Dictionaries

- A dictionary is an **unordered** collection of key, value pairs.
- Each key-value pair is separated by a colon : and each key is separated by a comma.
- Example - you build an app and you ask each user to register when they download it.
  - You know you want to collect user's name, email and age (and only that information).
  - A dictionary is a good way to store this information, and you can access the user data using the corresponding "key" or label that you assigned.
  - Because you interact with a dictionary via the "key", the order of the key,values pairs doesn't matter!

Recall that with a list you have just one value in each position

In [ ]:
```
names = ['john', 'ella']
names[1]
```

Out[ ]: 'ella'

Dictionary - store a set of key, value pairs

- use {} to denote a dictionary object
- index by key

In [ ]:
```
d = {'john':41, 'ella':12}

# index by key is ok
print(d['john'])

# but indexing by position won't work
# print(d[0])

# nor will indexing by value...
# print(d[41])
```

```
41
```

Note: Remember that because you interact with a dictionary via the "key", the order of the key,values pairs doesn't matter!

In [ ]:
```
# no functional impact of swapping the order of entry
d = {'ella':12, 'john':41}
# index by key...
print(d['john'])
```

```
41
```

Can also use the "dict" function to make a dictionary object

- simple example here (could also use the {} syntax for this...)

In [ ]:
```
d = dict(SD='Padres', SF='Giants')
print(d['SD'])
```

```
Padres
```

Use the "dict" constructor to convert list of tuples to dictionary

- Could also make a dictionary out of a list of lists (this is where the dict function comes in handy)

In [ ]:
```
list_of_tuples = [('SD', 'Padres'), ('SF', 'Giants')]
print(list_of_tuples[1])
d = dict(list_of_tuples)
print(d['SF'])
```

```
('SF', 'Giants')
Giants
```

Methods associated with dictionaries

- many, but a few handy ones: clear, get, pop

In [ ]:
```
d = dict(SD='Padres', SF='Giants')

print(d)
d.clear()
print(d)
```

```
{'SD': 'Padres', 'SF': 'Giants'}
{}
```

Pop method - returns the **value** associated with the requested key, and removes the key,value pair from the dictionary

In [ ]:
```
d = dict(SD='Padres', SF='Giants')

x = d.pop('SD')
print(x)
print(d)
```

```
Padres
{'SF': 'Giants'}
```

popitem() method will return the **key, value pair** as a tuple and then remove the key,value pair from the dictionary

- In Python v3.7 and later, popitem removes the last item entered
- In earlier versions it will remove a random item

In [ ]:
```
d = dict(SD='Padres', SF='Giants')

kv = d.popitem()
print(kv)
```

```
('SF', 'Giants')
```

Asking for a key that isn't in the list

- can use the get method to see if something is in a list...
- will return 'None' if key,value pair not present, but will not throw an error - can be handy sometimes if you're not sure what's in the list but you don't want program execution to halt if you mistakenly ask for the wrong thing...

In [ ]:
```
d = dict(SD='padres', SF='giants')

# this will throw an error
# d['LA']

# this will return None and keep going.
print(d.get('LA'))
```

```
None
```

Example of d.get()

- Looping through a set of potential key values, and not sure if they are actually in the dictionary

In [ ]:
```
keys_to_search = ['LA', 'SD', 'SF']
for i in range(0,3):
  print(d.get(keys_to_search[i]))
 # print(d[keys_to_search[i]])
```

```
None
padres
giants
```

'Update' will merge two dictionaries - note what happens to redundant entries!

In [ ]:
```
d1 = {'john':41, 'ella':11}
d2 = {'jack':9, 'vy':25, 'john':28}

# update or merge the two...
d1.update(d2)
print(d1)   # how old is john????
```

```
{'john': 28, 'ella': 11, 'jack': 9, 'vy': 25}
```

Separately get the keys and values and return as 'dict_keys' and 'dict_values' objects

- can convert these to indexible lists

In [ ]:
```
uc_enrollment = {'UCSD':35816, 'Irvine':27331, 'Merced':6815}

schools = uc_enrollment.keys()
print(schools)

# can't index into this
schools[0]

# but can convert to a list!
school_list = list(schools)
print(school_list[0])
```

```
dict_keys(['UCSD', 'Irvine', 'Merced'])
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-ada302bcfacc> in <module>()
      5
      6 # can't index into this
----> 7 schools[0]
      8
      9 # but can convert to a list!

TypeError: 'dict_keys' object does not support indexing
```

get just the values

In [ ]:
```
num_students = uc_enrollment.values()
print(num_students)
```

Can get the combined key,value pairs using the items() method

In [ ]:
```
# use items to loop over key/value pairs.
for k, v in uc_enrollment.items():
  print(k)
  print(v)
```

More advanced indexing (and storing function names in a list/dictionary)

In [ ]:
```
my_stuff = ['Minnie', [21, 19], sorted, sum, [100, 20, 20]]

# 2nd entry of the 5th element in my_stuff
my_stuff[4][1]

# sort the list 21,19
my_stuff[2](my_stuff[1])

# sum of 20+20
my_stuff[3](my_stuff[4][-2:])
```