Lecture 6: Dictionaries and modules

This class will cover two independent objects – dictionaries, which are objects to hold associative arrays, and modules, which are the equivalent of libraries in other languages.

Dictionaries

Dictionaries are a mutable data type somewhat similar to lists. Lists, as you should remember, are pairs of indexes and elements. In other words, lists are accessed using an index such as data[5], which returns the fifth element in the list. In contrast, dictionaries are accessed by keys, which can take on any sort of value, as long as they are immutable. The key is able to return a value that is associated with that key.

It is no consequence that dictionaries are called dictionaries, as real life dictionaries are useful examples of how an associative array works. A dictionary contains a set of definitions for a word. In this case the key is the word, and the value is its definition. Let's say you wanted to store a definition in a computer program.

```
>>> newdictionary = {"aardvark":"animal that eats bugs", "butterfly": "pretty
insect", "cat":"crazy pet"}
>>> newdictionary
{'aardvark': 'animal that eats bugs', 'butterfly': 'pretty insect', 'cat': 'crazy
pet'}
```

We have implicitly created a dictionary object using the {} (similar to how [] creates a list and "" creates a string). We can now access the values by their keys using the [] operator:

```
>>> newdictionary["cat"]
'crazy pet'
```

So instead of using a for loop to find the word we want, we can just use one line! This is the power of the dictionary, it allows you to rapidly map one object with another with minimal amounts of code. You can also create and add to a dictionary like so:

```
>>> newdictionary = {}
>>> newdictionary
{}
>>> newdictionary["aardvark"] = "animal that eats bugs"
>>> newdictionary["butterfly"] = "pretty insect"
>>> newdictionary["cat"] = "crazy pet"
>>> newdictionary
{'aardvark': 'animal that eats bugs', 'butterfly': 'pretty insect', 'cat': 'crazy
pet'}
```

What happens if you try to access a key that doesn't exist yet in the dictionary?

```
>>> newdictionary["new word"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'new word'
```

So this could be problematic – if you try to access a key that doesn't exist then you get an error. Luckily there's an easy way to test this using the previously described in statement:

```
>>> 'cat' in newdictionary
True
>>> 'new word' in newdictionary
False
```

As we can see below, newdictionary contains an __iter__ magic method which we can use to iterate through the dictionary.

```
>>> dir(newdictionary)
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__',
'__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy',
'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update',
'values']
>>> iterator = newdictionary.__iter__()
>>> next(iterator)
'aardvark'
>>> next(iterator)
'butterfly'
>>> next(iterator)
'cat'
>>> next(iterator)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Note that the iterator returns the key value. If you want to access the value in a loop, you can either use the key:

```
>>> for key in newdictionary:
...     print(key)
...     print(newdictionary[key])
...
aardvark
animal that eats bugs
butterfly
pretty insect
cat
crazy pet
```

You can also use the built-in methods keys(), values(), and items() to loop through the keys, values, or both:

```
>>> for key in newdictionary.keys():
...     print(key)
...
aardvark
butterfly
cat
>>> for value in newdictionary.values():
...     print(value)
...
animal that eats bugs
pretty insect
crazy pet
>>> for key,value in newdictionary.items():
...     print(key + " - " + value)
...
aardvark - animal that eats bugs
butterfly - pretty insect
cat - crazy pet
```

Modules

The second topic we will cover today are modules. If you quit the Python interpreter and enter it again, all of the functions you have created will have been lost. A module allows you to logically organize your Python code and both reuse it and share it. Simply, a module is a file consisting of Python code that can define functions, classes, and variables. These modules can then be imported into the interpreter or into your script for you to use. Some modules come with Python, some modules are created by you, and some are created by others and downloaded by you.

Built-in modules

Python likes to make the claim that it comes with batteries-included. What do they mean by this? Python comes equipped with a number of useful modules that makes your life easier. For example, so far we have seen how to do some simple math functions like adding and subtracting, but you also might find yourself needing to do a bit more complicated manipulation like exponentials or trigonometry. Python includes the math module for this.

The first think you need to do is import this module into Python using the import statement:

```
>>> import math
>>> math
<module 'math' from '/Users/pythonclass/opt/miniconda3/lib/python3.9/lib-
dynload/math.cpython-39-darwin.so'>
>>> type(math)
<class 'module'>
```

As you can see, we imported the math module (the location of the file was printed out) to create an object called math. The math object belongs to the module class. We can also change the name of the object that is created using the as keyword:

```
>>> import math as m
>>> m
<module 'math' from <module 'math' from
'/Users/pythonclass/opt/miniconda3/lib/python3.9/lib-dynload/math.cpython-39-
darwin.so'>
```

Now that we have imported this object, we can access its methods/attributes similarly to how we would for strings, lists, and dictionaries. As always, we can see what's available using the dir() function:

```
>>> dir(m)
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign',
'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isfinite',
'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf',
'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Ok, we can see there are a few magic methods, along with a large number of other methods and attributes. For example, pi is a useful attribute for calculating the area of a circle:

```
>>> m.pi
3.141592653589793
>>> radius = 6
>>> area = m.pi*radius*radius
>>> area
113.09733552923255
```

We can also use methods within each module:

```
>>> m.sin(0)
0.0
>>> m.sin(m.pi/6)
0.49999999999999994
>>> m.factorial(6)
720
```

Information on the built-in modules can be found at docs.python.org. For example, for math: https://docs.python.org/2/library/math.html.

You can also import individual methods from a module. For example:

```
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> m.cos(0)
1.0
>>> from math import cos
>>> cos(0)
1.0
>>> <class 'builtin_function_or_method'>
```

When you import an individual method, it does not attach it to a module object. Rather it creates a new function object. I.e. use cos() instead of m.cos(). Finally you can import all of the methods in a module using the command: from math import *

User-defined modules

You can easily create your own module. A module is essentially a python script containing functions and variables. Create a file called my_module.py in your home directory. Within this file create a function and a variable:

```
def tokenizer(a,b):
    out_list = a.split(b)
    return out_list
number = 2.423423432
```

start up the interpreter and we can now import it using the following:

```
>>> import my_module
>>> dir(my_module)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__', 'number', 'tokenizer']
>>> my_module.number
2.423423432
>>> my_module.tokenizer("A bunch of words", " ")
['A', 'bunch', 'of', 'words']
```

Python automatically looks in the directory where you started it up for anything called [name of module].py. So by typing “import my_module”, we are instructing the interpreter to look for a file called “my_module.py” (which it was able to find. All the functions and variables we defined in that file are loaded as an object attached to the my_module variable.

If you want to see all of the directories that python looks for modules, you can use the sys module:

```
>>> import sys
>>> sys.path
['',                          '/Users/pythonclass/opt/miniconda3/lib/python39.zip',
'/Users/pythonclass/opt/miniconda3/lib/python3.9',
'/Users/pythonclass/opt/miniconda3/lib/python3.9/lib-dynload',
'/Users/pythonclass/opt/miniconda3/lib/python3.9/site-packages']
```

User-built modules are a great way to reuse useful code. If you want to create a Directory that contains all of your code, you can modify the path python looks in by modifying the PYTHONPATH environmental variable in your shell configuration file. Add the following:

```
export PYTHONPATH=~/PythonClass/python_modules:$PYTHONPATH
```

Open-source modules

One of the most useful features of Python is the availability of modules from the Python community. Highly-skilled Python coders will often write modules that will do something really useful. However, since they aren't built-in, you have to somehow get them onto your computer. Typically, you will use an installer, which takes care of most of the nitty-gritty details. In this class, we will use conda for downloading and installing modules.

Let's say that you want to connect to the internet. Normally you would use a browser like Firefox or Chrome, however, you also might want to incorporate this in your Python script (let's say you wanted to automatically download data from the web once a week or run some sort of web tool on your data). Normally, this would seem extremely difficult. Luckily however, others have created Python code that deals with the complicated stuff, and allows you to access the internet at a much simpler level (not that it's easy, just much easier than it would be).

An example of this is the requests module. https://2.python-requests.org/en/master/

You can download this module using the command:

```
conda install -c anaconda requests
```

conda is the name of the command we are running. Install is an argument to this command that tells it to install a package, '-c' is theand then requests is the name of the package to install. Once installed you can use the new module just like a built-in:

```
>>> import requests
>>> r = requests.get('http://www.google.com')
>>> r.headers
{'content-encoding': 'gzip', 'x-xss-protection': '1; mode=block', 'cache-
control': 'private, max-age=0', 'set-cookie':
'PREF=ID=1111111111111111:FF=0:TM=1441293536:LM=1441293536:V=1:S=jCj5x1XTHKPkZLJT
; expires=Thu, 31-Dec-2015 16:02:17 GMT; path=/; domain=.google.com,
NID=71=fCu5RoZ_BtRBnXVxT7i05SP4OMv1XYxuEQcmqyZ_8-
zuP5m808Pywm4untHWGHkMHdtz20bFI8ow2mGrpCsYucFIJiZjlAiuI3FN_zAv-
pc2lhK4x3K5Mji8lN25wceI; expires=Fri, 04-Mar-2016 15:18:56 GMT; path=/;
domain=.google.com; HttpOnly', 'content-type': 'text/html; charset=ISO-8859-1',
'p3p': 'CP="This is not a P3P policy! See
http://www.google.com/support/accounts/bin/answer.py?hl=en&answer=151657 for more
info."', 'content-length': '7251', 'expires': '-1', 'server': 'gws', 'x-frame-
options': 'SAMEORIGIN', 'date': 'Thu, 03 Sep 2015 15:18:56 GMT'}
>>> r.encoding
'ISO-8859-1'
>>> r.cookies
<RequestsCookieJar[Cookie(version=0, name='NID',
value='71=fCu5RoZ_BtRBnXVxT7i05SP4OMv1XYxuEQcmqyZ_8-
zuP5m808Pywm4untHWGHkMHdtz20bFI8ow2mGrpCsYucFIJiZjlAiuI3FN_zAv-
pc2lhK4x3K5Mji8lN25wceI', port=None, port_specified=False, domain='.google.com',
domain_specified=True, domain_initial_dot=True, path='/', path_specified=True,
```

```
secure=False, expires=1457104736, discard=False, comment=None, comment_url=None,
rest={'HttpOnly': None}, rfc2109=False), Cookie(version=0, name='PREF',
value='ID=1111111111111111:FF=0:TM=1441293536:LM=1441293536:V=1:S=jCj5x1XTHKPkZLJ
T', port=None, port_specified=False, domain='.google.com', domain_specified=True,
domain_initial_dot=True, path='/', path_specified=True, secure=False,
expires=1451577737, discard=False, comment=None, comment_url=None, rest={},
rfc2109=False)]>
>>> r.url
'http://www.google.com/'
```