Python Overview

Basic Data Structures you might need for hw 1

- Lists
- Tuples
- Dictionaries
- Sets
- Stacks
- Queues
- Priority Queues
- Classes

Lists

Creating a list

```
l1 = ['ai', 'is', 'awesome']
l2 = [ 5, 4, 3, 2, 1]
l3 = []
```

Accessing values in a list

```
>>> print(l1[0] )
“ai”
>>> print(l2[-1]) #gets the last element
1
```

Updating list

```
>>> l2[2] = 7
>>> print(l2[2])
7
```

Lists

Slicing lists #creates copy different from original list

```
list[start:end:step]
>>>l2 = [ 5, 4, 3, 2, 1]
>>>l2[:]
[5, 4, 3, 2, 1]

>>>l2[1:4]
[4, 3, 2]

>>> l2[0:5:2]
[5, 3, 1]

>>> l2[::-1] #reverses list
[1, 2, 3, 4, 5]
```

Lists

Copying lists

copy.deepcopy is slow, we recommend copying as follows:

```
list1 = [5,6]
list2 = list(list1) or list2 = list1[:]
```

Checking for membership:

```
>>> l1 = ['ai', 'is', 'awesome']
>>> if "ai" in l1:
        print("present")
present
```

Lists

Reversing a list:

```
l1.reverse()
sorted(l1, reverse = True)
l1= l1[::-1]
```

Other list operations:

Functions:

```
len(list), max(list), min(list)
```

Methods:

```
list.append(object), list.pop(), list.remove(object), list.insert(index,
object), list.sort()...
```

Tricky list behaviors for Python beginners

1. Elements in a list can be of different types:

```
list1 = [‘I’, ‘ate’, 1, ‘apple’]
```

2. Elements in a list can be lists - that’s how we form 2D arrays!

```
list2 = [[1, 2, 3], [4, 5, 6]]
```

3. Slicing a list creates a NEW list:

If function rev reverses a list, and list3 = [1, 2, 3, 4, 5]

Calling rev(list[1:4]) will **not** result in list3 becoming [1, 4, 3, 2, 5] - list3 remains the same!

Tricky list behaviors for Python beginners

4. Passing a list as a parameter to a function passes its reference:

```
def foo(somelist):
    if len(somelist) >= 1:
        somelist[0] += 2
list4 = [1, 2, 3]
foo(list4)
print(list4) # [3, 2, 3]
```

Tuples

Immutable versions of lists

Creating a tuple:

```
tuple1 = (1, 2, 3)
```

Accessing a value in a tuple:

```
print(tuple1[0]) # 1
```

Updating tuple - You can’t do this!:

```
tuple1[0] = 5 # throws error!
```

Dictionaries

key-value pairs, key has to be immutable (strings, tuples, **not** lists)

Creating a dictionary:

```
dict1 = {'a': 5, 'b': 3, 'c': 4}

dict2 = {} # creates empty dictionary
```

Checking for membership (O(1)):

Use **in** operator, e.g. 'a' in dict1

Insert/Modify key-value pair:

```
dict1['a'] = 6
```

Dictionaries

Delete key-value pair:

Use **del** operator: del dict1[‘a’]

Iterating over a dictionary (order not guaranteed):

```
for k, v in dict1.items():
    print(k, v)
```

Similarly, use dict1.keys() and dict1.values() for keys and values

Sets

Creating a set:

```
set1 = {1, 1, 1, 2, 2, 3} # set1 will in fact be {1, 2, 3}

set2 = set() # cannot use set2 = {} to create empty set!
```

Checking for membership (O(1)):

Use **in** operator, e.g. 1 in set1

Insert value into set:

```
set1.add(4) # set1 will become {1, 2, 3, 4}
```

Sets

Delete value from set:

```
set1.remove(2) # set1 will become {1, 3, 4}
```

Manipulating two sets:

Union: set1 | set2

Intersection: set1 & set2

Difference: set1 - set2

Stacks

You can use lists in python as stacks:

```
>>> mystack = [1,2,3,4,5]
>>> mystack
[1, 2, 3, 4, 5]

>>> mystack.append(6)
>>> mystack
[1, 2, 3, 4, 5, 6]
>>> mystack.pop()
6
>>> mystack.pop()
5
>>> mystack
[1, 2, 3, 4]
```

Queues

```
>>> from collections import deque
creating a deque object
>>> queue = deque([1,2,3])
>>> queue
deque([1, 2, 3])

adding to it
>>> queue.append(5)
>>> queue
deque([1, 2, 3, 5])

>>> queue.popleft()#implements FIFO functionality
1
>>> queue
deque([2, 3, 5])
```

Priority Queues

You have a couple of options when it comes to using priority queues

1. heapq

```
>>> import heapq
>>> q2 = [] #create an empty list

adding to the priority queue
>>> heapq.heappush(q2, (2, 'intelligence'))
>>> heapq.heappush(q2, (1, 'artificial'))

#the code fragment below is just to show the order in which elements are
popped from the priority queue
>>> while q2:
    next_item = heapq.heappop(q2)
    print(next_item)
(1, 'artificial')
(2, 'intelligence')
```

Priority Queues

2. queue.PriorityQueue

```
from queue import PriorityQueue
>>> q = PriorityQueue() #creating the object

adding to the priority queue
>>> q.put((2, 'intelligence'))
>>> q.put((1, 'artificial'))

#the code fragment below is just to show the order in which elements are
popped from the priority queue
>>> while not q.empty():
    next_item = q.get()
    print(next_item)

(1, 'artificial')
(2, 'intelligence')
```

OTHER USEFUL FUNCTIONS

map(function, iterable)

Applies function to every element of the iterable

e.g. in the skeleton code: map(int, begin_state) converts every element in begin_state to int (returns **new** data structure, does not modify original)