Python  Programming

Day 5: Sets and Dictionaries

Set and Dictionary

Set

Ordered Set

Dictionary

Color and symbol meaning

Hint

Preferred

Student's activity

Code to run

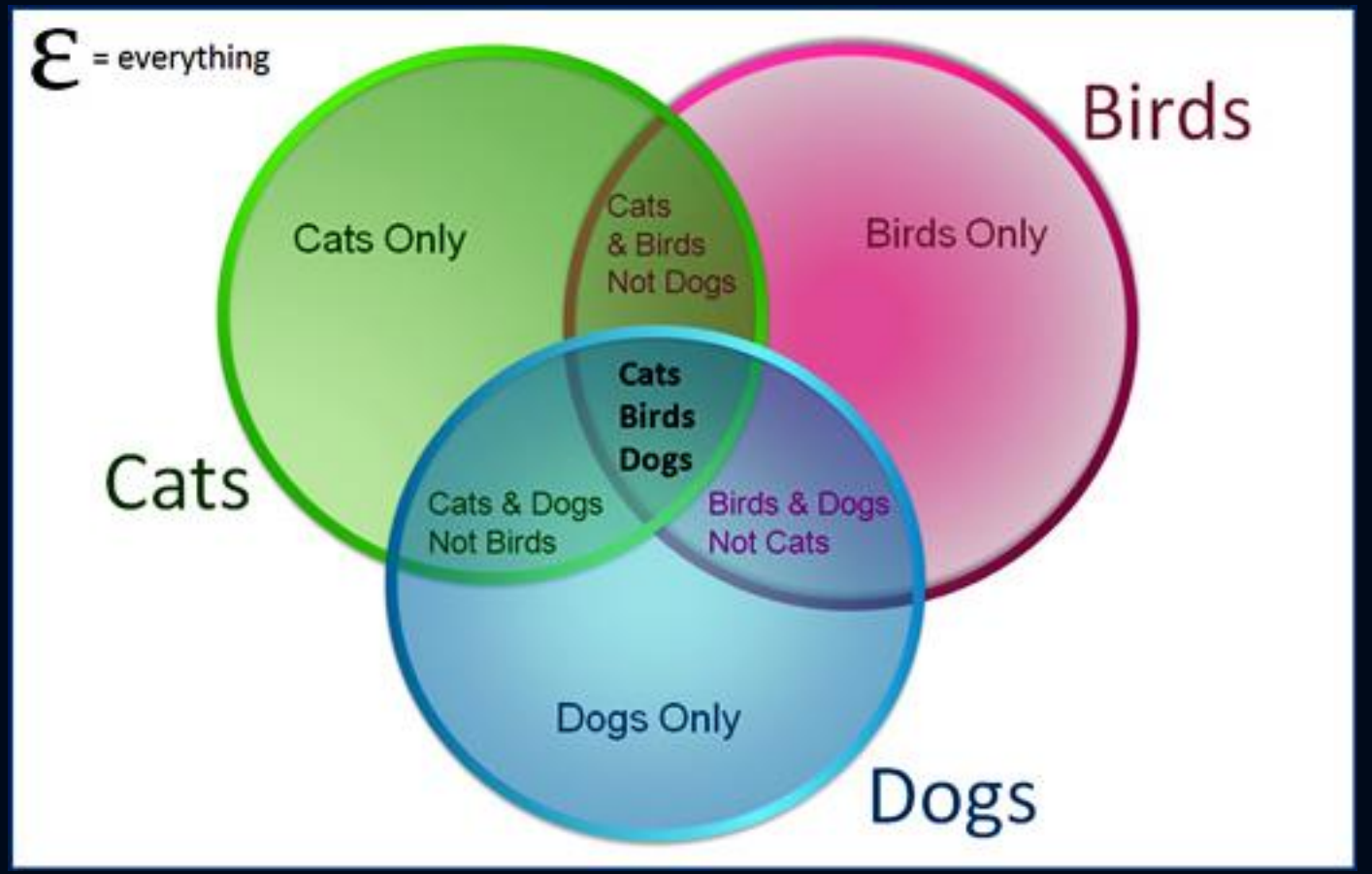
	Python Keyword
	In-built modules
	Strings
	Output

Set

A set is an unordered collection of unique and immutable objects.

A set is a collection that does not allow repetitions

Sets are mutable sequences that contains immutable objects (str, int, etc.)



Set

X = set( )  Declares an empty set

We convert a list to a set to remove duplicate items

Unlike sequences, sets provide no indexing or slicing operations.

```
>>> numberList =
[2,1,3,2,5,5,2]
>>> aSet = set(numberList)
>>> aSet
{1, 2, 3, 5}
```


Set

What is the cause of the error?

```
>>> mySet = set(['One', [2, 3, 4], 'Five'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```


Set

What is the cause of the error?
Because a set can only contain immutable objects

```
>>> mySet = set(['One', [2, 3, 4], 'Five'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```


Set Methods

item	Description
len(s)	Returns the number of items in s
s.copy( )	Makes a copy of s
s.difference(t)	Set difference. Returns all the items in s but not in t
s.intersection(t)	Intersection. Returns all the items that are both in s and in it
s.isdisjoint(t)	Returns True if s and t have no items in common.
s.issubset(t)	Returns True if s is a subset of t
s.issuperset(t)	Returns True if s is a superset of t
s.union(t)	Union. Returns all items in s or t

Set Operations

Standard mathematical set operations

item	Set function	Description
set1 & set 2	Intersection	AND
set1 | set 2	Union	OR
set1 ^ set 2	Symmetric difference	XOR
set1 - set 2	Difference	In set1 but not in set2
set1 <= set 2	Subset	set2 contains set1
set1 >= set 2	superset	set1 contains set2

Set – Add item

Add item to a set

```
>>> num_set = set ( )
>>> num_set.add(5)
>>> num_set.add(3)
>>> num_set.add(2)
>>> num_set.add(8)
>>> num_set.add(5)
>>> num_set
{8, 2, 3, 5}
```


Set – Remove item

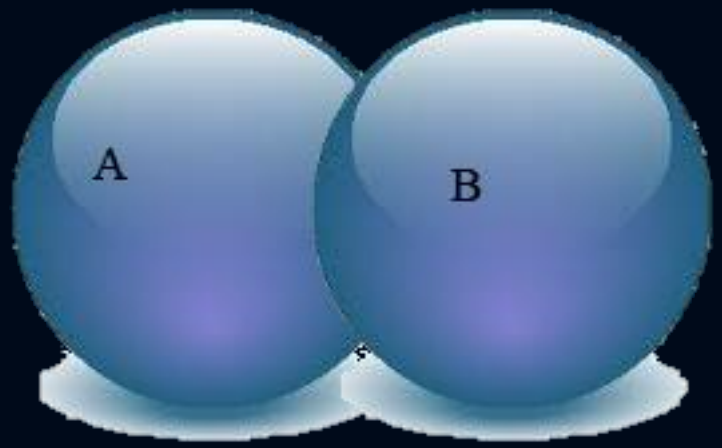
To **remove** item(s) from set.

```
num_set = set([0, 1, 3, 4, 5])
num_set.pop( )
print(num_set)
num_set.pop( )
print(num_set)
```

```
{1, 3, 4, 5}
{3, 4, 5}
```


Set - Union

To create an **Union** of sets.

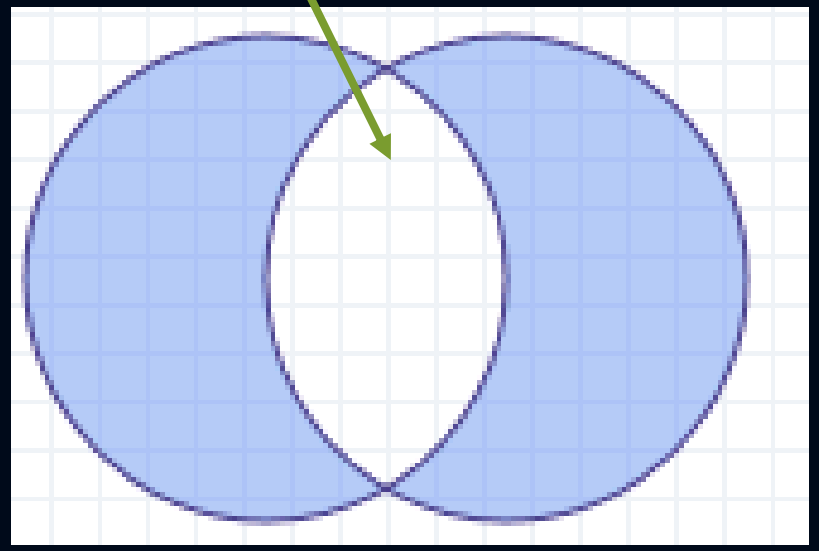



```
setx = set(["green", "blue"])
sety = set(["blue", "yellow"])
seta = setx | sety
print(seta)

{'yellow', 'green', 'blue'}
```

Set - Intersection

To create an intersection of sets.



```
setx = set(["green", "blue"])
sety = set(["blue", "yellow"])
setz = setx & sety
print(setz)

{'blue'}
```


Set – Subset and Superset

To check if two sets are sub or super set.

The conditional operators returns a boolean.

Run the code to get output

```
setx = set(["apple", "mango"])
sety = set(["mango", "orange"])
setz = set(["mango"])
issubset = setx <= sety
print (issubset)
issuperset = setx >= sety
print (issuperset)
issubset = setz <= sety
print (issubset)
issuperset = sety >= setz
print (issuperset)
```

Frozenset

Frozensets are exactly like sets, however the difference is that Frozensets are immutable

Mutable means that, we can add, remove or modify the contents of an object

Frozenset

Create an empty frozen set

mySet = frozenset( )

```
>>> mySet = frozenset(['Apple', 'Banana', 'Mango', 'Orange', 'Apple'])
>>> mySet
frozenset(['Orange', 'Mango', 'Apple', 'Banana'])
>>> type(mySet)
<type 'frozenset'>
```


Frozenset

Adding an element to a frozenset

```
>>> mySet.add('Potato')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```


Class Activity 1

Make a copy of setp below, clear setp and print both sets

setp = set(["Red", "Green"])

Use copy() and clear()

Python  Programming

Dictionary

Dictionary

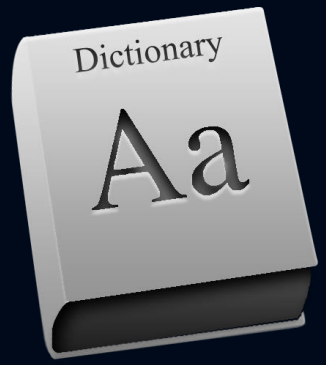
A dictionary in Python is a collection of unordered values accessed by key rather than by index.

List examples of hashable data types.

The keys have to be hashable (immutable)

Dictionary

Python has a built in dictionary type called **dict** which you can use to create dictionaries.



The empty dictionary is denoted { }:

Dictionary

One way to create a dictionary is to start with the empty dictionary and add key-value pairs.

For example, let's create a dictionary to translate English words into Spanish.

```
>>> spanish = dict()
>>> spanish['one'] = 'uno'
>>> spanish['two'] = 'dos'
>>> print (spanish)
{'two': 'dos', 'one': 'uno'}
```


Dictionary

The key-value pairs of the dictionary are separated by commas. Each pair contains a key and a value separated by a colon.

```
>>> print(spanish['two'])
dos
```

The key 'two' yields the value 'dos'.

Dictionary Operations and Methods

item	Description
len(m)	Returns the number of items in m
m[k]	Returns the item of m with key k
m[k] = x	Sets m[k] to x
del m[k]	Removes m[k] from m
k in m	Returns True if k is a key in m.
m.clear()	Removes all items from m.
m.copy()	Makes a copy of m
m.get(k [, v])	Returns m[k] if found; otherwise, returns v.

Dictionary

m.items()	Returns a sequence of (key, value) pairs
m.keys()	Returns a sequence of key values
m.pop(k [, default])	Returns m[k] if found and removes it from m; otherwise, returns default if supplied or raises key error if not
m.popitem()	Removes a random (key, value) pair from m and returns it as a tuple
m.setdefault(k [, v])	Returns m[k] if found; otherwise, returns v and sets m[k] = v.
m.update(b)	Adds all objects from b to m
m.values()	Returns a sequence of all values in m

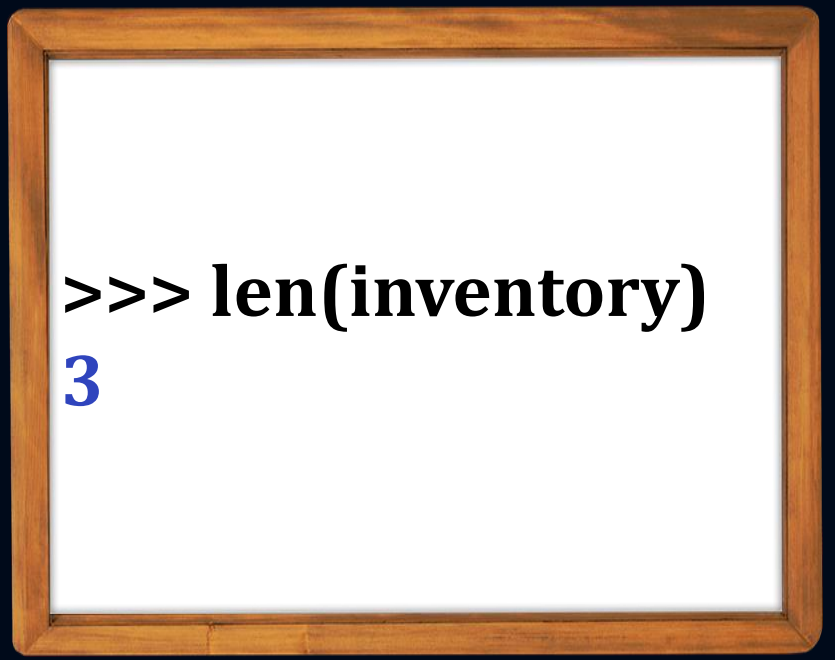
Dictionary operations

The ***del*** statement removes a key-value pair from a dictionary.

```
>>> inventory = {'apples':430,
'bananas':312, 'oranges':525,
'pears':217}
>>> print (inventory)
{'apples': 430, 'oranges': 525,
'pears': 217, 'bananas': 312}
>>> del inventory['pears']
>>> print (inventory)
{'apples': 430, 'oranges': 525,
'bananas': 312}
```


Dictionary operations

The ***len*** function returns the number of key-value pairs



Dictionary operations

The ***keys*** method takes a dictionary and returns a list of its keys.

```
>>> spanish.keys()
dict_keys(['two', 'one'])
```


Dictionary operations

The ***values*** method returns a list of the values in the dictionary

```
>>> spanish.values()
dict_values(['dos', 'uno'])
```


Dictionary operations

The ***items*** method returns both key with value, in the form of a list of tuples — one for each key-value pair

```
>>> spanish.items()
dict_items([('two', 'dos'),
('one', 'uno')])
```


Dictionary operations

The *in* operator returns ‘True’ if the key appears in the dictionary and ‘False’ if otherwise:

```
>>> 'one' in spanish
True
>>> 'deux' in spanish
False
```


Dictionary operations

Looking up a non-existent key in a dictionary causes a runtime error

```
>>> spanish['dog']
Traceback (most recent call last):
  File "<pyshell#138>", line 1, in <module>
    spanish['dog']
KeyError: 'dog'
```


Dictionary Aliasing and copying

Use the ***copy*** method to modify a dictionary and keep a copy of the original.

```
>>> opposites = {'up':'down','right':'wrong','true':'false'}
>>> alias = opposites
>>> copy = opposites.copy()
```


Dictionary Aliasing and copying

If we modify alias, opposites is also changed

```
>>> alias['right'] = 'left'
>>> opposites['right']
'left‘
```


Dictionary Aliasing and copying

If we modify copy, opposites is unchanged:

```
>>> copy['right'] =
'privilege'
>>> opposites['right']
'left‘
```


Counting letters

In previous module, we wrote a function that counted the number of occurrences of a letter in a string.

```
count = 0
for letter in 'banana':
  if letter == 'a':
    count +=1
print (count)
```

A more general version of this problem is to form a histogram of the letters in the string, that is, how many times each letter appears.

Counting letters

Dictionaries provide an elegant way to generate a histogram.

```
>>> letter_counts = { }
>>> for letter in "Mississippi":
        letter_counts[letter] =
letter_counts.get(letter, 0) +1


>>> letter_counts
{'M': 1, 'i': 4, 'p': 2, 's': 4}
```


Counting letters

To display the histogram in alphabetical order

```
>>> letter_counts
{'M': 1, 'i': 4, 'p': 2, 's': 4}
>>> letter_items =
letter_counts.items()
>>> sorted (letter_items)
[('M', 1), ('i', 4), ('p', 2), ('s', 4)]
```


Class Activity 2

Write a Python program to check if a given key already exists in a dictionary.

Python  Programming

Tutorials

Exercise 1:

Write a Python program that accepts a comma separated sequence of words as input and prints the unique words in sorted form (alphanumerically).

Sample Words : red, white, black, red, green, black

Expected Result : black, green, red, white, red

Exercise 2:

Write a Python program to count the number of characters (character frequency) in a string.
Sample String : google.com'
Expected Result : {'o': 3, 'g': 2, '.': 1, 'e': 1, 'l': 1, 'm': 1, 'c': 1}

Exercise 3:

Write a Python program to add a key to a dictionary.
Sample Dictionary : {0: 10, 1: 20}
Expected Result : {0: 10, 1: 20, 2: 30, 3:45}

Exercise 4:

Write a Python program to find common items from two lists.

Exercise 5:

Write a Python script to merge two Python dictionaries.

Exercise 6:

Write a Python program to find common items from two lists.

Exercise 7:

Write a Python program to concatenate following dictionaries to create a new one.

Sample Dictionary:

dic1={1:10, 2:20}

dic2={3:30, 4:40}

dic3={5:50,6:60}

Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

Exercise 8:

Write a Python program to print a dictionary where the keys are numbers between 1 and 15 (both included) and the values are square of keys.

Sample Dictionary

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11: 121, 12: 144, 13: 169, 14: 196, 15: 225}

Exercise 9:

Write a Python program to get the key, value and item in a dictionary.

Next Lecture ...

Python Programming

Day 6: Modules and Packages