ISTA 350

...

Tuples and Sets
Spring 2020

Reminders + Previously in ISTA 350 (lab)

Reminders

- Hw2 is due this Thursday, 2/13

Last week

- Linked Lists and While Loops

Tuples

Tuples

A tuple is a Python **sequence** type (like a list or a string, etc.).

They are used to group elements of any type into one coherent unit.

Tuples can essentially be thought of as **immutable** lists, meaning you cannot change the elements of a tuple once it has been created.

Note: If a tuple contains a list as one of its elements then you can mutate the list within the tuple, but you are still not mutating the tuple itself.

Creating Tuples

Tuple literals are created similar to a list, but with the elements enclosed by parentheses.

```
>>> tup = (1, 2, 'a', 'D', 'w', [0, 1, ':)'], True)
>>> one_el_tup = (1,) #NOTE: A one element tuple, notice the comma.
>>> type(one_el_tup)
<class 'tuple'>
>>> not_one_el_tuple = (2) #NOTE: Not a tuple.
>>> type(not_one_el_tuple)
<class 'int'>
```

Creating Tuples

You can convert any iterable type (list, string, dictionary, np.array) to a tuple using the `tuple()` function.

```
>>> tuple('Python')
('P', 'y', 't', 'h', 'o', 'n')
>>> tuple({1: 'a', 2: 'b', 3: 'c'})
(1, 2, 3) # tuple() on a dictionary returns a tuple of the keys.
```

*Tuple Hack: there is a way to change the elements of a tuple. Use the list() function to convert the tuple into a list and then tuple() to convert it back to a tuple!

Creating Tuples

Functions and methods can be defined to return a tuple.

```
>>> def tuplize_SL(section_leader, time):
>>>     return section_leader, time


>>> tuplize_SL('Hannah', '2:00pm')
('Hannah', '2:00pm')
```

Accessing Tuple Values

Tuples use the same sequence operations as strings or lists to access its values.

Use square brackets, [], to index elements in a tuple (0 based indexed).

Negative based indexing is supported.

Accessing Tuple Values

```
>>> tup = (1, 2, 'a', 'D', 'w', [0, 1, ':)'], True)

>>> tup[1]
2
>>> tup[-2]
[0, 1, ':)']
>>> tup[2:4]
('a', 'D')
```

Accessing Tuple Values

```
>>> tup[0] = 101 # TypeError!

>>> tup.append('Hi') # AttributeError!

>>>

>>> tup[5].append(':P')

>>> tup[5][0] = 21

>>> tup

(1, 2, 'a', 'D', 'w', [21, 1, ':)', ':P'], True)
```

Tuple Operations

`tup2 = (3, 2, 1)`	
`len(tup2)`	`3`
`('sup',) * 4`	`('sup', 'sup', 'sup', 'sup')`
`tup2 + tup2`	`(3, 2, 1, 3, 2, 1)`
`max(tup2)`	`3`
`sum(tup2)`	`6`
`sorted(tup2)`	`[1, 2, 3]`
`for x in tup2:` `  print(x, end=' ')`	`3 2 1`

Sets

Sets

Sets are unordered collections of immutable Python objects.

Recall that unordered collections have no order, cannot be indexed with integers, and all elements in the collection are unique (no duplicates).

Creating Sets

Similar to making a dictionary, except that there are no mappings. You can think of a set as a dictionary that only contains keys.

```
>>> set1 = {1, 2, 3, True, 'Hi', None, 3.2}

>>> type(set1)

<class 'set'>

>>> set2 = set() # The empty set.
```

Operations on Sets

Operation Name	Symbol	Functionality
Union	|	Returns a new set with all elements in both sets.
Intersection	&	Returns a new set with only those elements common to both sets.
Difference	-	Returns a new set with all the items from the first set not in the second.
Subset	<=	Asks whether all elements of the first set are in the second (making it a subset).
Proper subset	<	Asks whether all elements of the first set are in the second, but first set is not equal to the second set.

Operations on Sets

```
>>> a_set = {1, 2, 3, 3, 3, 4, 4, 5, 6, 7, 8}
>>> a_set
{1, 2, 3, 4, 5, 6, 7, 8} # Recall that sets have no duplicates.
>>> b_set = {1, 2, 3, 4, 9, 10, 8, 12, 0}

>>> a_set | b_set
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12}
>>> a_set & b_set
{8, 2, 1, 3, 4}
>>> a_set - b_set
{5, 6, 7}
>>> a_set <= b_set
False
```

Set Methods

Method Name	Functionality
`add()`	Adds items to the set.
`remove()`	Removes item from the set.
`pop()`	Removes an arbitrary element from the set.
`clear()`	Removes all elements from the set.

Set Methods

```
>>> a_set = {1, 2, 3, 3, 3, 4, 4, 5, 6, 7, 8}
>>> a_set
{1, 2, 3, 4, 5, 6, 7, 8}
>>> a_set.add(10)
>>> a_set
{1, 2, 3, 4, 5, 6, 7, 8, 10}
>>> a_set.remove(4)
>>> a_set
{1, 2, 3, 5, 6, 7, 8, 10}
>>> a_set.pop()
1
>>> a_set
{2, 3, 5, 6, 7, 8, 10}
```

Exercises

Download and complete the functions in lab3.py