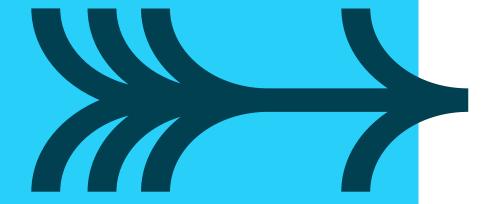


Collections

Module 12



COLLECTIONS



Contents

- Python types reminder
- Generic built-in functions
- Useful tuple operations
- Python lists
- Tuple and list slicing
- Adding items to a list
- Removing items from a list
- Sorting
- List methods
- Sets
- Set operators
- Python dictionaries
- Dictionary methods
- View objects

Python types - reminder

Built in sequence types:

```
Strings (str)
    'Norwegian Blue', "Mr. Khan's bike"
Lists (list)
    ['Cheddar', ['Camembert', 'Brie'], 'Stilton']
Tuples (tuple)
    (47, 'Spam', 'Major', 683, 'Ovine Aviation')
```

- We also have bytearray (read/write) and bytes (read only)
- Used for binary data

Not all collections are sequences

- A set is an *unordered* collection of *unique* objects
- Dictionaries are a special form of set

```
{'Totnes':'Barber', 'BritishColumbia':'Lumberjack'}
```

Generic built-in functions

- Most iterables support the len, min, max, and sum built-in functions
- len Number of elements
- min
 Minimum value
- max
 Maximum value
- sum
 Numeric summation (not string or byte objects)
- String and byte objects do not support sum
- Dictionaries implement min, max, and sum on keys
- The sum built-in will raise a TypeError if the item is not a number

Useful tuple operations

Swap references

```
a, b = b, a
```

Set values from a numeric range

```
Gouda, Edam, Caithness = range(3)
```

0 1 2



Repeat values

```
mytuple = 'a', 'b', 'c'
another = mytuple * 4
```

```
('a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c', 'a', 'b', 'c')
```

Be careful of single values and the trailing comma

```
thing = ('Hello')
print(type(thing))

thing = ('Hello',
print(type(thing))

<class 'str'>
<class 'tuple'>
```

Python lists

- Python lists are similar to arrays in other languages*
- Can contain objects of any type
- Multi-dimensional lists are just lists containing references to other lists
- Create a list using list(object) or []
- Access list elements using [] or by method calls
- Indexes on the left start at zero
- Indexes on the right start at -1

```
cheese = ['Cheddar', 'Stilton', 'Cornish Yarg']
print(cheese[1])
cheese[-1] = 'Red Leicester'
print(cheese)

Stilton
['Cheddar', 'Stilton', 'Red Leicester']
```

Multiply operator * can also be applied to a list

Tuple and list slicing

- Slice by start and end position
- Counting from zero on lhs, from -1 on rhs

List elements may be removed using del

Extended iterable unpacking

- Python 3 allows unpacking to a wildcard
- Only allowed on the left-side of an assignment

```
py3
```

```
mytuple = 'eggs', 'bacon', 'spam', 'tea'
x, y, z = mytuple

ValueError: too many values to unpack
```

Adding items to a list

On the left

```
cheese[:0] = ['Cheshire', 'Ilchester']
```

On the right

```
cheese += ['Oke', 'Devon Blue']
cheese.extend(['Oke', 'Devon Blue'])
Same effect
```

append can only be used for one item

```
cheese.append('Oke')
```

Anywhere

```
cheese = ['Cheddar', 'Stilton', 'Cornish Yarg']
cheese.insert(2, 'Cornish Brie')
cheese[2:2] = ['Cornish Brie']
print(cheese)

['Cheddar', 'Stilton', 'Cornish Brie', 'Cornish Yarg']
```

Removing items by position

Use pop (index)

- The index number is optional, default -1 (rightmost item)
- Returns the deleted item

```
cheese = ['Cheddar', 'Stilton', 'Cornish Yarg']
saved = cheese.pop(1)
print("Saved1:", saved,", Result:", cheese)
saved = cheese.pop()
print("Saved2:", saved,", Result:", cheese)
```

```
Saved1: Stilton , Result: ['Cheddar', 'Cornish Yarg']
Saved2: Cornish Yarg , Result: ['Cheddar']
```

Remember that del may also be used

- Does not return the deleted item
- May delete more than one item by using a slice

Removing list items by content

Use the remove method

Removes the leftmost item matching the value

Raises an exception if the item is not found

Exceptions will be handled later...

Sorting

sorted built-in and sort method

- sorted can sort any iterable (often a sequence)
- sorted returns a sorted list regardless of the original type
- sort sorts a list in-place
- Both have the following optional named parameters

```
key=sort_key Function which takes a single argument
```

reverse=True Default is False

```
nums = ['1001', '34', '3', '77', '42', '9', '87']
newstr = sorted(nums)
newnum = sorted(nums, key=int)
```

```
newstr: ['1001', '3', '34', '42', '77', '87', '9']
newnum: ['3', '9', '34', '42', '77', '87', '1001']
```

Miscellaneous list methods

Count

list.count('value') Return the number of occurrences of 'value'

Index

list.index('value') Return index position of leftmost 'value'

Reverse

list.reverse() Reverse a list in place

```
cheese = ['Cheddar', 'Cheshire', 'Stilton', 'Cheshire']
print(cheese.count('Cheshire'))
print(cheese.index('Cheshire'))
cheese.reverse()
print(cheese)

2
1
['Cheshire', 'Stilton', 'Cheshire', 'Cheddar']
```

List methods

list.append(item)	Append item to the end of list	
list.clear()	Remove all items from <i>list</i> (3.3)	
list.count(item)	Return number of occurrences of item	
list.extend(items)	Append items to the end of list (as +=)	
list.index(item, start, end)	Return the position of item in the list	
list.insert(position, item)	Insert item at position in list	
list.pop()	Remove and return last item in list	
list.pop(position)	Remove and return item at position in list	
list.remove(item)	Remove the first item from the list	
list.reverse()	Reverse the <i>list</i> in-place	
list.sort()	Sort the <i>list</i> in-place - arguments are the same as sorted()	

Sets

A set is an unordered container of object references

- A set is *mutable*, a frozenset is *immutable*
- Set items are unique

Creating a set

Any iterable type may be used

```
s1 = {5, 6, 7, 8, 5}
print(s1)

s2 = set([9, 10, 11, 12, 9])
print(s2)

s3 = frozenset([9, 10, 11, 12, 9])
print(s3)
```

```
py3
```

```
Python versions >= 2.7
Not <= 2.6
```

Python versions >= 2.4

```
{8, 5, 6, 7}
{9, 10, 11, 12}
frozenset({9, 10, 11, 12})
```

The format when printing a set changed at Python 3

Set methods

Add using the add method, remove using remove

```
s4 = {23, 42, 66, 123}
s5 = {56, 27, 42}
print("{:20} {:20}".format(s4, s5))

s4.remove(123)
s5.add(123)
print("{:20} {:20}".format(s4, s5))
{66, 123, 42, 23}
{56, 42, 27}
{66, 42, 23}
```

Other set methods:

• Len	Return the	number of e	lements in the set

- discard Remove element if present
- pop Remove and return the next element from the set
- clear
 Remove all elements

Exploiting sets

How do I remove duplicates from a list?

But we lose the original order

```
['Cornish Yarg', 'Cheshire', 'Cheddar', 'Stilton', 'Oke']
```

How do I remove several items from a list?

```
['Cornish Yarg', 'Cheshire', 'Cheddar']
```

Set operators

Includes set operators and method calls

Operator	Method	Returns a new set containing
&	s6.intersection(s7)	Each item that is in both sets
1	s6.union(s7)	All items in both sets
-	s6.difference(s7)	Items in s6 not in s7
^	s6.symmetric_difference(s7)	Items that occur in one set only

```
s6 = {23, 42, 66, 123}
s7 = {123, 56, 27, 42}

print(s6 & s7)
print(s6 | s7)
print(s6 - s7)
print(s6 ^ s7)
{66, 27, 42, 23, 56, 123}
print(s6 ^ s7)
{66, 23, 56, 27}
```

py3

Python dictionaries

Dictionaries are similar to sets but are accessed by keys

```
    Constructed from {}
        varname = {key1:object1, key2:object2, key3:object3,...}
    Or using dict()
        varname = dict(key1=object1, key2=object2, key3=object3,...)
    Accessed by key
        A key is usually a text string, or anything that yields a text string varname[key] = object
```

Dictionary values

Objects stored can be of any type

- Lists, tuples, other dictionaries, etc...
- Can be accessed using multiple indexes or keys in []
- Add a new value just by assigning to it

```
FR: ['Paris', 'Lyon', 'Bordeaux', 'Toulouse']
US: ['Miami', 'Springfield', 'New York', 'Boston']
UK: ['London', 'Wigan', 'Macclesfield', 'Bolton']
```

Removing items from a dictionary

To remove a single key/value pair:

- del dict[key]
- Raises a KeyError exception if the key does not exist
- dict.pop(key[,default])
- Returns *default* if the key does not exist

```
>>> fred={}
>>> del fred['dob']
Traceback (most recent call last):
   File "<pyshell#11>", line 1, in <module>
        del fred['dob']
KeyError: 'dob'
>>> fred.pop('dob', False)
False
```

- Also:
- dict.popitem() removes the next key/value pair used in iteration
- dict.clear() removes all key/value pairs from the dictionary

Dictionary methods

dict.clear()	Remove all items from dict
dict.copy()	Return a copy of <i>dict</i>
dict.fromkeys(seq[,value])	Create a new dictionary from seq
dict.get(key[,default])	Return the value for <i>key</i> , or <i>default</i> if it does not exist
dict.items()	Return a view of the key-value pairs
dict.keys()	Return a view of the keys
dict.pop(key[,default])	Remove and return <i>key's</i> value, else return <i>default</i>
dict.popitem()	Remove the next item from the dictionary
dict.setdefault(key[,default])	Add key if it does not already exist
dict.update(dictionary)	Merge another dictionary into dict.
dict.values()	Return a view of the values

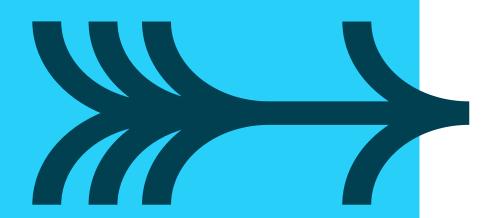
View objects - examples

May be used in iteration

To store as a list

In set operations

SUMMARY



- Lists are like arrays in other languages
- Tuples are "immutable"
- But can contain variables
- Slice lists and tuples using object[start:end+1]
- Sets store unordered unique objects
- May be joined, along with other operations
- Dictionaries store objects accessed by key
- Keys are unique
- Not ordered