CSCI-UA-0002

Intro to Computer Programming (No Prior Experience)

Module 10: Advanced Lists + Dictionaries

Professor Emily Zhao

Section 008
T/R 12:30-1:45PM

Section 012
T/R 4:55-6:10PM

Agenda

- Module 9 Review Quiz
- Your Questions on Ed
- Module 10 Review
- World Series: List vs Dictionary
- Programming Challenges

Poll Everywhere

https://pollev.com/emilyzhao

Your Questions

→ What is the difference between "{}", "[]" and "()"?

→ What is the difference between `.pop()`, `.popItem()`, and `del`?

→ What is the difference between `.sort()` and `sorted()`?

Advanced Lists

Programming Challenge

```
# function: shuffle_list
# input: a list
# processing: create a shuffled copy of the list
# output: return shuffled list

a = [1, 2, 3, 4, 5]
shuffled_a = shuffle_list(a)
print(shuffled_a)

# >> [2, 5, 4, 1, 3]
```

Shuffling / Randomizing a List

You can shuffle (or randomize) the elements in a list by applying the following algorithm:

- Create an empty list
- Enter into a while loop
- Obtain a random number between 0 and the length of the list you wish to shuffle
- Place a copy of the data that exists at this random position into our new list
- Remove the item from original list
- Test the length of the original list – if it is zero we can end the while loop
- Your new list now contains a shuffled version of your original list

```
import random

# function: shuffle_list
# input: a list
# processing: create a shuffled copy of the list
# output: return the shuffled list

def shuffle_list(oldList):

    newList = []

    # while there are still items in the old list
    while len(oldList) > 0:

        #pick a new position and add it to the new list
        pos = random.randint(0, len(oldList)-1)
        newList.append( oldList[pos] )

        # remove element by position from old list
        del oldList[pos]

    return newList
```

Multi-dimensional lists

- All of the lists we have been creating so far have been one dimensional (i.e. linear) in nature
- You can create a two dimensional list in Python by simply nesting a list inside of another list.

```
mylist = [ [ 'a', 'b', 'c' ],
           [ 'd', 'e', 'f'] ]

print (mylist[0])
print (mylist[0][0])
print (mylist[1][1])

>> ['a', 'b', 'c']
>> a
>> e
```

Given the following list:

```
myList = [True, [0, 1, 2], "a", ["x","y","z"], "b",
          [3, 4, ["cat", "dog"]], "c"]
```

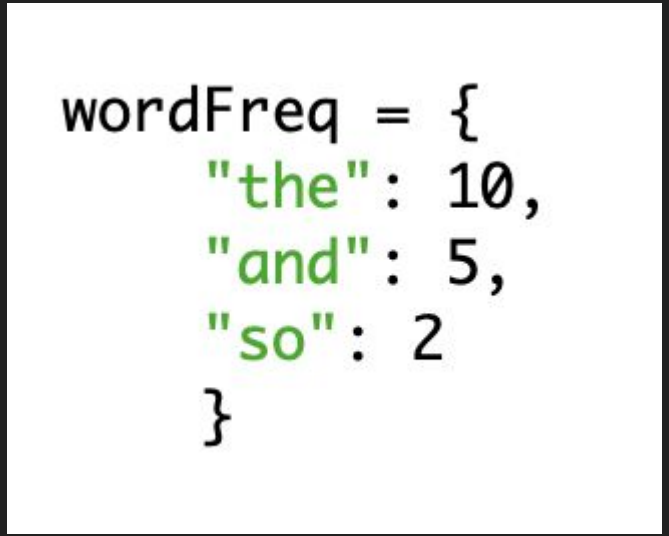
Write the line of code that will print the following:

- `[0,1,2]` `myList[1]`
- `"b"` `myList[4]`
- `"x"` `myList[3][0]`
- `"dog"` `myList[5][2][1]`

Dictionaries

Dictionaries

- A dictionary is a data structure for storing pairs of values
- Unlike a list, a Dictionary doesn't use integer based index values.
- Instead, Dictionaries use immutable objects (like Strings) to index their content. These are also call **keys**. Keys are unique and cannot be repeated within a dictionary.
- Values can be accessed by their keys. Like lists, dictionaries are mutable.



Creating an empty dictionary and adding key-value pairs

```
# create empty dictionary
versions = {}

# add python to dictionary
versions["python"] = 3.2

print(versions)
# >>> {"python": 3.2}

print(versions["python"])
# >>> 3.2

print(versions["java"])
# >>> KeyError: 'java'
```

Creating a dictionary with values

- You cannot access elements in a Dictionary that have not been defined.
- You can use the “in” operator to test to see if a key is in a dictionary like this (note: this will check for the presence of a key in a dictionary, not for the data that the key is storing!)

```
# create dictionary with values
versions = {
    "python": 3.2,
    "java": 1.8
    }

del versions["sql"]
# >>> KeyError: 'sql'

# check to see if java exists
# if so, remove it
if "java" in versions:
    del versions["java"]

print(versions)
# >>> {"python": 3.2}
```

Removing Items from a Dictionary

`.pop()` is used to remove and return the value associated with a specific key

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
value = my_dict.pop('b')
print(value)  # Output: 2
print(my_dict)  # Output: {'a': 1, 'c': 3}
```

`.popitem()` is used to remove and return an arbitrary (key, value) pair.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
item = my_dict.popitem()
print(item)  # Output: ('c', 3)
print(my_dict)  # Output: {'a': 1, 'b': 2}
```

Removing Items from a Dictionary

`del` is a general-purpose statement for deleting objects in Python.

For dictionaries, you can use del to remove an item by specifying its key or delete the entire dictionary.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
del my_dict['b']
print(my_dict)  # Output: {'a': 1, 'c': 3}
del my_dict # Deletes entire dictionary
```

Clearing a Dictionary

You can also clear all keys in a Dictionary by doing the following:

```
my_dictionary.clear()
```

Lists vs. Dictionaries

	Order	Access	When to use
List			
Dictionary			

Lists vs. Dictionaries

	Order	Access	When to use
List	ordered		
Dictionary	unordered		

Lists vs. Dictionaries

	Order	Access	When to use
List	ordered	numeric index	
Dictionary	unordered	immutable key (i.e. String)	

Lists vs. Dictionaries

	Order	Access	When to use
List	ordered	numeric index	when order matters
Dictionary	unordered	immutable key (i.e. String)	need to access values with a unique key

Trace the Output

```
myList = []
myDict = {}

myList[0] = "Emily"
print(myList)

myDict["Emily"] = 26
print(myDict)
```

Trace the Output

```
myList = []
myDict = {}

myList[0] = "Emily"
print(myList)

myDict["Emily"] = 26
print(myDict)
```

```
    myList[0] = "Emily"
IndexError: list assignment index out of range
```

```
{'Emily': 26}
```

myDictionary.keys() myDictionary.values() myDictionary.items()

```
versions = {
    "python": 3.2,
    "java": 1.8
    }

print(versions.keys())
print(versions.values())
print(versions.items())

>>> dict_keys(['python', 'java'])
>>> dict_values([3.2, 1.8])
>>> dict_items([('python', 3.2), ('java', 1.8)])
```

Sorting

In Python, dictionaries are inherently unordered collections of key-value pairs, meaning they don't have a fixed order.

If you want to sort the items of a dictionary based on their keys or values, you can create a sorted representation using the `sorted()` function.

```
my_dict = {'b': 2, 'a': 1, 'd': 4, 'c': 3}

# Sort by keys
sorted_items_by_keys = sorted(my_dict.items())
print(sorted_items_by_keys)

# Output:
# [('a', 1), ('b', 2), ('c', 3), ('d', 4)]
```

Sorting

`.sort()` is a method that can be called on a list to sort the elements in-place.

It modifies the original list and **does not return** a new list.

```
my_list = [3, 1, 4, 1, 5, 9, 2]
my_list.sort()
print(my_list)
# Output: [1, 1, 2, 3, 4, 5, 9]
```

`sorted()` is a built-in function that takes an iterable (e.g., a list, tuple, or string) and returns a new sorted list.

It does not modify the original iterable; instead, **it creates and returns a new sorted list**.

```
my_list = [3, 1, 4, 1, 5, 9, 2]
sorted_list = sorted(my_list)
print(sorted_list)
# Output: [1, 1, 2, 3, 4, 5, 9]
print(my_list)
# Output: [3, 1, 4, 1, 5, 9, 2]
# (original list is unchanged)
```

Lists vs. Dictionaries vs. Sets vs. Tuples

(https://www.educative.io/answers/list-vs-tuple-vs-set-vs-dictionary-in-python)

Curly Braces `{}`

- Used to define a dictionary.

Square Brackets `[]`

- Used to define lists.
- Used for indexing and accessing elements in both lists and dictionaries.
- Used for indexing and slicing in both lists and strings.

Round Brackets `()`

- Used to define tuples (aka an immutable list).
- Used for function calls.
- Used for grouping expressions.

```
versions = {
    "python": 3.2,
    "java": 1.8
    }

for i in versions:
    # what is i?
    print(i)
```

```
python
java
```

```
for i in versions.items():
    # what is i?
    print(i)
```

```
('python', 3.2)
('java', 1.8)
```

```
for i in versions.items():
    lang = i[0] # "python"
    vers = i[1] # 3.2
    print("Your version of", lang, "is", vers)
```

```
for lang, vers in versions.items():
    print("Your version of", lang, "is", vers)
```

```
Your version of python is 3.2
Your version of java is 1.8
```

Programming Challenges

How to import data from URL

```
import urllib.request

# define url
url = "http://www.example.com"

# initial request to URL
response = urllib.request.urlopen(url)

# read data from URL as string
data = response.read().decode("utf-8")
```

Programming Challenge

This data file contains all World Series winning teams up until a few years ago:

— http://002-text-files.glitch.me/world-series.txt

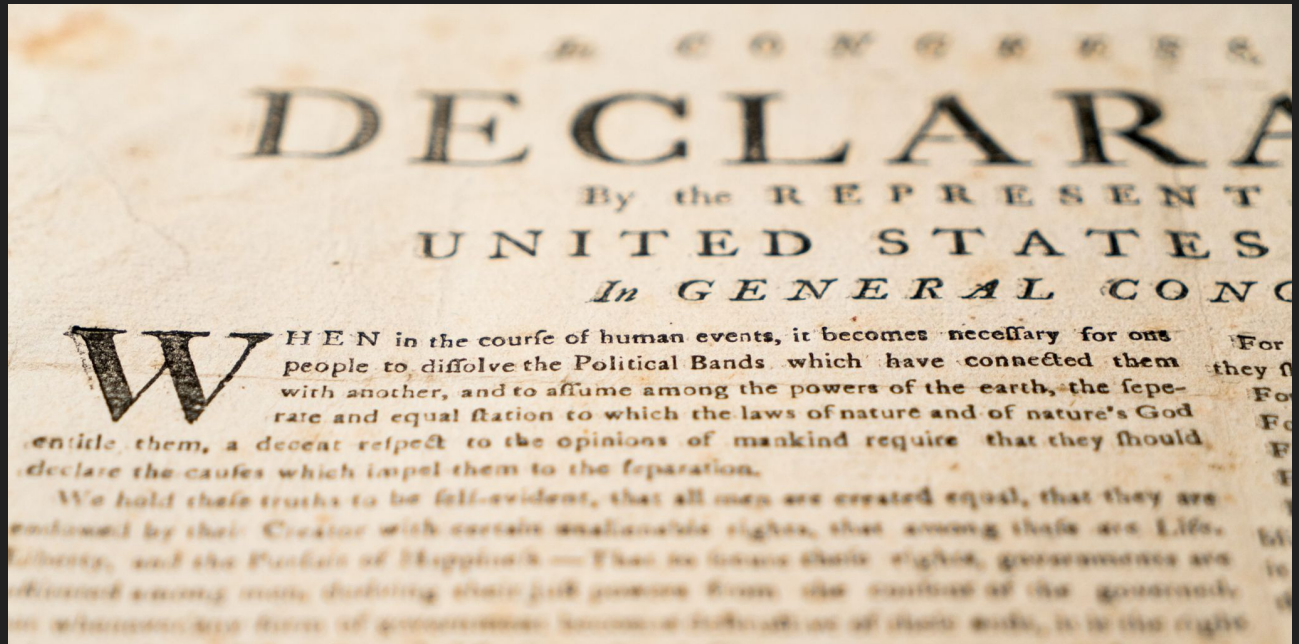
Write a program that reads in this data and finds the team that won the most games, now using a **dictionary**. Does it help?

> `world-series-list.py`
> `world-series-dict.py`

Answer code on class website

Programming Challenge

- Count the frequency of all words in the Declaration of Independence.
- Which word is said the most?



Programming Challenge

Write a program that creates a dictionary containing the U.S. states as keys, and their capitals as values. Use this states.txt file to build your dictionary.

The program should then randomly quiz the user by displaying the name of a state and asking the user to enter that state's capital.

The user has 3 lives – meaning if they get three or more wrong answers, the game is over.