Recitation 14 - Python (Dictionary and Classes)

CSCI 1300

Instructor: Dr. David Knox

Dictionaries in Python:

A dictionary is an associative array. Dictionaries in python have *key* and *value* pairs; each *key* is associated to a *value*. Each key in dictionary must be unique and the values can be of any Python data type. Due to this reason, dictionaries in python are **unordered** key-value pairs.

In a dictionary, a colon (:) is used to denote each key-value pair, while the key-value pairs themselves are separated by commas. The whole thing is enclosed in curly braces.

Example:

```
mydict = {'Name': 'John', 'Job': 'Teacher', 'Age': 31, 'SSN': '2345678'}

print mydict['Name']          #Output is John
print mydict['SSN']           #Output is 2345678
```

Tip: it is also possible to print the entire dictionary, which may be helpful for debugging.

Updating values in a dictionary:

To change an element in a dictionary, simply access the value with the specific key

```
mydict['Job'] = 'Principal'   #The value associated with 'Job' is string type
mydict['Age'] += 1            #The value associated with 'Age' is int type

print mydict['Age']           #Output is 32
```

To add a new key-value pair to a dictionary use the below command

```
mydict['Gender'] = 'Male'     #adds the new key 'Gender' with value 'Male'
```

To iterate through all the elements in a dictionary, use the keyword **in**:

```
for Key in mydict:
  print Key, ':', mydict[Key]
```

Deleting specific elements from a dictionary:

To delete a element in a dictionary with a specific key we use the del keyword

```
del mydict['Name']            # remove the element with key 'Name'
```

To delete all the elements in a dictionary we use the clear command

```
mydict.clear()                  # delete all the entries in dict
```

To delete the entire dictionary, we use the below command

```
del mydict                      # delete the entire dictionary
```

Accessing elements in a dictionary (more):

To check to see if a key is in a dictionary, use the keyword **in**:

```
Key = 'Height'
if Key in mydict:
  print Key, ':', mydict[Key]
else:
  print 'key:', Key, 'does not exist'
```

The empty dictionary:

It is possible to create an empty dictionary in Python, just as you may create an empty list:

```
Mydict2 = {}                    # an empty dictionary
Mylist = []                     # an empty list
```

Errors and error handling with Dictionaries:

If we try to access a key which is not in our dictionary we will get an error as shown:

```
print mydict['Height']
```

```
mydict['Height']:
Traceback (most recent call last):
   File "example.py", line 2, in <module>
      print mydict['Height'];
KeyError: 'Height'
```

Use the **try** / **except** keywords to attempt to safely access a key you are unsure exists:

```
Key = 'Height'
try:
  print mydict[Key]
except:
  print 'key:', Key, 'does not exist'
```

Classes in Python:

As we learned previously within C++, a class is a user-defined prototype for an object that defines a set of attributes that characterize any object within the class. These attributes are called data members and methods, known as class and instance variables. Basically, a class is a way to take a grouping of functions or data and place them inside a container so you can access its attributes with the (dot) operator.

Terminology:

The terminology is basically the same for python as we saw in C++. You will want to refer back to Recitation 10 for detailed descriptions of Attributes, Methods, Classes, Objects, getters and setters.

Creating A Class:

Example of creating a Simple class:

```
class Student:
   numStudents = 0            # class variable, shared by all instances


   def __init__(self, name, gpa):               # initialization method
      self.name = name        # instance variable, unique to each instance
      self.gpa = gpa          # instance variable, unique to each instance
      Student.numStudents += 1

   def displayNumStudents(self):
     print "There are %d students" % Student.numStudents

   def displayStudentInfo(self):
      print "Name: ", self.name, ", GPA: ", self.gpa
```

Comments on Python classes:

- Here the class is created with the keyword class: **class classname**
- numStudents is known as a **class variable**. This variable's value is shared among all instances of the class and hence can be accessed by any instance of the class. The syntax for accessing the class variable is **classname.classvariable**.
  In this example, the class variable is accessed as **Student.numStudents**
- The **methods** of the Student class are: __init__ , displayNumStudents(), and displayStudentInfo().
- __init__ is an **initialization method,** similar to a constructor in C++. This method is automatically called every time you create a new instance(object) of the Student class.
- The other methods are like functions similar to the methods in C++. Though there is a difference. In Python, when you define the methods, the first parameter is always **self.** However, when you call these methods, you do *not* include the self parameter. The self parameter can also be called anything you prefer, however self is generally used by convention.

Why do I need self when I define the __init__ or other class methods?

If you don't have `self`, then code like `cheese = 'Frank'` is ambiguous. That code isn't clear about whether you mean the *instance's* `cheese` attribute, *or* a local variable named `cheese`. With `self.cheese = 'Frank'` it's very clear that you mean the instance attribute `self.cheese`.[1]

Creating Instances:

To create an instance of a class, otherwise known as an object, we call the class initialization method using the class name and pass all the parameters that the __init__ method accepts, *except the **self** attribute!*.

```
student1 = Student("John", 3.5)
student2 = Student("Susan", 3.86)
```

Here you will notice that we only give the name and gpa as arguments. The self parameter is not included.

[1] https://learnpythonthehardway.org/book/ex40.html

Accessing the attributes:

To access the attributes we use the dot operator with the object name. The class variables will be accessed using the class name.

```
student1.displayStudentInfo()
student2.displayStudentInfo()
student1.displayNumStudents()
student2.displayNumStudents()
```

Output:

```
Name: John , GPA: 3.5
Name: Susan , GPA: 3.86
There are 2 students
There are 2 students
```

More examples and explanations for python classes:
https://docs.python.org/3/tutorial/classes.html

Recitation Activity: Coding Activity Due by Sunday 5PM