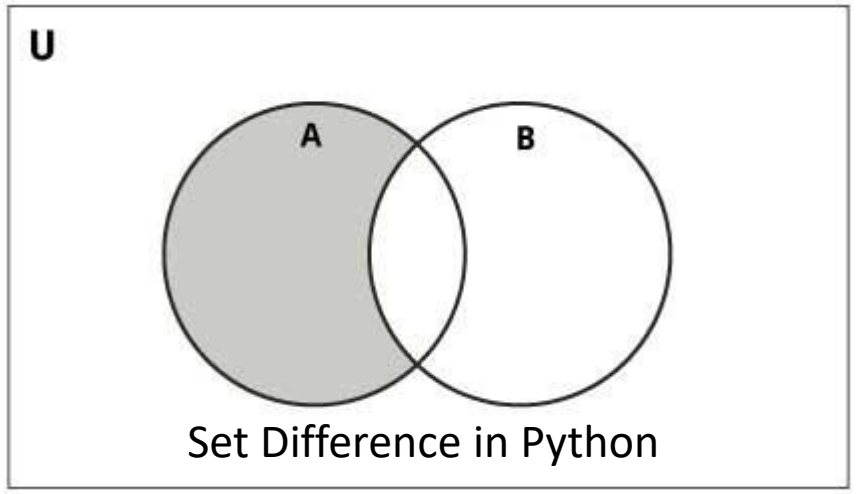
Set and Frozenset Methods



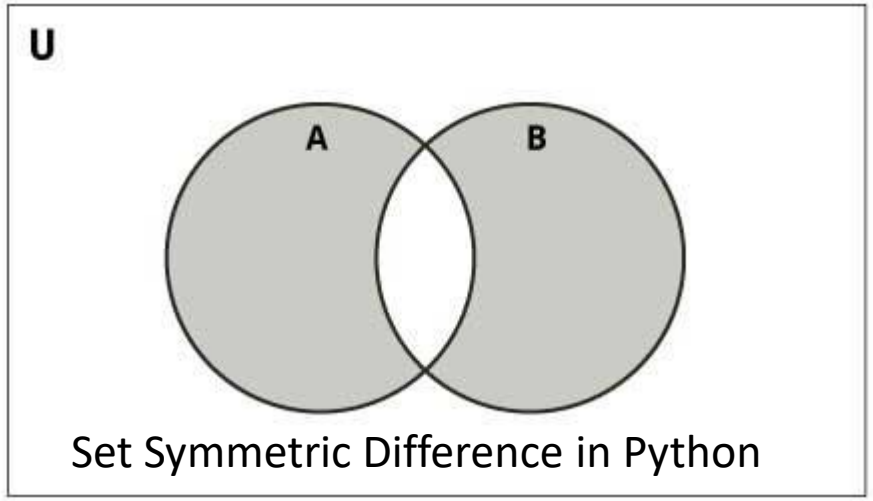
Set Union in Python



Set Intersection in Python



Set Difference in Python



Set Symmetric Difference in Python

softserve

Dictionary

An unordered collection of key/value pairs

Each key maps to a value

Also called "mapping", "hash table" or "lookup table"

The key is:

Usually an integer or a string

Should (must!) be an immutable object

Any key occurs at most once in a dictionary!
The value may be any object:

Values may occur many times

Use command 'del' to get rid of stuff:

del h['hello']

```
>>> h = {'key1': 12, 'python': 'word'}
>>> h['key']
Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    h['key']
KeyError: 'key'
>>> h['key1']
12
>>> h.has_key('key1')
True
>>>
>>> h['hello'] = 'world'
>>> h
{'python': 'word', 'key1': 12, 'hello': 'world'}
>>> h['hello'] = 'PC'
>>> h
{'python': 'word', 'key1': 12, 'hello': 'PC'}
```

softserve

Python Dictionary Methods

Method	Description
clear()	Remove all items form the dictionary.
copy()	Return a shallow copy of the dictionary.
fromkeys(seq[, v])	Return a new dictionary with keys from seq and value equal to v(defaults to None).
get(key[,d])	Return the value of key. If key doesnot exit, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(key[,d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[,d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Return a new view of the dictionary's values

softserve

Built-in Functions with Dictionary

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.
cmp()	Compares items of two dictionaries.
sorted()	Return a new sorted list of keys in the dictionary.

Python Dictionary Comprehension

Dictionary comprehension consists of an expression pair **(key: value)** followed by for statement inside **curly braces {}**

```
squares = {x: x*x for x in range(6)}
print(squares)

# Output: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

=

```
squares = {}
for x in range(6):
   squares[x] = x*x
```

Using **If** statement in comprehension

```
odd_squares = {x: x*x for x in range(11) if x%2 == 1}
print(odd_squares)

# Output: {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
```

softserve

Iterating Through a Dictionary

```
d = {'name': 'Vasyl', 'surname': 'Bilan','id': '1', 'task': 'run application'}

for key in d:
    print("student {} = {}".format(key, d[key]))


for key, val in d.items():
    print("{} = {} .".format(key, val))


for key in d.keys():
    print("student {} = {}".format(key, d[key]))


for val in d.values():
    print("student {} = {}".format("?", val))
```


TO THE FUTURE
softserve