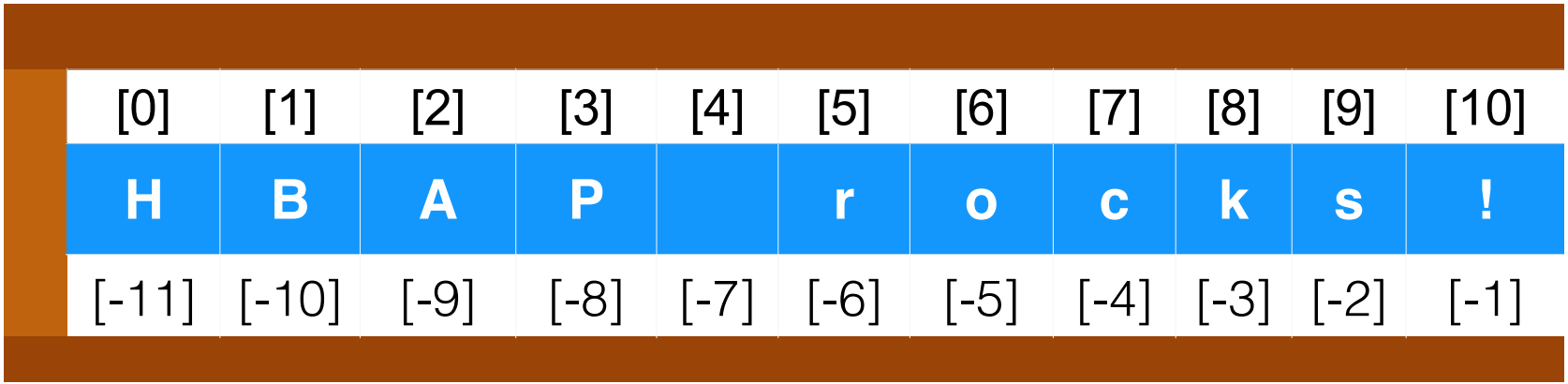
Python

Strings

Strings as Character Sequences

- Concatenation
- Repetition
- Indexing and Slicing

strings are immutable



- Searching

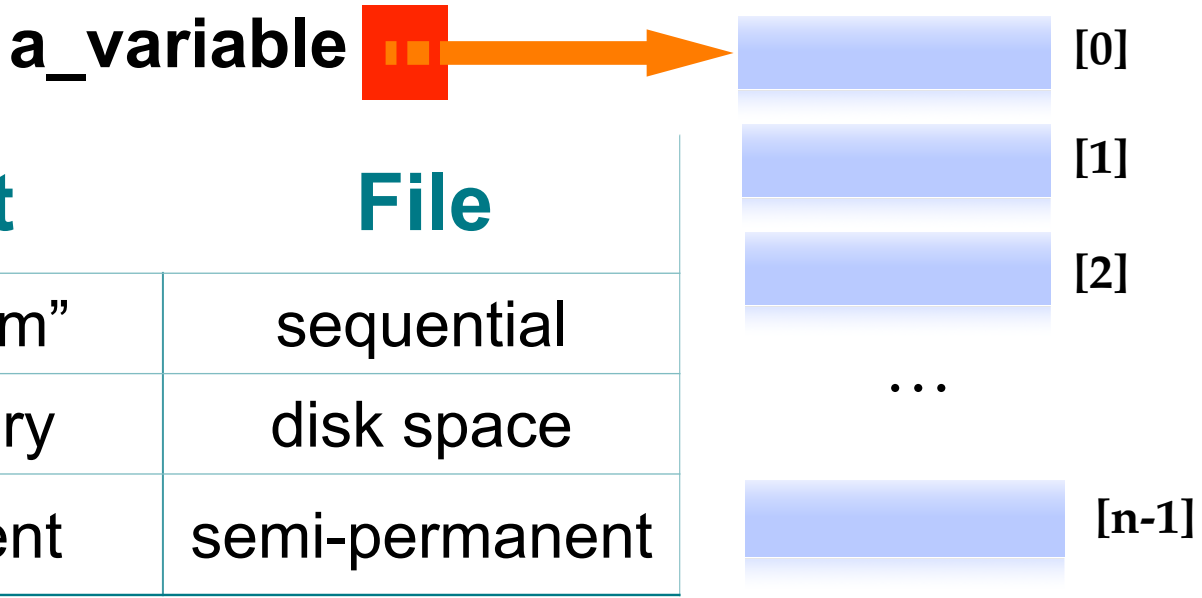
String Functions

See stringTest.py

Example	Explanation
a_string.center(w)	returns *a_string* evenly surrounded by spaces to make it w characters long
a_string.count(str)	# of occurrences of str in *a_string*
a_string.upper()	returns *a_string* in all uppercase
a_string.index(str)	returns index of first occurrence of str in *a_string*, or an error if not found
a_string.find(str)	returns index of first occurrence of str in *a_string*, or -1 if not found
a_string.replace(s1, s2)	returns a string with all occurrences of **s1** substring replaced by **s2** substring

Lists and Files

- Lists and files are important structures for collections of data that are all of the same type. Both consist of a sequence of elements:



	List	File
Access	"random"	sequential
Size	memory	disk space
Lifetime	transient	semi-permanent

Python

Lists

List Initialization & Traversal

- A *list* is an ordered, mutable sequential collection of heterogeneous objects, written as comma-delimited values enclosed in square brackets.
- Lists can be explicitly initialized when created, e.g.,
  - ```
    banks = ['JP Morgan', 'BofA', 'Wells Fargo',
             'Citigroup']
    ```
- Use square brackets to access a list element, e.g., **banks [2]**
- Every Python sequence has a function named **len**. What will the following code print?
  - ```
    for i in range (len(banks)) :
        print (len(banks[i])-1)
    ```

“List Mystery" Problem

- Traversal: An examination of each element of a list.
- What error occurs in the following? After fixing the problem, what values get printed at the end?

```
a = [1, 7, 5, 6, 4, 14, 11]
for i in range (len(a)):
    if a[i] > a[i + 1]:
        a[i + 1] = a[i + 1] * 2

print (a)
```

Common List Functions and Operators

Operation	Description
[] [$elem_1$, $elem_2$, ..., $elem_n$]	creates a new empty list, or a list that contains the initial n elements provided
len (*lst*)	returns the number of elements in list *lst*
list (*sequence*)	creates a new list containing all elements of the *sequence*
values_list * num	creates a new list by replicating the elements in the *values_list* **num** times
values_list + more_list	creates a new list by concatenating elements in both lists

More List Functions and Operators

Operation	Description
lst [from : to]	creates a sublist from a subsequence of elements in list **lst**, starting at position from and going through but not including position to. Both **from** and **to** are optional
sum (*lst*)	computes the sum of the values in list ***lst***
min (*lst*) max (*lst*)	returns the minimum or maximum value in list ***lst***
lst_1 == lst_2	tests whether two lists have the same elements, in the same order

3 Ways to Create a New List

Suppose we have `countries = ['USA', 'Canada', 'Mexico']`

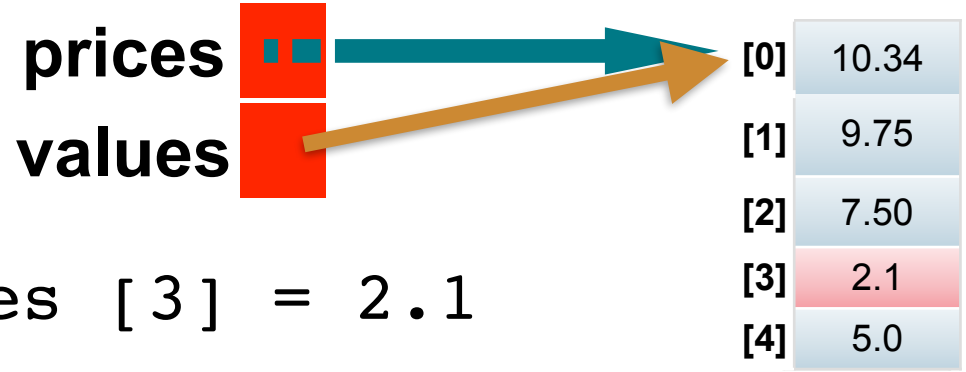
- Using the append function
  - `x = []`
  - `for c in countries:`
    `x.append (len(c))  # or x = x + [len(c)]`

- By creating and modifying a list …
  - `x = [0, 0, 0]`
  - `for i in range (len(countries)):`
  - `x[i] = len(countries[i])`

- Most Pythonic: using a “list comprehension” …
  - `[len(c) for c in countries]`

List References

- When you copy a list variable into another, both variables refer to the same list. The second variable is an alias for the first. For example,
- `prices = [10.34, 9.75, 7.50, 4.22, 5.0]`
- `values = prices`



- `values [3] = 2.1`

Python

Files and Command-Line Arguments

Opening and Closing Files

- `infile = open ('input.txt', 'r')`
  - use read() and readline() to input characters from a file
- `outfile = open ('output.txt', 'w')`
  - use write() method to output data into a file
  - use 'a' for appending output to existing file, not 'w'
- Close files after data is processed
  - `infile.close()`
  - `outfile.close()`

Command-Line Arguments

- When you run a Python program from the “command line,” you type the name of the program … but you can also type in additional information that the program can use. These additional strings are *command line arguments*.
- For example, in `python program.py -v input.dat`
  program.py receives 3 command line arguments: the strings "program.py", "-v" and “input.dat"
- Your program receives its command line arguments in the **argv** list defined in the **sys** module. In our example, the **argv** list has a length of 3 and contains these strings argv[0] == "program.py" argv[1] == "-v" argv[2] == "input.dat"

Python

CSV Files

Working with Spreadsheet Files

State	Cases	Deaths
California	4978363	72811
Texas	4255308	72337
Florida	3661182	60418
New York	2593360	56280
Pennsylvania	1597065	31915
Illinois	1725686	28877
Georgia	1605859	28430
New Jersey	1208321	28092

https://raw.githubusercontent.com/nytimes/covid-19-data/master/live/us-states.csv

- Most spreadsheet applications store their data in proprietary file formats. Fortunately, most can save a copy of the data in a portable format known as CSV (Comma-Separated Values).

File employees.csv

- A CSV file is simply a text file in which each row of the spreadsheet is stored as a line of text. The data values in each row are separated by commas. For example:

Manager	LastName	FirstName	Title	BirthDate	HireDate	Address	City
	Adams	Andrew	General Manager	2/18/1962	8/14/2002 0	11120 Jasper Ave NW	Edmonto
Michael Mitchell	Callahan	Laura	IT Staff	1/9/1968 0	3/4/2004 0:	923 7 ST NW	Lethbridg
Andrew Adams	Edwards	Nancy	Sales Manager	12/8/1958	5/1/2002 0:	825 8 Ave SW	Calgary
Nancy Edwards	Johnson	Steve	Sales Support Agent	3/3/1965 0	10/17/2003	7727B 41 Ave	Calgary
Michael Mitchell	King	Robert	IT Staff	5/29/1970	1/2/2004 0:	590 Columbia Boulevard We	Lethbridg
Andrew Adams	Mitchell	Michael	IT Manager	7/1/1973 0	10/17/2003	5827 Bowness Road NW	Calgary
Nancy Edwards	Park	Margaret	Sales Support Agent	9/19/1947	5/3/2003 0:	683 10 Street SW	Calgary
Nancy Edwards	Peacock	Jane	Sales Support Agent	8/29/1973	4/1/2002 0:	1111 6 Ave SW	Calgary

Working with CSV Files

- First: `import csv` to access the *reader* and *writer* methods
- Second: open the CSV file in the usual manner; e.g.,
  - `infile = open ("myCSVfile", "r")`
- Third: create a CSV reader: `r = csv.reader(infile)`
  - `r` is an "iterator object" for moving through the rows in the file. Each row is returned as a *list*, e.g.,

```
for row in r:
    print(row)
```

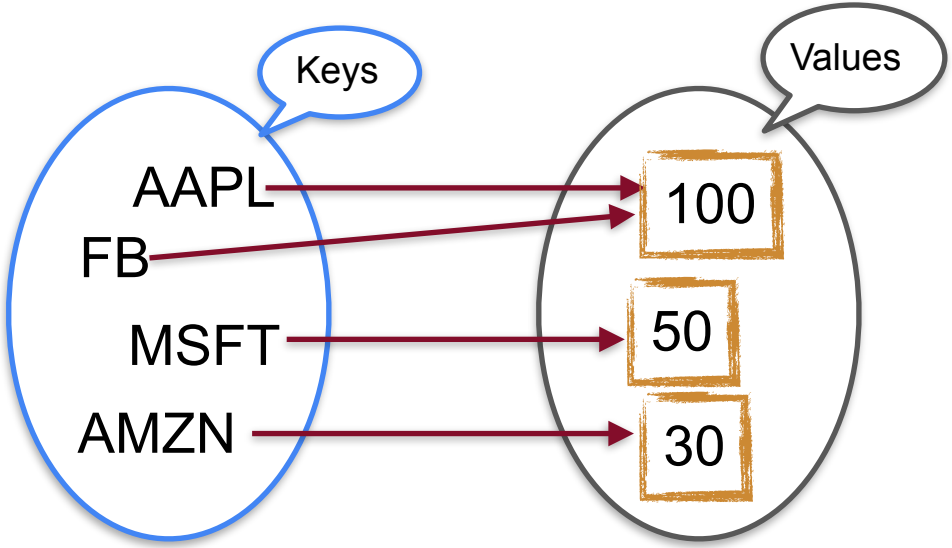
  - You can also skip a row: `next(r)`

Python

Dictionaries

Dictionaries

- A dictionary is a container that keeps associations between *keys* and *values*. Every *key* in the dictionary has an associated *value*. Keys are unique, but a value may be associated with several keys.
- Syntactically,
  - `empty_dict = {}`
  - ```
    fav_stocks = {
      'AAPL': 100,
      'MSFT': 50,
      'FB': 100,
      'AMZN': 30   }
    ```


    ```

Common Dictionary Operations

Operation	Returns
d = dict() *d* = dict(*c*)	Creates a new empty dictionary or a duplicate copy of dictionary c
d = {} *d* = {k_1:v_1, k_2:v_2, …, k_n:v_n}	Creates a new empty dictionary, or a dictionary that contains the initial items provided. Each item consists of a key (k) and a value (v) separated by a colon
len(*d*)	Returns the number of items in dictionary d
key `in` *d* *key* `not in` *d*	Determines if the key is in the dictionary, d

More Dictionary Operations

Operation	Returns
d[key] = value	Adds a new key/value item to the dictionary if the key does not exist. If the key does exist, it modifies the value associated with the key
x = d[key]	Returns the value associated with the given key. If the key does not exist, an *exception* occurs
***d*.pop(*key*)**	Removes the item associated with the given key and returns its value
d.values()	Returns a *sequence* containing all values of the dictionary
d.get(key, default)	Returns the value associated with the given key, or the default value if the key is not present

List vs. Dictionary

- It's nice to index an item of interest directly, where the index is not necessarily an integer

Index	A **List**
[0]	$element_1$
[1]	$element_2$
[2]	$element_3$
[3]	$element_4$
…	…

Index	A **Dictionary**
key_1	$value_1$
key_2	$value_2$
key_3	$value_3$
key_4	$value_4$
…	…

Disadvantage of Lists

- What happens if we sort one of these lists? Or if we want to retrieve a particular student's grade?

Index	name_list
[0]	"Mary"
[1]	"Henry"
[2]	"Arturo"
[3]	"David"
…	…

Index	grade_list
[0]	"A"
[1]	"C-"
[2]	"A-"
[3]	"Pass"
…	…

Dictionary Values and Keys

- **Values**
  - Can be any type (immutable and mutable)
  - Can be duplicates
  - Can be lists, even other dictionaries!

- **Keys**
  - Must be unique
  - Immutable type (int, float, string, bool, tuple)

- There's no order to keys or their values!

Processing CSV files

- **First, let's create a new CSV file by modifying an existing one**
  - Introduce the **del** operator
  - Illustrate **try - except** statement to make programs more robust

- **Second, let's modify file employees1.py**
  - Utilize a *DictReader* (available in CSV module)
  - Will allow us to reference fields by their names

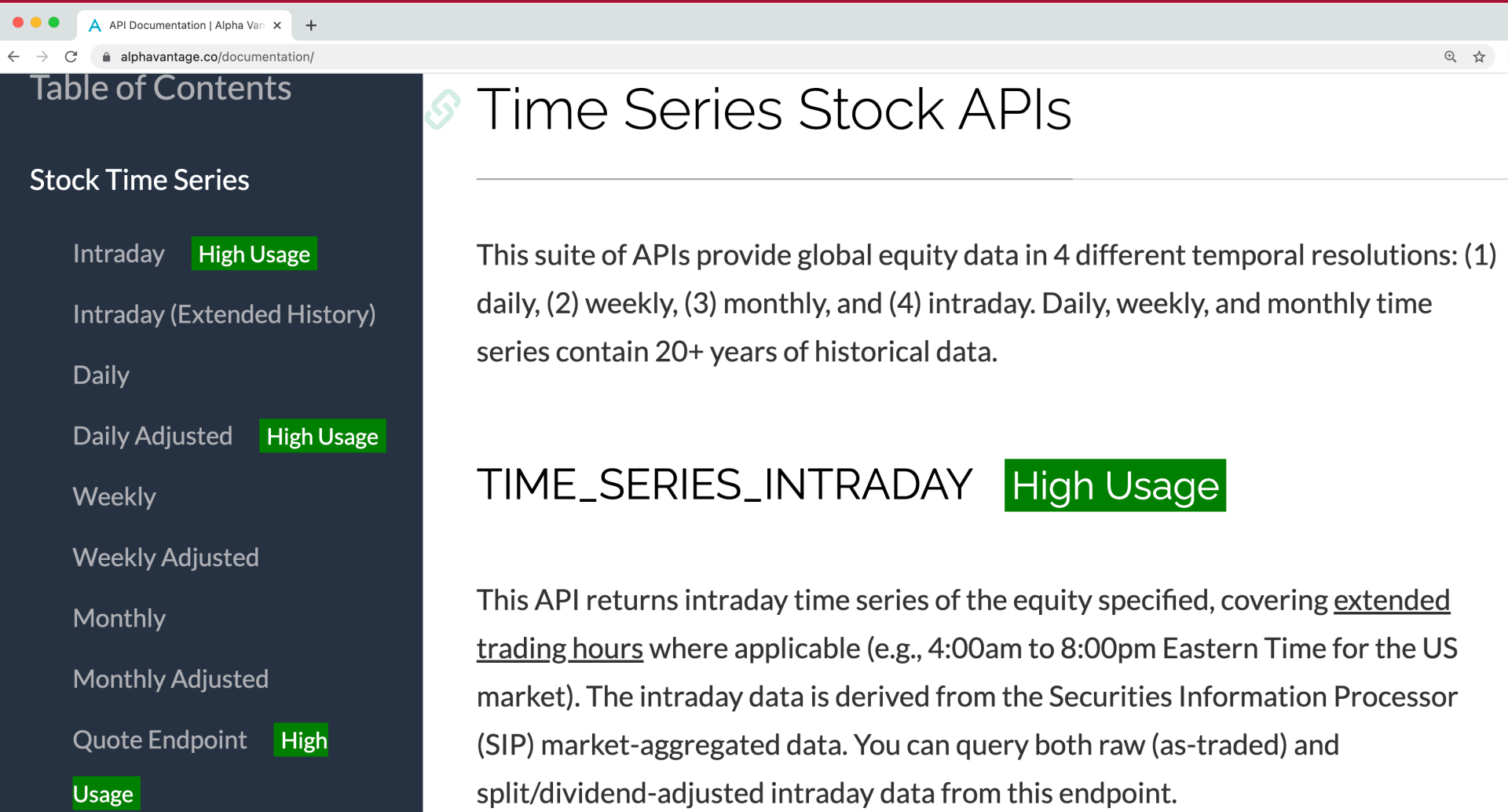
Python

Retrieving Web Data and APIs

How to Read Data from the Web

- The **requests** module is used to make HTTP requests in Python. It abstracts various complexities behind a simple API:
  - The **get** method indicates that you're trying to retrieve data from a specified resource. For example,

```
response = requests.get('some URL')
if response:
    print('Success!')
else:
    print('An error has occurred.')
```

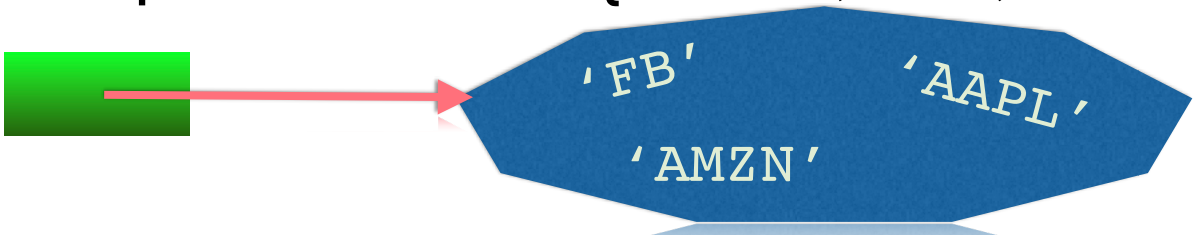

API Documentation | Alpha Van
alphavantage.co/documentation/
Table of Contents
Stock Time Series
Intraday High Usage
Intraday (Extended History)
Daily
Daily Adjusted High Usage
Weekly
Weekly Adjusted
Monthly
Monthly Adjusted
Quote Endpoint High Usage
Time Series Stock APIs
This suite of APIs provide global equity data in 4 different temporal resolutions: (1) daily, (2) weekly, (3) monthly, and (4) intraday. Daily, weekly, and monthly time series contain 20+ years of historical data.
TIME_SERIES_INTRADAY High Usage
This API returns intraday time series of the equity specified, covering extended trading hours where applicable (e.g., 4:00am to 8:00pm Eastern Time for the US market). The intraday data is derived from the Securities Information Processor (SIP) market-aggregated data. You can query both raw (as-traded) and split/dividend-adjusted intraday data from this endpoint.

Python

Sets

Sets

- A *set* stores a collection of unique values. Unlike lists, the members of a set are not stored in any particular order (they can't be accessed by position).
- Operations on sets are the same as the mathematical set operations, such as *intersection* and *union*.
- Simple example: stocks = {'AAPL', 'FB', 'AMZN'}



Creating and Using Sets

- You can also use the **set** function to convert any sequence in a set. For example,
  - `stock_symbols = ['AAPL', 'FB', 'AMZN']`
  - `stocks = set(stock_symbols)`
- You cannot use {} to create an empty set. Instead, use **set()**
- You can use:
  - the **len** function
  - the **in** operator
  - a **for** loop to iterate over all elements in a set

Manipulating Sets

- Like lists, sets are *mutable* containers, so you can add and remove elements:
  - `stocks.add('MSFT')`
  - `stocks.discard('TESLA')`
  - `stocks.remove('Foobar')`
- The **issubset** method returns *True* or *False* to report whether one set is a subset of another
- Methods union, intersection and difference also allow the use of operators |, &, - There is also ^

Python

Tuples

Tuples

- A *tuple* is very similar to a *list*, but once created, its contents cannot be modified.
- A tuple is created by specifying its contents as a comma-separated sequence. You can enclose the sequence in parentheses:
  `triple = (5, 10, 15)` If you prefer, you can omit the parentheses:
  `triple = 5, 10, 15`
- Any list operation that does not modify the content of a list can be used, e.g.,
  - `element = triple[0]`
  - `len(triple)`
  - `47 not in triple`

Tuple Capabilities

- A *tuple* can be used to swap the value of two variables:
  - ```
    (x, y) = (y, x)
    ```
- A *tuple* can return more than one value from a function:
  - ```
    def quot_and_remainder (top, bottom):
        quot = top // bottom
        remainder = top % bottom
        return (quot, remainder)
    ```
  - ```
    q, r = quot_and_remainder (34, 5)
    ```
    ```

Python Data Types

- **Simple**
  - *int* `17` `-3`
  - *float* `1.7` `3.2e2`
  - *bool* `True` `False`
  - *str* `"foobar"` `'foobar'`
  - *NoneType* `None`

- **Complex**
  - *range* `range(3, 22, 2)` `range(1, 10)` `range(10)`
  - *list* `[1, 3, 4, 4, 4, 5]`
  - *set* `{1, 3, 4, 4, 4, 5}` *#duplicates not allowed*
  - *dict* `ages = {'Dave':18, 'Mary':22}`
  - *tuple* `(0, 0)`