Use all the new skills you've learned for iterating over data structures like dictionaries and lists to practice in this notebook.

Data structures

The trick is that it is all about state!

Lists

Lists are easy to encounter and easy to abuse. Lists hold individual items, keeping a specific order. To access them, treat the order like an index. The index starts at `0,` and it continues incrementally every time a new item gets added. A loop (sometimes referred to as *"for loop"*) is the most common operation you can encounter.

In [2]:
```
directories = ['Documents', 'Music', 'Desktop', 'Downloads', 'Pictures', 'Mov
for directory in directories:
  print(directory)
```

```
Documents
Music
Desktop
Downloads
Pictures
Movies
```

In [3]:
```
import os
for item in os.listdir('sample_data'):
  if os.path.isdir(item):
    print("This is a directory {0}".format(item))
  else:
    print("This is a file: {0}".format(item))
```

```
This is a file: wine-ratings-small.csv
This is a file: wine-ratings.csv
This is a file: wine-ratings.json
```

In [4]:
```
# Looping is easy, but what about state?
# here state is captured in a new variable called `important_directories`
important_directories = []
for item in os.listdir('.'):
  if os.path.isdir(item):
    important_directories.append(item)
print(important_directories)
```

```
['sample_data', '.ipynb_checkpoints', '.git']
```

In [11]:

```
os.listdir('.')
```

Out[11]:

```
['README.md',
 'sample_data',
 '.gitignore',
 'looping-data-structures.ipynb',
 '.ipynb_checkpoints',
 '.git']
```

In [6]:

```
important_directories = []
for item in os.listdir('.'):
  if item.startswith('.'):
    continue # flow control!
  if os.path.isdir(item):
    important_directories.append(item)
print(important_directories)
```

```
['sample_data']
```

In [4]:

```
items = ['first', 'second', 'third', 'foo']
items[-1]
url = "https://colab.research.com/drive/asdfjhasdf/alfredo/oreilly"

parts = url.split('/')
print(parts)
# Everything except the first three items
print(parts[3:])
protocol, _, fqdn = parts[:3]
print("protocol is: %s" % protocol)
print(fqdn)
company = parts[-1]
print(company)

print("The first item is: {0}".format(items[0]))

items[1]

# you can also 'ask' for a given item:
items.index('foo')
# watchout for `ValueError` though!
# items.index('fifth')
```

```
['https:', '', 'colab.research.com', 'drive', 'asdfjhasdf', 'alfredo', 'orei
lly']
['drive', 'asdfjhasdf', 'alfredo', 'oreilly']
protocol is: https:
colab.research.com
oreilly
The first item is: first
```

Out[4]:

```
3
```

Tuples

Should be treated as "read only" lists, the differences are subtle!

In [8]:

```
ro_items = ('first', 'second', 'third')
print("first item in the tuple is: %s" % ro_items.index('first'))
print(ro_items[-1])
for item in ro_items:
    print(item)
```

```
first item in the tuple is: 0
third
first
second
third
```

In [9]:

```
# expect an error here, just like a list!
ro_items[9]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/var/folders/29/d5rl30vx2g914ldm2s662ft80000gn/T/ipykernel_49458/1353477027.
py in <module>
----> 1 ro_items[9]

IndexError: tuple index out of range
```

In [ ]:

```
# same with indexes
ro_items.index('fifth')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-cb8059424059> in <module>()
----> 1 ro_items.index('fifth')

ValueError: tuple.index(x): x not in tuple
```

In [12]:

```
# find out what methods are available in a tuple
for method in dir(tuple()):
  if method.startswith('__'):
    continue
  print(method)
```

```
count
index
```

In [14]:

```
# tuples are inmmutable
ro_items.append('a')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/var/folders/29/d5rl30vx2g914ldm2s662ft80000gn/T/ipykernel_49458/4263146556.
py in <module>
      1 # tuples are inmmutable
----> 2 ro_items.append('a')

AttributeError: 'tuple' object has no attribute 'append'
```

List Comprehensions

So easy to abuse!

In [ ]:

```
items = ['a', '1', '23', 'b', '4', 'c', 'd']
numeric = []
for item in items:
  if item.isnumeric():
    numeric.append(item)
print(numeric)
```

```
['1', '23', '4']
```

In [ ]:

```
# notice the `if` condition at the end, is this more readable? or less?
inlined_numeric = [item for item in items if item.isnumeric()]
inlined_numeric
```

Out[30]:

```
['1', '23', '4']
```

In [ ]:

```
# doubly nested items are usually targetted for list comprehensions
items = ['a', '1', '23', 'b', '4', 'c', 'd']
nested_items = [items, items]
nested_items
```

Out[31]:

```
[['a', '1', '23', 'b', '4', 'c', 'd'], ['a', '1', '23', 'b', '4', 'c', 'd']]
```

In [ ]:

```
numeric = []
for parent in nested_items:
    for item in parent:
      if item.isnumeric():
        numeric.append(item)
numeric
```

Out[32]:

```
['1', '23', '4', '1', '23', '4']
```

In [ ]:

```
# and now with list comprehensions
numeric = [item for item in parent for parent in nested_items if item.isnumer
numeric
```

Out[33]:

```
['1', '1', '23', '23', '4', '4']
```

In [ ]:

```
# this can improve readability
numeric = [
    item for item in parent
        for parent in nested_items
            if item.isnumeric()
]
numeric
```

Out[34]:

```
['1', '1', '23', '23', '4', '4']
```

The awesome dictionary

One of my favorite data structures in Python, learning it can yield inmense benefits.

In [ ]:

```
# dictionaries are mappings, usually referred to as key/value mappings
contacts = {
    'alfredo': '+3 678-677-0000',
    'noah': '+3 707-777-9191'
}
contacts
```

Out[36]:

```
{'alfredo': '+3 678-677-0000', 'noah': '+3 707-777-9191'}
```

In [ ]:

```
contacts['noah']
```

Out[37]:

```
'+3 707-777-9191'
```

In [ ]:

```
# you can get keys as list-like objects
contacts.keys()
```

Out[40]:

```
dict_keys(['alfredo', 'noah'])
```

In [ ]:

```
# or you can get the values as well
contacts.values()
```

Out[43]:

```
dict_values(['+3 678-677-0000', '+3 707-777-9191'])
```

In [ ]:

```
# looping over dictionaries default to `.keys()` and you can loop over both ke
for key in contacts:
  print(key)
for name, phone in contacts.items():
  print("Key: {0}, Value: {1}".format(name, phone))
```

```
alfredo
noah
Key: alfredo, Value: +3 678-677-0000
Key: noah, Value: +3 707-777-9191
```

In [ ]:

```
# you should treat dictionaries like a small database, with cheap (and fast!)
contacts['alfredo']
contacts['John']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-46-4b326074f145> in <module>()
      1 # you should treat dictionaries like a small database, with cheap (a
nd fast!) access
      2 contacts['alfredo']
----> 3 contacts['John']

KeyError: 'John'
```

In [ ]:

```
# super nice way to "fallback" when things do not exist
print(contacts.get('John', "Peter"))
try:
  contacts['John']
except KeyError:
  print("Peter")
```

```
Peter
Peter
```

Walking the filesystem, inspecting files

Python has built-in utilities to walk the filesystem. It is a bit clunky, and creating something useful requires stitching things together to produce good output

In [ ]:

```
import os

# yields the 'current' dir, then the directories, and then any files it finds
# for each level it traverses
for path_info in os.walk('.'):
    print(path_info)
    break
```

```
('.', ['.config', 'sample_data'], [])
```

In [ ]:

```
import os
from os.path import abspath, join

# producing absolute paths, instead of a tuple of three items
for top_dir, directories, files in os.walk('.'):
    for directory in directories:
        print(abspath(join(top_dir, directory)))
    for _file in files:
        print(abspath(join(top_dir, _file)))
    break
```

```
/content/.config
/content/sample_data
```

In [ ]:

```
# Now that absolute paths are shown, we can inspect them for file metadata

import os
from os.path import abspath, join, getsize

sizes = {}

for top_dir, directories, files in os.walk('.'):
    for _file in files:
        full_path = abspath(join(top_dir, _file))
        size = getsize(full_path)
        sizes[full_path] = size
        #break

sorted_results = sorted(sizes, key=sizes.get, reverse=True)

for path in sorted_results[:10]:
    print("Path: {0}, size: {1}".format(path, sizes[path]))
```

```
Path: /content/sample_data/mnist_train_small.csv, size: 36523880
Path: /content/sample_data/mnist_test.csv, size: 18289443
Path: /content/sample_data/california_housing_train.csv, size: 1706430
Path: /content/sample_data/california_housing_test.csv, size: 301141
Path: /content/.config/logs/2020.12.02/22.03.37.873126.log, size: 27136
Path: /content/.config/logs/2020.12.02/22.04.13.854338.log, size: 9917
Path: /content/sample_data/anscombe.json, size: 1697
Path: /content/sample_data/README.md, size: 930
Path: /content/.config/logs/2020.12.02/22.04.37.441505.log, size: 625
Path: /content/.config/logs/2020.12.02/22.04.38.150307.log, size: 620
```