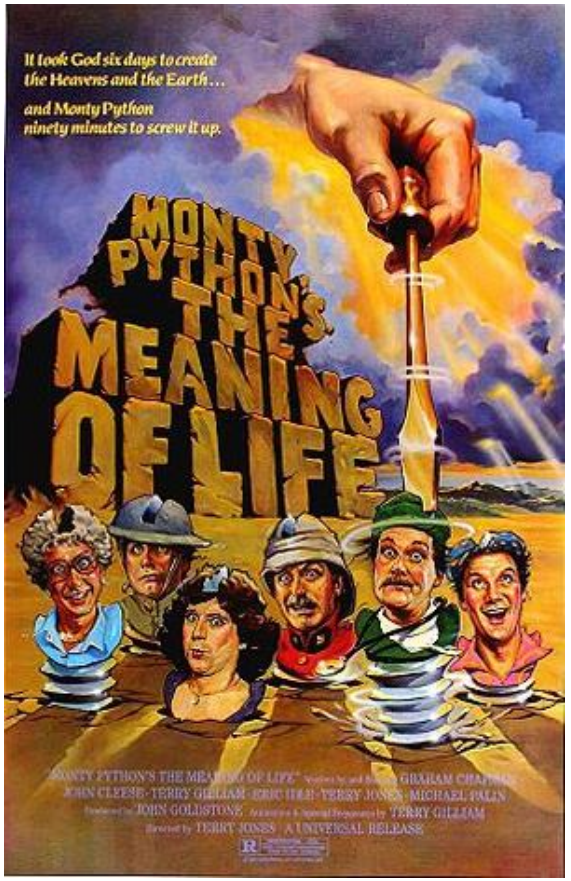
More Python Data Structures

- Classes
  - Should have learned in Simpson's OOP
  - If not, read chapters in Downey's *Think Python: Think like a Computer Scientist*
- Dictionaries
- Tuples next time



Dictionaries

- A dictionary is more general than a list
  - Indices do not have to be integer (can be almost any type)
- A dictionary is a mapping between keys and a set of values.
  - Key-value pair is sometimes called an item
- Examples:
  - Map English words to Spanish words (keys and values are strings)
  - Map words to frequency counts (keys are strings, values are integers)

Creating a dictionary

- The function `dict()` creates a dictionary with no items
  - This is the name of a built-in function, do NOT use it as variable name

```
eng2sp = dict()
wordfreq = dict()
print (eng2sp)  → {}
```

Adding to a dictionary

- To add items, user square brackets
  - Create an item that maps from the key to the value

```
eng2sp['one'] = 'uno'
wordfreq ['a'] = 0
print (eng2sp)  → {'one': 'uno'}

eng2sp['two'] = 'dos'
eng2sp['three'] = 'tres'
print (eng2sp)
    →  {'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

- Note: The order of the items is not the same as order added
  - You may get a different order on another machine
  - Similar to a hash

Adding items

- Can add more than one item at a time
  - Note format is the same as the output
  - Example: create a new dictionary with three items

```
eng2sp = {'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

Accessing items

- Access by the key

```
print(eng2sp ['two'])
```

- If the key isn't in the dictionary, get an exception

```
print(eng2sp ['four']) → KeyError: 'four'
```

dictionary functions and operators

- `len()` tells you how many items you have in the dictionary

  `print(len(eng2sp))` → `3`

- `in` operator indicates if a key is in the dictionary

  `'one' in eng2sp` → `True`

  `'uno' in eng2sp` → `False`

- `in` and method `values()` indicates if a value is in the dictionary

  `'uno' in eng2sp.values()` → `True`

get ( ) method

- Dictionaries have a method called get that takes a key and a default value.
  - If the key appears in the dictionary, get return the corresponding value; otherwise it returns the default value.

```
>>>d = frequency('a')
>>>print (d)
{'a': 1}
>>>d.get ('a', 0)
1
>>>d.get ('b', 0)
0
```

Looping and dictionaries

- Use a dictionary in a for statement, it traverses the keys of the dictionary

```
def print_items(d):
    for c in d:
        print (c, d[c])

>>>h = {'p': 1, 'a':1, 'r':2, 't':1, 'o':1}
>>>print_items (h)
a 1
p 1
r 2
t 1
o 1
```

`keys()` and `values()` method

- `keys()` is a method that returns a **`list`** of keys in the dictionary
- Print list of keys in a dictionary

```
for k in h.keys():
        print (k)
```

- `values()` is a method that returns a **`list`** of values in the dictionary
- Print list of values in a dictionary

```
for v in h.values():
        print (v)
```

`items()` method

- `items ()` is a method that returns a **`list`** of items in the dictionary

- Print list of items in a dictionary

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

- Example: counts letters in a string using a dictionary

- Note: Can't sort a dictionary, use an ordered dictionary from collections or use a tuple instead

Example

- Demo Dictionary example

Reverse lookup

- Given a dictionary and a key, easy to find correspond value

```
v = d[k]
```

- What if you want to find the key, given the value?
  - two problems - what are they?
    - might be two keys with the same value
    - no simple way to do it

```
def reverse_lookup (d,v):
    for k in d:
        if d[k] == v:
            return k
    raise ValueError
```

Running reverse_lookup

```
>>>h = {‘p’: 1, ‘a’:1, ‘r’:2, ‘t’:1}

>>>k = reverse_lookup (h, 2)
>>>print (k)
r

>>>k = reverse_lookup (h, 3)
Traceback(most recent call last):
  File”<stdin>”, line 1, in ?
  File “<stidin>”, line 5, in reverse_lookup ValueError
```

Lists and Dictionaries

- Lists can appear as values in a dictionary
- Example: have a dictionary that maps from frequencies to lists
  - {1 : ['p', 'a', 't', 'o'], 2 : ['r']}

```
def invert_dict(d):
    inverse = dict()
    for key in d:
        val = d[key]
        if val not in inverse:
            inverse[val] = [key]
        else:
            inverse[val].append(key)
    return inverse
```

Lists and dictionaries

- Lists can be values in a dictionary, but they cannot be keys
  - a dictionary is implemented as a hash table
  - keys have to be “hashable”, that is they have to be immutable

- If you want to get around this, use tuples

- ***Both*** lists and dictionaries can be used as values

Example

```
d = {'Broncos' : {'QB' : ['Manning', 'Osweiler'],
                  'WR' : ['Sanders', 'Thomas', 'Latimer']},
     'Cowboys' : {'QB' : ['Romo', 'Wheedon'],
                  'WR' : ['Bryant', 'Beasley', 'Williams']},
     'Colts' :   {'QB' : ['Luck', 'Hasselbeck'],
                  'WR' : ['Johnson', 'Moncrief', 'Hilton']} }


print (d)
```

DEMO

Note

- You wouldn’t hard code a hash of a hash of lists, would you?