Computational Thinking with Programming

@cse_bennett @csebennett

Lecture Contents

- **Non Sequential Collections**
- Set
- Set Operations

Set Data Type in Python

- A set is a non-sequential collection which is unordered and unindexed.
- A set is a mutable data type with nonduplicate, unordered values. providing the usual set operations. In Python sets are written with curly brackets.

```
#Example: Create a Set

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

```
Output:
{'cherry', 'apple', 'banana'}
```

- ***Note:*** *Since the set list is unordered, it means the items will appear in a random order. It cannot be sure in which order the items will appear.*
- To determine how many items a set has, use the len() method:

```
#Example: Get the number of items in a set.

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

```
Output:

3
```

The set() Constructor

- It is also possible to use the set() constructor to make a set.

- **Example:** Using the set() constructor to make a set:

```
thisset = set(("apple", "banana", "cherry"))

# note the double round-brackets

print(thisset)
```

```
Output:

{'banana', 'apple', 'cherry'}
```

Note: *The set list is unordered, so the result will display the items in a random order.*

Defining Empty set and Initializing Set

- To define an initially empty set, or to initialize a set to the values of a particular sequence, the set constructor is used.
- **Example:**

```
>>> set1 = set()
>>> len(set1)
0
```

```
>>> vegs = ['peas', 'corn']
>>> set(vegs)
{'corn', 'peas'}
```

```
>>>vowels = 'aeiou'
>>> set(vowels)
{'a', 'i', 'e', 'u', 'o'}
```

Note: *Note that set(), and not empty braces are not used to create an empty set, since that notation is used to create an empty dictionary.*

Accessing Items in Sets

- You cannot access items in a set by referring to an index, since sets are unordered the items has no index.
- You can loop through the set items using a for loop, **or** ask if a specified value is present in a set, by using the in keyword.

#Example: Loop through the set, and print the values.

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Output:

```
cherry
apple
banana
```

#Example: Check if "banana" is present in the set.

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Output:

```
True
```

Change Items in Sets

- Once a set is created, you ***cannot change*** its items, but you **can add new items**.

- **Add Items:** To add one item to a set use the add() method.

#Example: Add an item to a set, using the `add()` method

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

Output:

```
{'banana', 'apple', 'cherry', 'orange'}
```

- To add more than one item to a set use the update() method.

#Example: Add multiple items to a set, using the update() method:

```
thisset = {"apple", "banana", "cherry"}
thisset.update(["orange", "mango", "grapes"])
print(thisset)
```

```
Output: {'apple', 'banana', 'grapes', 'cherry', 'mango', 'orange'}
```

Set operations

- Python supports usual mathematical set operations.

Set operator	Set A = {1,2,3}	Set B = {3,4,5,6}	
membership	`1 in A`	True	*True if 1 is a member of set*
add	`A.add(4)`	{1,2,3,4}	*Adds new member to set*
remove	`A.remove(2)`	{1,3}	*Removes member from set*
union	`A | B`	{1,2,3,4,5,6}	*Set of elements in either set A or set B*
intersection	`A & B`	{3}	*Set of elements in both set A and set B*
difference	`A - B`	{1,2}	*Set of elements in set A, but not set B*
symmetric difference	`A ^ B`	{1,2,4,5,6}	*Set of elements in set A or set B, but not both*
size	`len(A)`	3	*Number of elements in set (general sequence operation)*

Remove Item From a Set

- To remove an item in a set, use the remove(), or the discard() method.

#Example: Remove "banana" by using the `remove()` method:

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

Output:

```
{'apple', 'cherry'}
```

- ***Note:*** *If the item to remove does not exist, remove() will raise an error.*

#Example: Remove "banana" by using the `discard()` method:

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

Output:

```
{'cherry', 'apple'}
```

Note: *If the item to remove does not exist, discard() will* ***NOT*** *raise an error.*

Remove Item From a Set

- You can also use the pop(), method to remove an item, but this method will remove the *last* item.

- The return value of the pop() method is the removed item.

#Example: Remove the last item by using the `pop()` method:

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()

print(x) #removed item
print(thisset) #the set after removal
```

Output:

```
banana
{'cherry', 'apple'}
```

Note: *Sets are unordered, so when using the pop() method, you will not know which item that gets removed.*

Empties and Delete the Set

- The clear() method empties the set. The del keyword will delete the set completely.

#Example1: clear() method.

```
thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)
```

Output:

```
set()
```

#Example: del keyword.

```
thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset) #this will raise an error because
the set no longer exists
```

Output:

```
    print(thisset) #this will raise an error because the set no longer exists
NameError: name 'thisset' is not defined
```

Join Two Sets

- There are several ways to join two or more sets in Python.
  - union(): This method: It returns a new set containing all items from both sets.
  - update(): It inserts all the items from one set into another.

#Example1: Join using `union()` method.

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

Output:

```
{2, 'c', 3, 'b', 1, 'a'}
```

#Example: Join set2 into set1 using update().

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

Output:

```
{2, 1, 3, 'b', 'a', 'c'}
```

- ***Note:*** *Both union() and update() will exclude any duplicate items.*
- **Note:** There are other methods that joins two sets and keeps ONLY the duplicates, or NEVER the duplicates.

Set Methods

- Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)

Set Methods (Cont..)

Method	Description
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Exercise:

Check if "apple" is present in the `fruits` set.

```
fruits = {"apple", "banana", "cherry"}

if "apple"      fruits:

      print("Yes, apple is a fruit!")
```

Frozenset Type in Python

- There are two set types in Python:
  - Mutable set type
  - Immutable set type
- *A* **frozenset** *is an immutable set type.*
- Methods add and remove are not allowed on sets of frozenset type. Thus, all its members are declared when it is defined.

```
#Example: Defining frozenset

apple_colors = frozenset(['red', 'yellow', 'green'])
```

- The values of a set of type *frozenset* must be provided in a single list when defined.
- A *frozenset* type is needed when a set is used as a key value in a given *dictionary*.

Exercise:

LET'S TRY IT

From the Python shell, enter the following and observe the results.

```
>>> s = {1,2,3}
>>> 1 in s
???

>>> s.add(4)
>>> s
???

>>> s = set('abcde')
>>> s
???
```

```
>>> s = set(['apple', 'banana', 'pear'])
>>> s
???

>>> s.add('pineapple')
???

>>> s = frozenset(['apple', 'banana', 'pear'])
>>> s.add('pineapple')
>>> ???
```

MCQs

1. Indicate all of the following that are syntactically correct for creating a set.
   a) set([1, 2, 3])
   b) set((1, 2, 3))
   c) {1, 2, 3}

2. For *set s* containing values 1, 2, 3, and *set t* containing 3, 4, 5, which of the following are the correct results for each given set operation?
   a) s | t → {3}
   b) s & t → {1, 2, 3, 4, 5}
   c) s - t → {1, 2}
   d) s ^ t → {1, 2, 4, 5}

3. For *set s* containing values 1, 2, 3 and *set w* of type *frozenset* containing values 'a', 'b', 'c', which of the following are valid set operations?
   a) 'a' *in* s
   b) 'a' in w
   c) len(s) + len(w)
   d) *s.add*(4)
   e) *w.add*('d')
   f) s | w
   g) s & w
   h) s - w

MCQs: Answers

1. Indicate all of the following that are syntactically correct for creating a set.

   a) set([1, 2, 3])

   b) set((1, 2, 3))

   c) {1, 2, 3}

2. For *set s* containing values 1, 2, 3, and *set t* containing 3, 4, 5, which of the following are the correct results for each given set operation?

   a) s | t → {3}

   b) s & t → {1, 2, 3, 4, 5}

   c) s - t → {1, 2}

   d) s ^ t → {1, 2, 4, 5}

3. For *set s* containing values 1, 2, 3 and *set w* of type *frozenset* containing values 'a', 'b', 'c', which of the following are valid set operations?

   a) 'a' *in* s

   b) 'a' in w

   c) len(s) + len(w)

   ***d) s.add*(4)**

   e) w.add('d')

   f) s | w

   g) s & w

   h) s - w

Thank You

?