# Data Wrangling with Python

#### 2. Data Containers

#### c. Dictionaries

- Usually these dictionaries come with key-value pairs, the words you use for looking up are called as keys and the definitions of these words are called the values.
- Let's say we try to create a dictionary that uses animal types as keys and counts of each animal as values then, this is how it would look.

In [1]:
```
animal_counts = {'cats':2,'dogs':4,'horses':3}
```

- If we had to access one of the values in the dictionary, that would be possible like this.

In [2]:
```
animal_counts['horses']
```
Out[2]:
```
3
```

- We can use lists within dictionaries too, let's say we had to create a dictionary using lists we can do it by two ways

#### 1st method

In [7]:
```
animal_names = {
    'cats': ['Tom','Bom'],
    'dogs': ['Rock','Sock','Mat','Rocky'],
    'horses': ['Bam','Ham','Sam']
}
```

In [8]:
```
animal_names
```
Out[8]:
```
{'cats': ['Tom', 'Bom'],
 'dogs': ['Rock', 'Sock', 'Mat', 'Rocky'],
 'horses': ['Bam', 'Ham', 'Sam']}
```

#### 2nd method

- We can assign the lists to variables and then put those variables in dictionaries.

In [9]:
```
cat_names = ['Tom','Bom']
dog_names = ['Rock','Sock','Mat','Rocky']
horses_names = ['Bam','Ham','Sam']
```

In [11]:
```
names_animals = {
    'cats' : cat_names,
    'dogs' : dog_names,
    'horses' : horses_names
}
```

In [12]:
```
names_animals
```

Out[12]:
```
{'cats': ['Tom', 'Bom'],
 'dogs': ['Rock', 'Sock', 'Mat', 'Rocky'],
 'horses': ['Bam', 'Ham', 'Sam']}
```

## Methods

### String Methods

- Think of the data types as nouns and the things they can do as verbs. The things that data types can do are called methods.

#### a. strip()

In [13]:
```
filename = 'budget    '
```

- Now this thing has a space now to remove the space we can use .strip() function and that would only reflect temporarily and the changes won't be saved.

In [14]:
```
filename.strip()
```

Out[14]:
```
'budget'
```

In [15]:
```
filename
```

Out[15]:
```
'budget    '
```

- Inorder for the changes to reflected assign it back to the variable using that method

In [16]:
```
filename = filename.strip()
```

In [17]:
```
filename
```

Out[17]:
```
'budget'
```

**b. upper()**

- Just like the above method this is also a built-in method to turn the string stored in the variable to upper case.

```
In [18]: filename

Out[18]: 'budget'

In [19]: filename = filename.upper()

In [20]: filename

Out[20]: 'BUDGET'
```

**Numerical Methods**

- Just like how strings are stored we can store mathematical calculations into a variable.

```
In [21]: name = 'Abhijith'+'Kasula'

In [22]: name

Out[22]: 'AbhijithKasula'

In [23]: details = ['1999','BLR','IN']+['CVG','USA']

In [24]: details

Out[24]: ['1999', 'BLR', 'IN', 'CVG', 'USA']

In [25]: personality = ['Winner','Positive','Talented','Powerful']-['Weak','Negativ

         ---------------------------------------------------------------------------
         --
         TypeError                                 Traceback (most recent call las
         t)
         ~\AppData\Local\Temp\ipykernel_19640\1687228455.py in <module>
         ----> 1 personality = ['Winner','Positive','Talented','Powerful']-['Wea
         k','Negative']

         TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

- As we can notice we can only perform addition in the list not subtraction but what if all the elements are available in the list and we remove them.

In [26]:
```
personality = ['Winner','Positive','Talented','Powerful','Weak','Negative'
```

In [27]:
```
personality = personality - ['Weak','Negative']
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_19640\3064431878.py in <module>
----> 1 personality = personality - ['Weak','Negative']

TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

- As the interpreter is not able to remove the unwanted traits, you must remove it on your own like how any user would remove it.

In [28]:
```
personality = ['Winner','Positive','Talented','Powerful']
```

In [29]:
```
personality
```

Out[29]:
```
['Winner', 'Positive', 'Talented', 'Powerful']
```

**List Methods**

- We can add values to a list using append(), let's create an empty list and add elements into it.

In [30]:
```
friends = []
```

In [31]:
```
friends.append('Bhargav')
```

In [32]:
```
friends
```

Out[32]:
```
['Bhargav']
```

In [33]:
```
friends.append('Sathyanath','Hemanth')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_19640\811441899.py in <module>
----> 1 friends.append('Sathyanath','Hemanth')

TypeError: list.append() takes exactly one argument (2 given)
```

- We can notice that we can't add multiple elements using .append(), we can use .remove() to remove elements from the list.

In [34]:
```
friends.remove('Bhargav')
```

In [35]:
```
friends
```

Out[35]:
```
[]
```

### Dictionary Methods

- The same thing can be for dictionary too, we can create an empty dictionary and add elements into the dictionary like this:

In [36]:
```
friends = {}
```

In [37]:
```
friends['BLR']=5
```

In [38]:
```
friends['CVG']=7
friends['BOS']=3
```

In [39]:
```
friends
```

Out[39]:
```
{'BLR': 5, 'CVG': 7, 'BOS': 3}
```

- To just check the keys of a dictionary we can do it by:

In [40]:
```
friends.keys()
```

Out[40]:
```
dict_keys(['BLR', 'CVG', 'BOS'])
```

- Let's try getting the values of the dictionary

In [41]:
```
friends.values()
```

Out[41]:
```
dict_values([5, 7, 3])
```

- Just get the value stored in a key using:

In [42]:
```
friends['BLR']
```

Out[42]:
```
5
```

- We can store the value in a separate variable

In [43]:
```
friends_count = friends['BLR']
```

In [44]:
```
friends_count
```

Out[44]:
```
5
```

## Helpful Tools

#### type()

- This can be used to extract the datatype of an object.

In [45]:
```
type(friends)
```

Out[45]:
```
dict
```

#### dir()

- Let's say you don't know what all methods are built in a particular data type then we can do find out by dir() that returns a list of built-in methods and properties.

In [46]:
```
dir(friends)
```

Out[46]:
```
['__class__',
 '__class_getitem__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__ror__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']
```

- Now I want to get a brief information about what a specific method does we can do it by .help() function.

```
In [50]: help(friends.clear)
```

```
Help on built-in function clear:

clear(...) method of builtins.dict instance
    D.clear() -> None.  Remove all items from D.
```

- We can notice that it can give out a prompt and says we can remove items from a dictionary.

```
In [51]: friends
```

```
Out[51]: {'BLR': 5, 'CVG': 7, 'BOS': 3}
```

```
In [53]: friends.clear()
```

```
In [54]: friends
```

```
Out[54]: {}
```

```
In [55]: type(friends)
```

```
Out[55]: dict
```

- We can notice that it has removed all the elements from the dictionary and it is empty.

---

# Chapter - 3

#### Machine readable files

- There are user readable files and machine readable files that are helpful.
- Machine readable files are usually CSV, JSON and XML files.
- Comma-Separated Values (CSV)
- JavaScript Object Notation (JSON)
- Extensible Markup Language (XML)

#### CSV Data

- In this type of file the values are separated by commas and then there is another type with similar format and that is TSV.

- TSV stands for Tab Separated Values and it is similar to comma separated values but the

### Importing a CSV file

- I have downloaded the csv file locally from github repo and then stored it in the same directory as of the notebook and then wrote this piece of code.

In [56]:
```
import csv
```

- A Python library is a package of code that provides functionality you can use in your Python programs. The **csv** library we are importing comes with your Python installation as part of the standard library

In [60]:
```
csvfile = open('data-text.csv','r')
```

- The second line of code takes our data-text.csv file, which should be located in the same folder as the script, and passes it to the open function.
- There are different modes in which a file can be opened that can be in read-only using 'r' and write mode using 'w' and 'rb' for read-only binary file and 'wb' for write in binary mode.
- If 'b' is not included it will be opened in text-mode, I tried opening the file in 'rb' mode it gave me this error (https://stackoverflow.com/questions/8515053/csv-error-iterator-should-return-strings-not-bytes)

In [61]:
```
reader = csv.reader(csvfile)
```

- We pass csvfile to the reader function in the csv module. This function tells the csv module to read the open file as a CSV:

In [62]:
```
for row in reader:
    print(row)
```

```
['Indicator', 'PUBLISH STATES', 'Year', 'WHO region', 'World Bank inco
me group', 'Country', 'Sex', 'Display Value', 'Numeric', 'Low', 'Hig
h', 'Comments']
['Life expectancy at birth (years)', 'Published', '1990', 'Europe', 'H
igh-income', 'Andorra', 'Both sexes', '77', '77.00000', '', '', '']
['Life expectancy at birth (years)', 'Published', '2000', 'Europe', 'H
igh-income', 'Andorra', 'Both sexes', '80', '80.00000', '', '', '']
['Life expectancy at age 60 (years)', 'Published', '2012', 'Europe',
'High-income', 'Andorra', 'Female', '28', '28.00000', '', '', '']
['Life expectancy at age 60 (years)', 'Published', '2000', 'Europe',
'High-income', 'Andorra', 'Both sexes', '23', '23.00000', '', '', '']
['Life expectancy at birth (years)', 'Published', '2012', 'Eastern Med
iterranean', 'High-income', 'United Arab Emirates', 'Female', '78', '7
8.00000', '', '', '']
['Life expectancy at birth (years)', 'Published', '2000', 'Americas',
'High-income', 'Antigua and Barbuda', 'Male', '72', '72.00000', '',
'', '']
['Life expectancy at age 60 (years)', 'Published', '1990', 'Americas',
'High-income', 'Antigua and Barbuda', 'Male', '17', '17.00000', '',
```

- A for loop is a way of iterating over Python objects, commonly used with lists. A for loop tells Python code, “For each thing in this list of things, do something.”

**Editing the code and trying to read the csv file as dictionary**

In [69]:
```
import csv

csvfile = open('data-text.csv','r')
reader = csv.DictReader(csvfile)

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

```
{'Indicator': 'Life expectancy at birth (years)', 'PUBLISH STATES': 'P
ublished', 'Year': '1990', 'WHO region': 'Europe', 'World Bank income
group': 'High-income', 'Country': 'Andorra', 'Sex': 'Both sexes', 'Dis
play Value': '77', 'Numeric': '77.00000', 'Low': '', 'High': '', 'Comm
ents': ''}
{'Indicator': 'Life expectancy at birth (years)', 'PUBLISH STATES': 'P
ublished', 'Year': '2000', 'WHO region': 'Europe', 'World Bank income
group': 'High-income', 'Country': 'Andorra', 'Sex': 'Both sexes', 'Dis
play Value': '80', 'Numeric': '80.00000', 'Low': '', 'High': '', 'Comm
ents': ''}
{'Indicator': 'Life expectancy at age 60 (years)', 'PUBLISH STATES':
'Published', 'Year': '2012', 'WHO region': 'Europe', 'World Bank incom
e group': 'High-income', 'Country': 'Andorra', 'Sex': 'Female', 'Displ
ay Value': '28', 'Numeric': '28.00000', 'Low': '', 'High': '', 'Commen
ts': ''}
{'Indicator': 'Life expectancy at age 60 (years)', 'PUBLISH STATES':
'Published', 'Year': '2000', 'WHO region': 'Europe', 'World Bank incom
e group': 'High-income', 'Country': 'Andorra', 'Sex': 'Both sexes', 'D
isplay Value': '23', 'Numeric': '23.00000', 'Low': '', 'High': '', 'Co
```

- we have successfully imported the CSV data into Python, meaning we were able to get the data from the file into a usable format Python can understand.

## JSON Data

- JSON stands for JavaScript Object Notation and it is the most commonly used formats for data transfers. It is clean, easy to read and easy to parse.
- Many websites use this format to transmit data to the javaScript on the page. Many sites have JSON enabled API's.

#### Importing data from a JSON file

In [71]:

```python
import json

json_data = open('data-text.json').read()

data = json.loads(json_data)

for item in data:
    print(item)
```

```
{'Indicator': 'Life expectancy at birth (years)', 'PUBLISH STATES': 'P
ublished', 'Year': 1990, 'WHO region': 'Europe', 'World Bank income gr
oup': 'High-income', 'Country': 'Andorra', 'Sex': 'Both sexes', 'Displ
ay Value': 77, 'Numeric': 77.0, 'Low': '', 'High': '', 'Comments': ''}
{'Indicator': 'Life expectancy at birth (years)', 'PUBLISH STATES': 'P
ublished', 'Year': 2000, 'WHO region': 'Europe', 'World Bank income gr
oup': 'High-income', 'Country': 'Andorra', 'Sex': 'Both sexes', 'Displ
ay Value': 80, 'Numeric': 80.0, 'Low': '', 'High': '', 'Comments': ''}
{'Indicator': 'Life expectancy at age 60 (years)', 'PUBLISH STATES':
'Published', 'Year': 2012, 'WHO region': 'Europe', 'World Bank income
group': 'High-income', 'Country': 'Andorra', 'Sex': 'Female', 'Display
Value': 28, 'Numeric': 28.0, 'Low': '', 'High': '', 'Comments': ''}
{'Indicator': 'Life expectancy at age 60 (years)', 'PUBLISH STATES':
'Published', 'Year': 2000, 'WHO region': 'Europe', 'World Bank income
group': 'High-income', 'Country': 'Andorra', 'Sex': 'Both sexes', 'Dis
play Value': 23, 'Numeric': 23.0, 'Low': '', 'High': '', 'Comments':
''}
{'Indicator': 'Life expectancy at birth (years)', 'PUBLISH STATES': 'P
ublished', 'Year': 2012, 'WHO region': 'Eastern Mediterranean', 'World
```

**What we did in the above code was:**

- Imported the json library to process the JSON file.
- Used open function in python to open the file.
- Used json.loads() to load the data into a variable.
- Used for loop to iterate over and print each row to the output.