Python
Dictionaries

Learning Objectives

- Introduce and discuss Dictionaries
- Identify the similarities and differences between Dictionaries and lists

Lists and Sequences

- **Recall:** lists are a sequence of values connected by a common name

```
grades = [76, 65, 98]
```

- Lists have many methods to manipulate their contents

```
grades.append(83)
grades.sort()
print(grades)
```

```
[65, 76, 83, 98]
```

- **Methods** are functions attached to some object and are accessed via the . operator

Dictionaries

- A dictionary is like a more general form of a list
  - Dictionaries are comprised of a collection of indices (keys) and a collection of related values
  - Dictionaries map a value to a key
    - Referred to as a key-value pair
  - We've seen this with lists
    - Lists map an index (an integer) to a value
  - Note that unlike strings and lists, dictionaries are not sequences. There is no implied ordering to the key-value pairs.

Declaring a Dictionary

- Declaring a dictionary can be accomplished by invoking the dict() function
- Notice the difference in the brackets
  - Square brackets [] indicate a list
  - Curly brackets {} indicate a dictionary

```
my_list = list()
my_dictionary = dict()
print(my_list)
print(my_dictionary)
```

```
[]
{}
```

Declaring a Dictionary

- Much like a list can be declared by initializing a variable to an empty list, a dictionary can be declared by initializing a variable to an empty dictionary

```
my_list = []
my_dictionary = {}
print(my_list)
print(my_dictionary)
```

```
[]
{}
```

Assigning Members

- To assign a member, use square brackets with an immutable value, such as a string, as the key
- To access a member, use the same mechanism
- If one attempts to access a key which does not exist, an exception occurs indicating a KeyError

```
my_dictionary = dict()
my_dictionary['key'] = "value"
print(my_dictionary['key'])
print(my_dictionary['no_key'])
```

```
value

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-2-f40fb13e6ae6> in <module>()
      2 my_dictionary['key'] = "value"
      3 print(my_dictionary['key'])
----> 4 print(my_dictionary['no_key'])

KeyError: 'no_key'
```

Short Form Assignment

- Attempting to print a dictionary displays a set of key-value pairs within curly braces
- That same format can be used to populate a dictionary
- Note: though these examples seem to indicate the order of assignment is retained, dictionaries are not a sequence

```
my_dictionary = dict()
my_dictionary['one'] = 'gary'
my_dictionary['two'] = 'dean'
my_dictionary['three'] = 'larry'
my_dictionary['four'] = 'sean'
my_dictionary['five'] = 'mitch'
print(my_dictionary)
```

```
{'one': 'gary', 'two': 'dean', 'three': 'larry', 'four': 'sean', 'five': 'mitch'}
```

```
my_dictionary = dict()
my_dictionary = {'uno': 'gary', 'dos': 'dean', 'tres': 'larry', 'cuatro': 'sean', 'cinco': 'mitch'}
print(my_dictionary)
```

```
{'uno': 'gary', 'dos': 'dean', 'tres': 'larry', 'cuatro': 'sean', 'cinco': 'mitch'}
```

in operator and len

- in checks to see if a key exists in the dictionary, not a value
- len returns the number of key-value pairs

```
my_dictionary = dict()
my_dictionary['one'] = 'gary'
my_dictionary['two'] = 'dean'
my_dictionary['three'] = 'larry'
my_dictionary['four'] = 'sean'
my_dictionary['five'] = 'mitch'
print('one' in my_dictionary)
print('gary' in my_dictionary)
print(len(my_dictionary))
```

```
True
False
5
```

Obtaining a list of keys

- When a dictionary is a parameter to the list function, the keys are returned in a list
- The sorted function returns a list of sorted keys

```
my_dictionary = dict()
my_dictionary = {'uno': 'gary', 'dos': 'dean', 'tres': 'larry'}
print(list(my_dictionary))
for elem in sorted(my_dictionary):
    print(elem)
```

```
['uno', 'dos', 'tres']
dos
tres
uno
```

for loops

- for loops over the keys of the dictionary
- Those keys can be used to access the values
- There is no automatic way to find a key given a value, though you could use a for loop and a comparison

```
for item in my_dictionary:
    print(item)
```

```
one
two
three
four
five
```

```
for item in my_dictionary:
    print(my_dictionary[item])
```

```
gary
dean
larry
sean
mitch
```

lists as values

- lists can be values of a dictionary
- Note the order of the keys/indices

```
students = dict()
grades = [83, 94, 34, 70]
students['greg'] = grades
students['bobby'] = [99,87,82,90]
print(students['greg'][2])
print(students)
```

```
34
{'greg': [83, 94, 34, 70], 'bobby': [99, 87, 82, 90]}
```

Lists of dictionaries of lists of…

```
student1 = dict()
student1['fname'] = 'greg'
student1['lname'] = 'nguyen'
student1['ID'] = '003003003'
student1['grades'] = [99,87,82,90]
student2 = dict()
student2['fname'] = 'bryan'
student2['lname'] = 'burlingame'
student2['ID'] = '001478315'
student2['grades'] = [49,56,75,90]
students = [student1, student2]
print(students[0]['fname'])
print(students[0]['grades'][2])
print(students)
```

- Dictionaries can be values stored in lists which can be values stored in dictionaries…
- This allows for storage of complex, related data
- Sometimes these collections of data are called records
- Note: I tend to use the plural for structures which refer to many things and the singular for structures which refer to one thing. This is a common style choice.

```
greg
82
[{'fname': 'greg', 'lname': 'nguyen', 'ID': '003003003', 'grades': [99, 87, 82, 90]}, {'fname': 'bryan', 'lname': 'burlingame',
'ID': '001478315', 'grades': [49, 56, 75, 90]}]
```

Looping through complex records

- Nested for loops can iterate over these complex records
- Given the previous students, let's print their average grades

```
for student in students:
    total = 0
    for grade in student['grades']:
        total += grade
    print(student['lname'] + ',' + student['fname'] + ":" + str(total/len(student['grades'])))
```

```
nguyen,greg:89.5
burlingame,bryan:67.5
```

Resources

- Bryan Burlingame's notes
- Downey, A. (2016) *Think Python, Second Edition* Sebastopol, CA: O'Reilly Media
- (n.d.). 3.7.0 Documentation. *String Methods — Python 3.7.0 documentation*. Retrieved October 16, 2018, from https://docs.python.org/3/library/stdtypes.html#string-methods