

Python Basics for Data Science

Begin your course today

Start course

Welcome to Python Basics for Data Science!

We are glad you are taking this course! The team spent a lot of time developing this Python Basics for Data Science course so everyone can learn to program with this incredibly powerful language.

As you may know, Python has become one of the top languages used not only by developers, but also by data scientists, data engineers, and researchers alike. It lends itself well to creating applications and analyzing big data.

So we hope you enjoy this beginner's course in Python for Data Science.

Diemice

About this course

 **General Information** 1 min

 **Learning Objectives** 1 min

 **Syllabus** 2 min

 **Grading Scheme** 1 min

 **Change Log** 1 min

Module 1 - Python Basics

- [Module Introduction and Learning Objectives](#) 1 min
- [Video: Types \(3:02\)](#) 4 min
- [Practice Quiz: Types](#) 1 activity
- [Video: Expressions and Variables \(3:55\)](#) 5 min
- [Hands-On Lab: Your First Program, Types, Expressions, and Variables](#) 1 min + 1 activity
- [Practice Quiz: Expressions and Variables](#) 1 activity
- [Video: String Operations \(3:58\)](#) 5 min
- [Hands-On Lab: Strings](#) 1 min + 1 activity
- [Practice Quiz: String Operations](#) 1 activity
- [Graded Quiz: Python Basics \(5 Questions\)](#) 1 activity
Graded Quiz due Jan 8, 2022, 11:49 PM GMT+8

Module 2 - Python Data Structures

- [Module Introduction and Learning Objectives](#) 1 min
- [Video: Lists and Tuples \(8:51\)](#) 10 min
- [Hands-On Lab: Lists](#) 1 min + 1 activity
- [Hands-On Lab: Tuples](#) 1 min + 1 activity
- [Practice Quiz: Lists and Tuples](#) 1 activity
- [Video: Dictionaries \(2:25\)](#) 3 min
- [Hands-On Lab: Dictionaries](#) 1 min + 1 activity
- [Practice Quiz: Dictionaries](#) 1 activity
- [Video: Sets \(5:17\)](#) 6 min
- [Hands-On Lab: Sets](#) 1 min + 1 activity
- [Discussion Prompt: Python Data Structures](#) 1 min + 1 activity
- [Practice Quiz: Sets](#) 1 activity
- [Graded Quiz: Python Data Structures \(5 Questions\)](#) 1 activity
Graded Quiz due Jan 11, 2022, 12:55 PM GMT+8

Module 3 - Python Programming Fundamentals

- ✓ Module Introduction and Learning Objectives 1 min
 - ✓ Video: Conditions and Branching (10:17) 11 min
 - ✓ Hands-on Lab: Conditions and Branching 1 min + 1 activity
 - ✓ Practice Quiz: Conditions and Branching 1 activity
 - ✓ Video: Loops (6:45) 7 min
 - ✓ Hands-on Lab: Loops 1 min + 1 activity
 - ✓ Practice Quiz: Loops 1 activity
 - ✓ Video: Functions (13:31) 14 min
 - ✓ Hands-on Lab: Functions 1 min + 1 activity
 - ✓ Practice Quiz: Functions 1 activity
 - ✓ Video: Exception Handling (3:49) 4 min
 - ✓ Hands-On Lab: Exception Handling 1 min + 1 activity
 - ✓ Practice Quiz: Exception Handling 1 activity
 - ✓ Video: Objects and Classes (10:52) 11 min
 - ✓ Hands-on Lab: Objects and Classes 1 min + 1 activity
 - ✓ Discussion Prompt: Classes in Python 1 min + 1 activity
 - ✓ Practice Quiz: Objects and Classes 1 activity
 - ✓ Graded Quiz: Python Programming Fundamentals (5 Questions) 1 activity
Graded Quiz due Jan 14, 2022, 2:00 AM GMT+8
-

Module 4 - Working with Data in Python

- ✓ Module Introduction and Learning Objectives 1 min
- ✓ Video: Reading Files with Open (3:43) 4 min
- ✓ Hands-On Lab: Reading Files with Open 1 min + 1 activity
- ✓ Video: Writing Files with Open (2:54) 3 min
- ✓ Hands-On Lab: Writing Files with Open 1 min + 1 activity
- ✓ Practice Quiz: Reading & Writing Files with Open 1 activity
- ✓ Video: Loading Data with Pandas 4 min
- ✓ Video: Pandas: Working with and Saving Data (2:06) 3 min
- ✓ Hands-on Lab: Pandas with IBM Watson Studio 1 min
- ✓ Practice Quiz: Pandas 1 activity
- ✓ Video: One Dimensional NumPy (11:23) 12 min
- ✓ Hands-On Lab: One Dimensional Numpy 1 min + 1 activity
- ✓ Video: Two Dimensional NumPy (7:13) 8 min
- ✓ Hands-On Lab: Two Dimensional Numpy 1 min + 1 activity
- ✓ Practice Quiz: Numpy in Python 1 activity
- ✓ Graded Quiz: Working with Data in Python (5 Questions) 1 activity

Graded Quiz due Jan 16, 2022, 3:06 PM GMT+8

✓ Module 5 - APIs and Data Collection

- ✓ Module Introduction and Learning Objectives 1 min
 - ✓ Video: Simple APIs (Part 1) (5:12) 6 min
 - ✓ Video: Simple APIs (Part 2) (5:06) 6 min
 - ✓ Hands-on Lab: Instruction for Speech to Text and Language Translator API Keys 1 min
 - ✓ Hands-On Lab: Introduction to API 1 min + 1 activity
 - ✓ Hands-On Lab: Watson Speech to Text and Language Translator API 1 min + 1 activity
 - ✓ Practice Quiz: Simple APIs 1 activity
 - ✓ Video: REST APIs & HTTP Requests - Part 1 (4:11) 5 min
 - ✓ Video: REST APIs & HTTP Requests - Part 2 (4:56) 5 min
 - ✓ Hands-on Lab: Access REST APIs & Request HTTP 1 min + 1 activity
 - ✓ Optional Video: HTML for Webscraping () 5 min
 - ✓ Video: Webscraping (4:59) 5 min
 - ✓ Hands-on Lab: Web Scraping 1 min + 1 activity
 - ✓ Video: Working with different file formats (csv, xml, json, xlsx) (4:11) 5 min
 - ✓ Hands-on Lab: Working with different file formats 1 min + 1 activity
 - ✓ Discussion Prompt: JSON and HTTP 1 min + 1 activity
 - ✓ Practice Quiz: REST APIs, Webscraping, and Working with Files 1 activity
 - ✓ Graded Quiz: APIs, and Data Collection (5 Questions) 1 activity
Graded Quiz due Jan 19, 2022, 4:11 AM GMT+8
-

 **Final Exam**

 **Instructions** 1 min

 **Final Exam (25 Questions)** 1 activity

Timed Exam due Jan 21, 2022, 5:17 PM GMT+8

 **Python Cheat Sheet**

 **Python Cheat Sheet: The Basics** 1 min

 **Course Rating**

 **Badge**

 **Credits and Acknowledgments**

Module Introduction and Learning Objectives

[Bookmark this page](#)

This module teaches the basics of Python and begins by exploring some of the different data types such as integers, real numbers, and strings. Continue with the module and learn how to use expressions in mathematical operations, store values in variables, and the many different ways to manipulate strings.

LEARNING OBJECTIVES

In this lesson, you will learn the basics of Python and you will write your first Python program:

- Demonstrate an understanding of types in python by converting/casting data types: strings, floats, integers.
- Interpret variables and solve expressions by applying mathematical operations.

TYPES

A type is how Python represents different types of data. In this video, we will discuss some widely used types in Python.

You can have different types in Python. They can be integers like 11, real numbers like 21.213, they can even be words.

Integers, real numbers, and words can be expressed as different data types. The following chart summarizes three data types for the last examples.

The first column indicates the expression. The second column indicates the data type.

We can see the actual data type in Python by using the type command. We can have int, which stands for an integer and

float that stands for float, essentially a real number. The type string is a sequence of characters.

Here are some integers.

- Integers can be negative or positive. It should be noted that there is a finite range of integers

but it is quite large.

- Floats are real numbers. They include the integers but also numbers in between the integers.

Consider the numbers between 0 and 1. We can select numbers in between them.

These numbers are floats. Similarly, consider the numbers between 0.5 and 0.6.

We can select numbers in between them. These are floats as well. We can continue the process zooming in for different

numbers. Of course there is a limit but it is quite small. You can change the type of the expression in Python, this is called typecasting. You can convert an int to a float.

For example, you can convert or cast the integer 2 to a float 2.0.

Nothing really changes, if you cast a float to an integer, you must be careful.

For example, if you cast the float 1.1 to 1, you will lose some information. If a string contains an integer value, you can

convert it to int. If we convert a string that contains a non-integer value, we get an error. Check out more examples in

the lab. You can convert an int to a string or a float to a string.

Boolean is another important type in Python. A Boolean can take on two values. The first value is True, just remember we use an uppercase T.

Boolean values can also be False with an uppercase F. Using the type command on a Boolean value, we obtain the term

bool. This is short for Boolean, if we cast a Boolean True to an integer or float, we will get a 1.

If we cast a Boolean False to an integer or float, we get a 0. If you cast a 1 to a Boolean, you get a True.

Similarly, if you cast a 0 to a Boolean, you get a False. Check the labs for more examples or check Python.org for other

kinds of types in Python.



Types

| | |
|--------------------|-------|
| 11 | int |
| 21.213 | float |
| "Hello Python 101" | str |

Types

| | |
|--------------------------|-------|
| type(11) | int |
| type(21.213) | float |
| type("Hello Python 101") | str |

Types: int

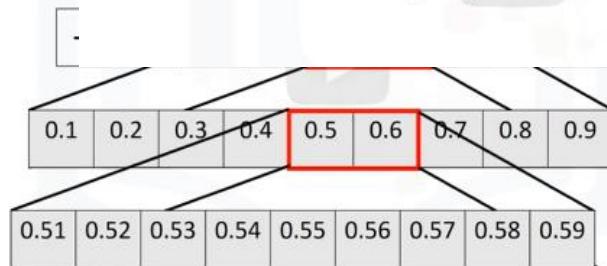


Types

Types:

`str(1): "1"`

`str(4.5): '4.5'`



Types

`float(2):2.0`

`int(1.1):1`

`int('1'):1`

~~`int('A')`~~

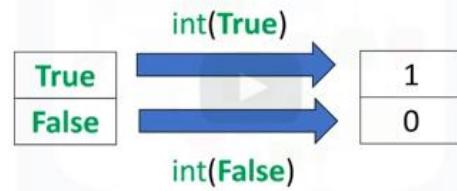
Types

True

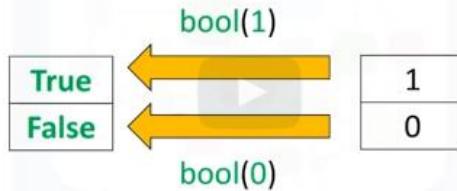
False

type(True):bool

Types



Types



Practice Quiz: Types

Bookmarked

Question 1

1/1 point (ungraded)

What is the type of the following: 0

Int

float



Answer

Correct:

Correct, as there is no decimal the number is of type int. You can also use the type function to verify your results.

Submit

You have used 1 of 2 attempts

Save

✓ Correct (1/1 point)

Question 2

1/1 point (ungraded)

What is the type of the following number: 3.12323

Int

Float



Answer

Correct:

Correct, as there is a decimal the number is of type float. You can also use the type function to verify your results.

Submit

You have used 1 of 2 attempts

Save

✓ Correct (1/1 point)

Question 3

1/1 point (ungraded)

What is the result of the following: `int(3.99)`

3.99

3



Answer

Correct:

Correct. In Python, if you cast a float to an integer, the conversion truncates towards zero, i.e. you just get rid of the numbers after the decimal place. So for 3.99 you just get rid of the ".99" leaving 3.

Submit

You have used 1 of 2 attempts

Save

✓ Correct (1/1 point)

Expressions and Variables

Expressions describe a type of operation the computers perform. Expressions are operations the

python performs. For example, basic arithmetic operations like adding multiple numbers.

The result in this case is 160. We call the numbers operands, and the math symbols in this case,

addition, are called operators.

We can perform operations such as traction using the subtraction sign. In this case, the result is

a negative number. We can perform multiplication operations using the asterisk. The result is

25. In this case, the operands are given by negative and asterisk. We can also perform division

with the forward slash- 25 / 5 is 5.0; 25 / 6 is approximately 4.167.

In Python 3, the version we will be using in this course, both will result in a float.

We can use the double slash for integer division, where the result is rounded.

Be aware, in some cases the results are not the same as regular division.

Python follows mathematical conventions when performing mathematical expressions.

The following operations are in a different order. In both cases, Python performs multiplication,

then addition, to obtain the final result. There are a lot more operations you can do with Python,

check the labs for more examples.

We will also be covering more complex operations throughout the course.

The expressions in the parentheses are performed first.

We then multiply the result by 60. The result is 1,920. Now, let's look at variables.

We can use variables to store values. In this case, we assign a value of 1 to the variable `my_variable` using the assignment operator, i.e, the equal sign.

We can then use the value somewhere else in the code by typing the exact name of the variable.

We will use a colon to denote the value of the variable. We can assign a new value to `my_variable` using the assignment operator.

We assign a value of 10. The variable now has a value of 10. The old value of the variable is not

important.

We can store the results of expressions. For example, we add several values and assign the result to `x`. `X` now stores the result. We can also perform operations on `x` and save the result to

a new variable-`y`. `Y` now has a value of 2.666. We can also perform operations on `x` and assign

the value `x`. The variable `x` now has a value: 2.666.

As before, the old value of `x` is not important. We can use the type command in variables as well. It's good practice to use meaningful variable names; so, you don't have to keep track of

what the variable is doing. Let say, we would like to convert the number of minutes in the highlighted examples to number of hours in the following music data-set. We call the variable, that contains the total number of minutes, `total_min`. It's common to use the underscore to represent the start of a new word. You could also use a capital letter.

We call the variable that contains the total number of hours, `total_hour`. We can obtain the total

number of hours by dividing `total_min` by 60. The result is approximately 2.367 hours.

If we modify the value of the first variable, the value of the variable will change.

The final result values change accordingly, but we do not have to modify the rest of the code.



Expressions: Mathematical Operations

operands

43 + 60 + 16 + 41

160

Expressions: Mathematical Operations

operators

$$43 \boxed{+} 60 \boxed{+} 16 \boxed{+} 41$$

160

Expressions: Mathematical Operations

$$50 \boxed{-} 60$$

-10

$$5 \boxed{*} 5$$

25

Expressions: Mathematical Operations

`25 / 5`

5.0

`25 / 6`

4.166..

Expressions: Mathematical Operations

`25 // 5`

5

`25 // 6`

`25 / 6`

4

4.166..

Expressions: Mathematical Operations

120
`2*60 + 30`

120
`30 + 2*60`

Expressions: Mathematical Operations

$$\begin{array}{c} 32 \\ (30+2)*60 \end{array}$$

1920

Variables

my_variable=1

my_variable:1



Variables

my_variable=10

my_variable:10



Variables

my_variable=10

my_variable:10



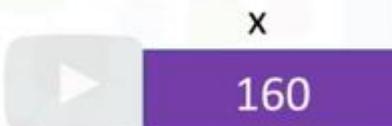
my_variable

10

Variables

x=43 + 60 + 16 + 41

x:160



Variables

x=160

y=x/60

y:2.6666..



Variables

x=160

x=x/60

x:2.6666..

type(x):float

x

2.666..

Variables

| Artist | Album | Released | Length | Genre | Music recording sales [millions] | Claimed sales [millions] | Released | Soundtrack | Rating (imdb) |
|-----------------|---------------------------------|----------|----------|-----------------------------|----------------------------------|--------------------------|-----------|------------|---------------|
| Michael Jackson | Thriller | 1982 | 00:42:19 | Pop, rock, R&B | 40 | 50 | 30-Nov-82 | | 10.0 |
| AC/DC | Back in Black | 1980 | 00:42:11 | HARD ROCK | 26.1 | 30 | 25-Jul-80 | | 8.5 |
| Pink Floyd | The Dark Side of the Moon | 1973 | 00:42:02 | Progressive rock | 24.2 | 45 | 01-Mar-73 | | 8.5 |
| Whitney Houston | The Bodyguard | 1992 | 00:57:44 | Soundtrack,R&B, soul, pop | 26.1 | 30 | 25-Jul-92 | Y | 7.0 |
| Metallica | But Out of Hell | 1987 | 00:48:02 | HARD ROCK, progressive rock | 20.6 | 43 | 21-Oct-87 | | 7.0 |
| Eagles | Their Greatest Hits (1971-1975) | 1976 | 00:43:08 | Rock, soft rock, adult rock | 32.2 | 43 | 17-Feb-76 | | 8.5 |
| Bon Jovi | Saturday Night Fever | 1977 | 00:45:54 | Disco | 20.6 | 40 | 10-Nov-77 | Y | 9.0 |
| Paul McCartney | Humorous | 1977 | 00:40:01 | Soft rock | 27.9 | 40 | 06-Feb-77 | | 8.5 |

Variables

total_min = 43 + 42 + 57



total_min
142 / 60

total_hr = total_min / 60



total_hr

2.367

total_hr:2.367

Variables

total_min = 43 + 42



total_min
85 / 60

total_hr = total_min / 60



total_hr

1.4167

total_hr:1.4167



**IBM Developer
SKILLS NETWORK**

Writing Your First Python Code

Estimated time needed: 25 minutes

Objectives

After completing this lab you will be able to:

- Write basic code in Python
- Work with various types of data in Python
- Convert the data from one type to another
- Use expressions and variables to perform operations

Table of Contents

- Say "Hello" to the world in Python
 - What version of Python are we using?
 - Writing comments in Python
 - Errors in Python
 - Does Python know about your error before it runs your code?
 - Exercise: Your First Program
- Types of objects in Python
 - Integers
 - Floats
 - Converting from one object type to a different object type
 - Boolean data type
 - Exercise: Types
- Expressions and Variables
 - Expressions
 - Exercise: Expressions
 - Variables
 - Exercise: Expression and Variables in Python

Estimated time needed: **25 min**

Say "Hello" to the world in Python

When learning a new programming language, it is customary to start with an "hello world" example. As simple as it is, this one line of code will ensure that we know how to print a string in output and how to execute code within cells in a notebook.

[Tip]: To execute the Python code in the code cell below, click on the cell to select it and press **Shift + Enter**.

```
[1]: # Try your first Python output  
  
print('Hello, Python!')  
  
Hello, Python!
```

After executing the cell above, you should see that Python prints `Hello, Python!`. Congratulations on running your first Python code!

[Tip]: `print()` is a function. You passed the string `'Hello, Python!'` as an argument to instruct Python on what to print.

What version of Python are we using?

There are two popular versions of the Python programming language in use today: Python 2 and Python 3. The Python community has decided to move on from Python 2 to Python 3, and many popular libraries have announced that they will no longer support Python 2.

Since Python 3 is the future, in this course we will be using it exclusively. How do we know that our notebook is executed by a Python 3 runtime? We can look in the top-right hand corner of this notebook and see "Python 3".

We can also ask Python directly and obtain a detailed answer. Try executing the following code:

```
[2]: # Check the Python Version  
  
import sys  
print(sys.version)
```

```
3.7.12 | packaged by conda-forge | (default, Oct 26 2021, 06:08:53)  
[GCC 9.4.0]
```

[Tip]: `sys` is a built-in module that contains many system-specific parameters and functions, including the Python version in use. Before using it, we must explicitly import it.

Writing comments in Python

In addition to writing code, note that it's always a good idea to add comments to your code. It will help others understand what you were trying to accomplish (the reason why you wrote a given snippet of code). Not only does this help **other people** understand your code, it can also serve as a reminder **to you** when you come back to it weeks or months later.

To write comments in Python, use the number symbol # before writing your comment. When you run your code, Python will ignore everything past the # on a given line

```
[3]: # Practice on writing comments

print('Hello, Python!')# This Line prints a string
# print('Hi')

Hello, Python!
```

After executing the cell above, you should notice that This line prints a string did not appear in the output, because it was a comment (and thus ignored by Python).

The second line was also not executed because print('Hi') was preceded by the number sign (#) as well! Since this isn't an explanatory comment from the programmer, but an actual line of code, we might say that the programmer *commented out* that second line of code.

Errors in Python

Everyone makes mistakes. For many types of mistakes, Python will tell you that you have made a mistake by giving you an error message. It is important to read error messages carefully to really understand where you made a mistake and how you may go about correcting it.

For example, if you spell print as frint, Python will display an error message. Give it a try:

```
[4]: # Print string as error message

frint("Hello, Python!")

-----
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_64/4017949674.py in <module>
      1 # Print string as error message
      2
----> 3 frint("Hello, Python!")

NameError: name 'frint' is not defined
```

The error message tells you:

1. where the error occurred (more useful in large notebook cells or scripts), and
2. what kind of error it was (NameError)

Here, Python attempted to run the function `frint`, but could not determine what `frint` is since it's not a built-in function and it has not been previously defined by us either.

You'll notice that if we make a different type of mistake, by forgetting to close the string, we'll obtain a different error (i.e., a `SyntaxError`). Try it below:

```
[5]: # Try to see built-in error message

print("Hello, Python!")

File "/tmp/ipykernel_64/3628399183.py", line 3
    print("Hello, Python!)
                  ^
SyntaxError: EOL while scanning string literal
```

Does Python know about your error before it runs your code?

Python is what is called an *interpreted language*. Compiled languages examine your entire program at compile time, and are able to warn you about a whole class of errors prior to execution. In contrast, Python interprets your script line by line as it executes it.

Python will stop executing the entire program when it encounters an error (unless the error is expected and handled by the programmer, a more advanced subject that we'll cover later on in this course).

Try to run the code in the cell below and see what happens:

```
[6]: # Print string and error to see the running order

print("This will be printed")
frint("This will cause an error")
print("This will NOT be printed")

This will be printed
-----
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_64/488948284.py in <module>
      2
      3     print("This will be printed")
----> 4     frint("This will cause an error")
      5     print("This will NOT be printed")

NameError: name 'frint' is not defined
```

Exercise: Your First Program

Generations of programmers have started their coding careers by simply printing "Hello, world!". You will be following in their footsteps.

In the code cell below, use the print() function to print out the phrase: Hello, world!

```
[7]: # Write your code below. Don't forget to press Shift+Enter to execute the cell
print("Hello, world!")

Hello, world!

▼ Click here for the solution
print("Hello, world!")
```

Now, let's enhance your code with a comment. In the code cell below, print out the phrase: Hello, world! and comment it with the phrase Print the traditional hello world all in one line of code.

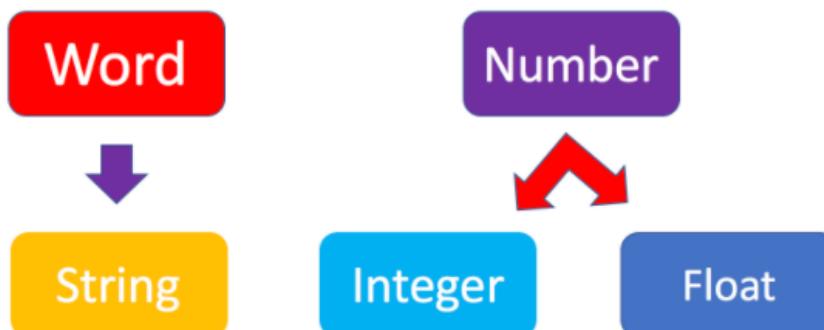
```
[10]: # Write your code below. Don't forget to press Shift+Enter to execute the cell
print("Hello, world!").# Print the traditional hello world

Hello, world!

▼ Click here for the solution
print("Hello, world!") # Print the traditional hello world
```

Types of objects in Python

Python is an object-oriented language. There are many different types of objects in Python. Let's start with the most common object types: *strings*, *integers* and *floats*. Anytime you write words (text) in Python, you're using *character strings* (strings for short). The most common numbers, on the other hand, are *integers* (e.g. -1, 0, 100) and *floats*, which represent real numbers (e.g. 3.14, -42.0).



The following code cells contain some examples.

```
[12]: # Integer
```

```
11
```

```
[12]: 11
```

```
[13]: # Float
```

```
2.14
```

```
[13]: 2.14
```

```
[14]: # String
```

```
"Hello, Python 101!"
```

```
[14]: 'Hello, Python 101!'
```

You can get Python to tell you the type of an expression by using the built-in `type()` function. You'll notice that Python refers to integers as `int`, floats as `float`, and character strings as `str`.

```
[16]: # Type of 12
```

```
| type(12)
```

```
[16]: int
```

```
[17]: # Type of 2.14
```

```
| type(2.14)
```

```
[17]: float
```

```
[18]: # Type of "Hello, Python 101!"
```

```
| type("Hello, Python 101!")
```

```
[18]: str
```

In the code cell below, use the `type()` function to check the object type of `12.0`.

```
[19]: # Write your code below. Don't forget to press Shift+Enter to execute the cell  
type(12.0)
```

```
[19]: float
```

```
<details><summary>Click here for the solution</summary>  
  
```python  
type(12.0)

```  
  
``</details>
```

Integers

Here are some examples of integers. Integers can be negative or positive numbers:

| | | | | | | | | |
|----|----|----|----|---|---|---|---|---|
| -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 |
|----|----|----|----|---|---|---|---|---|

We can verify this is the case by using, you guessed it, the `type()` function:

```
[20]: # Print the type of -1
```

```
type(-1)
```

```
[20]: int
```

```
[21]: # Print the type of 4
```

```
type(4)
```

```
[21]: int
```

```
[22]: # Print the type of 0
```

```
type(0)
```

```
[22]: int
```

Floats

Floats represent real numbers; they are a superset of integer numbers but also include "numbers with decimals". There are some limitations when it comes to machines representing real numbers, but floating point numbers are a good representation in most cases. You can learn more about the specifics of floats for your runtime environment, by checking the value of `sys.float_info`. This will also tell you what's the largest and smallest number that can be represented with them.

Once again, can test some examples with the `type()` function:

Once again, can test some examples with the `type()` function:

```
[23]: # Print the type of 1.0
type(1.0).# Notice that 1. is an int..and 1.0 is a float
[23]: float

[24]: # Print the type of 0.5
type(0.5)
[24]: float

[25]: # Print the type of 0.56
type(0.56)
[25]: float

[26]: # System settings about float type
sys.float_info
[26]: sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2, m=1)
```

Converting from one object type to a different object type

You can change the type of the object in Python; this is called typecasting. For example, you can convert an integer into a float (e.g. 2 to 2.0).

Let's try it:

```
[27]: # Verify that this is an integer
type(2)
[27]: int
```

Converting integers to floats

Let's cast integer 2 to float:

```
[29]: # Convert 2 to a float
float(2)
```

```
[29]: 2.0
```

```
[30]: # Convert integer 2 to a float and check its type
type(float(2))
```

```
[30]: float
```

When we convert an integer into a float, we don't really change the value (i.e., the significand) of the number. However, if we cast a float into an integer, we could potentially lose some

information. For example, if we cast the float 1.1 to integer we will get 1 and lose the decimal information (i.e., 0.1):

```
[31]: # Casting 1.1 to integer will result in loss of information  
int(1.1)
```

```
[31]: 1
```

Converting from strings to integers or floats

Sometimes, we can have a string that contains a number within it. If this is the case, we can cast that string that represents a number into an integer using int():

```
[32]: # Convert a string into an integer  
int('1')
```

```
[32]: 1
```

But if you try to do so with a string that is not a perfect match for a number, you'll get an error. Try the following:

```
[33]: # Convert a string into an integer with error  
int('1 or 2 people')  
  
-----  
ValueError                                                 Traceback (most recent call last)  
/tmp/ipykernel_64/3609426630.py in <module>  
      1 # Convert a string into an integer with error  
      2  
----> 3 int('1 or 2 people')  
  
ValueError: invalid literal for int() with base 10: '1 or 2 people'
```

You can also convert strings containing floating point numbers into float objects:

```
[34]: # Convert the string "1.2" into a float  
float('1.2')
```

```
[34]: 1.2
```

[Tip:] Note that strings can be represented with single quotes (`'1.2'`) or double quotes (`"1.2"`), but you can't mix both (e.g., `"1.2'`).

Converting numbers to strings

If we can convert strings to numbers, it is only natural to assume that we can convert numbers to strings, right?

```
[35]: # Convert an integer to a string  
str(1)
```

```
[35]: '1'
```

And there is no reason why we shouldn't be able to make floats into strings as well:

```
[36]: # Convert a float to a string  
str(1.2)
```

```
[36]: '1.2'
```

Boolean data type

Boolean is another important type in Python. An object of type *Boolean* can take on one of two values: True or False:

```
[37]: # Value true  
True
```

```
[37]: True
```

Notice that the value `True` has an uppercase "T". The same is true for `False` (i.e. you must use the uppercase "F").

```
[ ]: # Value false  
False
```

When you ask Python to display the type of a boolean object it will show `bool` which stands for boolean:

```
[38]: # Type of True  
type(True)
```

```
[38]: bool
```

```
[39]: # Type of False  
type(False)
```

We can cast boolean objects to other data types. If we cast a boolean with a value of True to an integer or float we will get a one. If we cast a boolean with a value of False to an integer or float we will get a zero. Similarly, if we cast a 1 to a Boolean, you get a True. And if we cast a 0 to a Boolean we will get a False. Let's give it a try:

```
[40]: # Convert True to int  
  
int(True)
```

```
[40]: 1
```

```
[43]: # Convert 1 to boolean  
  
bool(1)
```

```
[43]: True
```

```
[44]: # Convert 0 to boolean  
  
bool(0)
```

```
[44]: False
```

```
[45]: # Convert True to float  
  
float(True)
```

```
[45]: 1.0
```

Exercise: Types

What is the data type of the result of: `6 / 2` ?

```
[46]: # Write your code below. Don't forget to press Shift+Enter to execute the cell  
type(6/2) #float
```

```
[46]: float
```

▼ Click here for the solution
`type(6/2) # float`

What is the type of the result of: `6 // 2` ? (Note the double slash `//`.)

```
[47]: # Write your code below. Don't forget to press Shift+Enter to execute the cell  
type(6//2) # int, as the double slashes stand for integer division.
```

```
[47]: int
```

▼ Click here for the solution
`type(6//2) # int, as the double slashes stand for integer division`

Expression and Variables

Expressions

Expressions in Python can include operations among compatible types (e.g., integers and floats). For example, basic arithmetic operations like adding multiple numbers:

```
[48]: # Addition operation expression  
      43 + 60 + 16 + 41
```

```
[48]: 160
```

We can perform subtraction operations using the minus operator. In this case the result is a negative number:

```
[50]: # Subtraction operation expression  
      50 - 60
```

```
[50]: -10
```

We can do multiplication using an asterisk:

```
[51]: # Multiplication operation expression  
      5 * 5
```

```
[51]: 25
```

We can also perform division with the forward slash:

```
[52]: # Division operation expression  
      25 / 5
```

```
[52]: 5.0
```

```
[53]: # Division operation expression  
      25 / 6
```

```
[53]: 4.166666666666667
```

As seen in the quiz above, we can use the double slash for integer division, where the result is rounded down to the nearest integer:

```
[54]: # Integer division operation expression  
      25 // 5
```

```
[54]: 5
```

```
[55]: # Integer division operation expression  
      25 // 6
```

```
[55]: 4
```

Exercise: Expression

Let's write an expression that calculates how many hours there are in 160 minutes:

```
[56]: # Write your code below. Don't forget to press Shift+Enter to execute the cell  
160/60
```

```
[56]: 2.6666666666666665
```

▼ Click here for the solution

160/60

Or

160//60

Python follows well accepted mathematical conventions when evaluating mathematical expressions. In the following example, Python adds 30 to the result of the multiplication (i.e., 120).

Variables

Just like with most programming languages, we can store values in variables, so we can use them later on. For example:

```
[60]: # Store value into variable  
x = 43 + 60 + 16 + 41
```

To see the value of `x` in a Notebook, we can simply place it on the last line of a cell:

```
[61]: # Print out the value in variable  
x
```

```
[61]: 160
```

We can also perform operations on `x` and save the result to a new variable:

```
[63]: # Use another variable to store the result of the operation between variable and value  
y = x / 60  
y
```

```
[63]: 2.6666666666666665
```

If we save a value to an existing variable, the new value will overwrite the previous value:

```
[64]: # Overwrite variable with new value  
x = x / 60  
x
```

```
[64]: 2.6666666666666665
```

It's a good practice to use meaningful variable names, so you and others can read the code and understand it more easily:

```
[65]: # Name the variables meaningfully  
total_min = 43 + 42 + 57 # Total Length of albums in minutes  
total_min
```

```
[65]: 142
```

```
[66]: # Name the variables meaningfully  
total_hours = total_min / 60 # Total Length of albums in hours  
total_hours
```

```
[66]: 2.3666666666666667
```

If you'd rather have total hours as an integer, you can of course replace the floating point division with integer division (i.e., `//`).

Exercise: Expression and Variables in Python

What is the value of `x` where `x = 3 + 2 * 2`

```
[68]: # Write your code below. Don't forget to press Shift+Enter to execute the cell  
3 + 2 * 2
```

[68]: 7

► Click here for the solution

What is the value of `y` where `y = (3 + 2) * 2`?

```
[73]: # Write your code below. Don't forget to press Shift+Enter to execute the cell  
y = (3 + 2) * 2
```

▼ Click here for the solution

10

What is the value of `p z` where `z = x + y`?

```
[74]: # Write your code below. Don't forget to press Shift+Enter to execute the cell  
z = 7 + 10
```

▼ Click here for the solution

17

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

Practice Quiz: Expressions and Variables

Bookmark this page

Question 1

0/1 point (ungraded)

What is the result of the following operation: $11//2$

5

5.5



Answer

Incorrect: This is incorrect, the symbol `//` means integer division. Therefore you must round the result down.

Submit

You have used 1 of 2 attempts

Save

Incorrect (0/1 point)

Question 2

1/1 point (ungraded)

What is the value of `x` after the following is run:

`x=4`

`x=x/2`

4.0

2.0



Answer

Correct:

Correct: The value `x=x/2` changes the value of `x`, if `x` is assigned to its self. It's helpful to replace the value of `x` with its current value in this case 4 or $x=4/2$. We can also see that the result is a float.

Submit

You have used 1 of 2 attempts

Save

Correct (1/1 point)

Python String Operations

In Python, a string is a sequence of characters. A string is contained within two quotes.

You could also use single quotes. A string can be spaces or digits.

A string can also be special characters. We can bind or assign a string to another variable.

It is helpful to think of a string as an ordered sequence. Each element in the sequence can be

accessed using an index represented by the array of numbers.

The first index can be accessed as follows:

We can access index six. Moreover, we can access the 13th index.

We can also use negative indexing with strings. The last element is given by the index negative

one.

The first element can be obtained by index negative 15 and so on. We can bind a string to another variable. It is helpful to think of string as a list or tuple.

We can treat the string as a sequence and perform sequence operations.

We can also input a stride value as follows:

The two indicates we'd select every second variable.

We can also incorporate slicing. In this case, we return every second value up to index four.

We can use the `len` command to obtain the length of the string. As there are 15 elements, the

result is 15.

We can concatenate or combine strings. We use the addition symbols.

The result is a new string that is a combination of both. We can replicate values of a string.

We simply multiply the string by the number of times we would like to replicate it-in this case, three. The result is a new string.

The new string consists of three copies of the original string. This means you cannot change the value of the string, but you can create a new string.

For example, you can create a new string by setting it to the original variable and concatenate it

with a new string. The result is a new string that changes from

Michael Jackson to Michael Jackson is the best. Strings are immutable.

Back slashes represent the beginning of escape sequences. Escape sequences represent strings

that may be difficult to input.

For example, backslashes "n" represent a new line. The output is given by a new line after the

backslashes "n" is encountered. Similarly, backslash "t" represents a tab.

The output is given by a tab where the backslash, "t" is.

If you want to place a backslash in your string, use a double backslash.

The result is a backslash after the escape sequence. We can also place an "r" in front of the string. Now, let's take a look at string methods.

Strings are sequences and as such, have apply methods that work on lists and tuples.

Strings also have a second set of methods that just work on strings. When we apply a method to

the string A, we get a new string B that is different from A. Let's do some examples. Let's try with the method "Upper".

This method converts lowercase characters to uppercase characters. In this example, we set the

variable A to the following value. We apply the method "Upper", and set it equal to B.

The value for B is similar to A, but all the characters are uppercase.

The method replaces a segment of the string- i.e.

a substring with a new string. We input the part of the string we would like to change.

The second argument is what we would like to exchange the segment with.

The result is a new string with a segment changed.

The method find, find substrings. The argument is the substring you would like to find.

The output is the first index of the sequence. We can find the substring Jack.

If the substring is not in the string, the output is negative one.

STRINGS

- A sting is a sequence of characters contained within two quotes:

“Michael Jackson”

- You can also use single quotes :

‘Michael Jackson’

STRINGS

- A string can be spaces or digits

“1 2 3 4 5 6”

- A String can also be special characters :

‘@#2_#]&*%^%\$’

STRINGS

Name= “Michael Jackson”

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| M | i | c | h | a | e | I | J | a | c | k | s | o | n | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |



Python

String Operations

STRINGS

Name= "Michael Jackson"

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| M | i | c | h | a | e | I | J | a | c | k | s | o | n | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

Name[0]:M

Name[6] :I

Name[13]:o

STRINGS

Name= "Michael Jackson"

| | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|
| M | i | c | h | a | e | I | J | a | c | k | s | o | n | |
| -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

Name[-15] :M

Name[-1]: n

STRINGS: Slicing

Name= "Michael Jackson"

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| M | i | c | h | a | e | I | J | a | c | k | s | o | n | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

Name[0:4] =Mich

Name[8:12] =Jack

STRINGS: Stride

Name= "Michael Jackson"

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| M | i | c | h | a | e | I | J | a | c | k | s | o | n | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

Name[::2]: "McaJcsn"

STRINGS: Stride

Name= "Michael Jackson"

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| M | i | c | h | a | e | I | J | a | c | k | s | o | n | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

Name[::2]: "McaJcsn" Name[0:5]:"Mca"

TUPLES: Slicing

`len("Michael Jackson") =15`

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| M | i | c | h | a | e | I | J | a | c | k | s | o | n | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

String

Name= "Michael Jackson"

Statement = Name + " is the best"



Statement = "Michael Jackson is the best"

Tuples

3 * "Michael Jackson "



"Michael Jackson Michael Jackson Michael Jackson "

Strings: Immutable

Name= "Michael Jackson"

Name~~P~~= "J"

Name= Name+ " is the best"

Name: "Michael Jackson is the best"

Strings: escape sequences

- \ are meant to proceed escape sequences
- escape sequences are strings that are difficult to input

print(" Michael Jackson \n is the best")

Michael Jackson
is the best

"\n" means a new line

Strings: escape sequences

```
print(" Michael Jackson \t is the best" )
```

Michael Jackson is the best

“\t” means a new tab (big or long space)

Strings: escape sequences

```
print(" Michael Jackson \\ is the best" )
```

Michael Jackson \ is the best

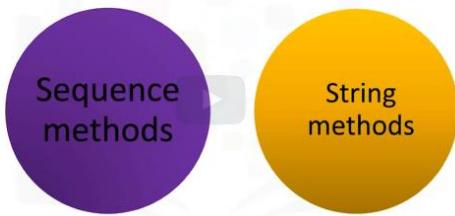
“\\” means a new back slash

Strings: escape sequences

```
print(r" Michael Jackson \ is the best" )
```

Michael Jackson \ is the best

String Methods



Method

A
↓

Method

↓
B

Method

A="Thriller is the sixth studio album"

A="Thriller is the sixth studio album"
B=A.upper()
B:"THRILLER IS THE SIXTH STUDIO ALBUM"

upper()

B:"THRILLER IS THE SIXTH STUDIO ALBUM"

Method

A='Michael Jackson is the best'

B=A.replace('Michael', 'Janet')

'Michael Jackson is the best'

Method

A='Michael Jackson is the best'
B=A.replace('Michael', 'Janet')
B: 'Janet Jackson is the best'



STRINGS: Stride

Name= "Michael Jackson"

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| M | i | c | h | a | e | I | J | a | c | k | s | o | n | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

Name.find('el'):5 Name.find('Jack'):8

STRINGS: Stride

Name= "Michael Jackson"

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| M | i | c | h | a | e | I | J | a | c | k | s | o | n | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

Name.find('el'):5 Name.find('Jack'):8
Name.find('&*D'):-1

If it cannot find, it will yield, -1



String Operations

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Work with Strings
- Perform operations on String
- Manipulate Strings using indexing and escape sequences

Table of Contents

- What are Strings?
- Indexing
 - Negative Indexing
 - Slicing
 - Stride
 - Concatenate Strings
- Escape Sequences
- String Operations
- Quiz on Strings

What are Strings?

The following example shows a string contained within 2 quotation marks:

```
[1]: # Use quotation marks for defining string  
      "Michael Jackson"
```

```
[1]: 'Michael Jackson'
```

We can also use single quotation marks:

```
[2]: # Use single quotation marks for defining string  
      'Michael Jackson'
```

```
[2]: 'Michael Jackson'
```

A string can be a combination of spaces and digits:

```
[3]: # Digits and spaces in string  
      '1 2 3 4 5 6 '
```

```
[3]: '1 2 3 4 5 6 '
```

A string can also be a combination of special characters :

```
[4]: # Special characters in string  
      '@#2_#]&*^%$'
```

```
[4]: '@#2_#]&*^%$'
```

We can print our string using the print statement:

```
[5]: # Print the string  
  
print("hello!")  
  
hello!
```

We can bind or assign a string to another variable:

```
[6]: # Assign string to variable  
  
name = "Michael Jackson"  
name  
  
[6]: 'Michael Jackson'
```

Indexing

It is helpful to think of a string as an ordered sequence. Each element in the sequence can be accessed using an index represented by the array of numbers:

Name= “Michael Jackson”

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| M | i | c | h | a | e | l | | J | a | c | k | s | o | n |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

The first index can be accessed as follows:

[Tip]: Because indexing starts at 0, it means the first index is on the index 0.

```
[8]: # Print the first element in the string  
  
print(name[0])  
M
```

We can access index 6:

```
[9]: # Print the element on index 6 in the string  
  
print(name[6])  
l
```

Moreover, we can access the 13th index:

```
[10]: # Print the element on the 13th index in the string  
  
print(name[13])  
o
```

Negative Indexing

We can also use negative indexing with strings:

Name= "Michael Jackson"

| | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|
| M | i | c | h | a | e | l | | J | a | c | k | s | o | n |
| -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

Negative index can help us to count the element from the end of the string.

The last element is given by the index -1:

```
[11]: # Print the last element in the string  
print(name[-1])  
n
```

The first element can be obtained by index -15:

```
[12]: # Print the first element in the string  
print(name[-15])  
  
M
```

We can find the number of characters in a string by using `len`, short for length:

```
[13]: # Find the Length of string  
len("Michael Jackson")  
  
[13]: 15
```

Slicing

We can obtain multiple characters from a string using slicing, we can obtain the 0 to 4th and 8th to the 12th element:

Name= "Michael Jackson"

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|--|---|---|---|---|---|---|---|
| M | i | c | h | a | e | l | | J | a | c | k | s | o | n |
| | | | | | | | | | | | | | | |

Name[0:4] =Mich

Name[8:12] =Jack

[Tip]: When taking the slice, the first number means the index (start at 0), and the second number means the length from the index to the last element you want (start at 1)

```
[14]: # Take the slice on variable name with only index 0 to index 3  
name[0:4]
```

```
[14]: 'Mich'
```

```
[15]: # Take the slice on variable name with only index 8 to index 11  
name[8:12]
```

```
[15]: 'Jack'
```

Stride

We can also input a stride value as follows, with the '2' indicating that we are selecting every second variable:



Name[::2]: "McalJcsn"

```
[16]: # Get every second element. The elements on index 1, 3, 5 ...  
name[::2]
```

```
[16]: 'McalJcsn'
```

We can also incorporate slicing with the stride. In this case, we select the first five elements and then use the stride:

```
[17]: # Get every second element in the range from index 0 to index 4  
name[0:5:2]
```

```
[17]: 'Mca'
```

Concatenate Strings

We can concatenate or combine strings by using the addition symbols, and the result is a new string that is a combination of both:

```
[18]: # Concatenate two strings
statement = name + "is the best"
statement
```

```
[18]: 'Michael Jacksonis the best'
```

To replicate values of a string we simply multiply the string by the number of times we would like to replicate it. In this case, the number is three. The result is a new string, and this new string consists of three copies of the original string:

```
[19]: # Print the string for 3 times
3 * "Michael Jackson"
```

```
[19]: 'Michael JacksonMichael JacksonMichael Jackson'
```

You can create a new string by setting it to the original variable. Concatenated with a new string, the result is a new string that changes from Michael Jackson to "Michael Jackson is the best".

```
[20]: # Concatenate strings
name = "Michael Jackson"
name = name + " is the best"
name
```

```
[20]: 'Michael Jackson is the best'
```

Escape Sequences

Back slashes represent the beginning of escape sequences. Escape sequences represent strings that may be difficult to input. For example, back slash "n" represents a new line. The output is given by a new line after the back slash "n" is encountered

```
[21]: # New Line escape sequence  
  
print(" Michael Jackson \n is the best")  
  
Michael Jackson  
is the best
```

Similarly, back slash "t" represents a tab:

```
[22]: # Tab escape sequence  
  
print(" Michael Jackson \t is the best")  
  
Michael Jackson      is the best
```

If you want to place a back slash in your string, use a double back slash:

```
[23]: # Include back slash in string  
  
print(" Michael Jackson \\ is the best")  
  
Michael Jackson \ is the best
```

We can also place an "r" before the string to display the backslash:

```
[24]: # r will tell python that string will be display as raw string  
  
print(r" Michael Jackson \ is the best")  
  
Michael Jackson \ is the best
```

String Operations

There are many string operation methods in Python that can be used to manipulate the data. We are going to use some basic string operations on the data.

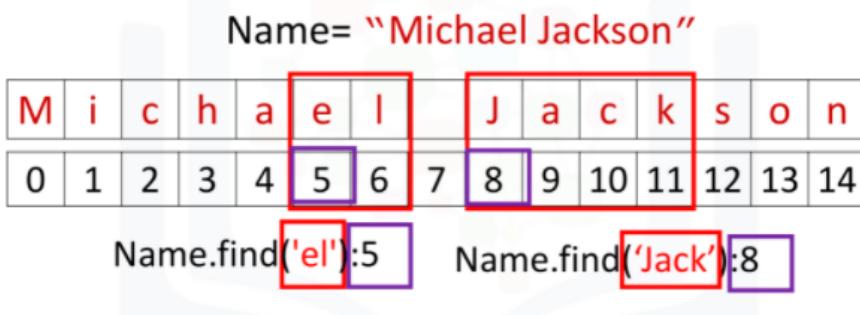
Let's try with the method upper; this method converts lower case characters to upper case characters:

```
[25]: # Convert all the characters in string to upper case  
  
a = "Thriller is the sixth studio album"  
print("before upper:", a)  
b = a.upper()  
print("After upper:", b)  
  
before upper: Thriller is the sixth studio album  
After upper: THRILLER IS THE SIXTH STUDIO ALBUM
```

The method replaces a segment of the string, i.e. a substring with a new string. We input the part of the string we would like to change. The second argument is what we would like to exchange the segment with, and the result is a new string with the segment changed:

```
[26]: # Replace the old substring with the new target substring is the segment has been found in the string  
  
a = "Michael Jackson is the best"  
b = a.replace('Michael', 'Janet')  
b  
  
[26]: 'Janet Jackson is the best'
```

The method `find` finds a sub-string. The argument is the substring you would like to find, and the output is the first index of the sequence. We can find the sub-string `jack` or `el`.



```
[27]: # Find the substring in the string. Only the index of the first element of substring in string will be the output  
  
name = "Michael Jackson"  
name.find('el')
```

```
[27]: 5
```

```
[28]: # Find the substring in the string.  
  
name.find('Jack')
```

```
[28]: 8
```

If the sub-string is not in the string then the output is a negative one. For example, the string 'Jasdfdasdasdf' is not a substring:

```
[29]: # If cannot find the substring in the string  
  
name.find('Jasdfdasdasdf')
```

```
[29]: -1
```

Quiz on Strings

What is the value of the variable `a` after the following code is executed?

```
[34]: # Write your code below and press Shift+Enter to execute..  
a = "1"  
a
```

```
[34]: '1'
```

[▼ Click here for the solution](#)

"1"

What is the value of the variable `b` after the following code is executed?

```
[35]: # Write your code below and press Shift+Enter to execute  
b = "2"  
b
```

```
[35]: '2'
```

[▼ Click here for the solution](#)

"2"

What is the value of the variable `c` after the following code is executed?

```
[36]: # Write your code below and press Shift+Enter to execute..  
c = a + b  
c
```

```
[36]: '12'
```

▼ Click here for the solution

"12"

Consider the variable `d` use slicing to print out the first three elements:

```
[37]: # Write your code below and press Shift+Enter to execute  
d = "ABCDEFG"  
print_(d[:3])
```

```
ABC
```

▼ Click here for the solution

`print(d[:3])`

or

`print(d[0:3])`

Use a stride value of 2 to print out every second character of the string `e`

```
[39]: # Write your code below and press Shift+Enter to execute  
  
e = 'clocrkr1e1c1t'  
print(e[::2])
```

correct

▼ Click here for the solution

```
print(e[::2])
```

Print out a backslash:

```
[40]: # Write your code below and press Shift+Enter to execute  
print("\\")
```

\

▼ Click here for the solution

```
print("\\\\")
```

or

```
print(r"\ ")
```

Convert the variable `f` to uppercase:

```
[41]: # Write your code below and press Shift+Enter to execute  
f = "You are wrong"  
f.upper()
```

```
[41]: 'YOU ARE WRONG'
```

▼ Click here for the solution

```
f.upper()
```

Consider the variable `g`, and find the first index of the sub-string `snow`:

```
[42]: # Write your code below and press Shift+Enter to execute  
g = "Mary had a little lamb Little lamb, little lamb Mary had a little lamb \  
Its fleece was white as snow And everywhere that Mary went Mary went, Mary went_\  
Everywhere that Mary went The lamb was sure to go"  
g.find("snow")
```

```
[42]: 95
```

▼ Click here for the solution

```
g.find("snow")
```

In the variable `g`, replace the sub-string `Mary` with `Bob`:

```
[43]: # Write your code below and press Shift+Enter to execute  
g.replace_("Mary", "Bob")
```

```
[43]: 'Bob had a little lamb Little lamb, little lamb Bob had a little lamb Its fleece was white as snow And everywhere that Bob went Bob went, Bob went Everywhere that Bob went The lamb was sure to go'
```

▼ Click here for the solution
`g.replace_("Mary", "Bob")`

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

S

E

Module Introduction and Learning Objectives



This module begins a journey into Python data structures by explaining the use of lists and tuples and how they are able to store collections of data in a single variable.

Next, learn about dictionaries and how they function by storing data in pairs of keys and values, and end with Python sets to learn how this type of collection can appear in any order and will only contain unique elements.

Learning Objectives

In this lesson you will:

- Understand tuples and lists by describing and manipulating tuple combinations and list data structures.
- Demonstrate understanding of dictionaries by writing structures with correct keys and values.
- Understand the differences between sets, tuples, and lists by creating sets.

List and Tuples

In this video we will cover lists and tuples.

These are called compound data types and are one of the key types of data structures in Python. Tuples.

Tuples are an ordered sequence.

- Here is a tuple ratings.
- Tuples are expressed as comma separated elements within parentheses.
- These are values inside the parentheses.

In Python, there are different types:

strings, integer, float. They can all be contained

in a tuple but the type of the variable is tuple.

Each element of a tuple can be accessed via an index.

The following table represents the relationship between the index and the elements in the tuple. The first element can be accessed by the name of the tuple followed by a square bracket with the index number,

in this case zero. We can access the second element as follows.

We can also access the last element. In Python, we can use negative index.

The relationship is as follows. The corresponding values are shown here.

We can concatenate or combine tuples by adding them.

The result is the following with the following index.

If we would like multiple elements from a tuple, we could also slice tuples. For example, if we want the first three elements we use the following command. The last index is one larger than the index you want; similarly if we want the last two elements, we use the following command. Notice, how the last index is one larger than the length of the tuple. We can use the len command to obtain the length of a tuple. As there are five elements, the result is 5.

Tuples are immutable which means we can't change them.

To see why this is important, let's see what happens when we set the variable ratings 1 to ratings. Let's use the image to provide a simplified explanation of what's going on.

Each variable does not contain a tuple, but references the same immutable tuple object. See the objects and classes module for more about objects. Let's say, we want to change the element at index 2. Because tuples are immutable we can't, therefore ratings 1 will not be affected by a change in rating because the tuple is immutable, i.e we can't change it.

We can assign a different tuple to the ratings variable.

The variable ratings now references another tuple.

As a consequence of immutability, if we would like to manipulate a tuple we must create a new tuple instead. For example, if we would like to sort a tuple we use the function sorted.

The input is the original tuple, the output is a new sorted list. For more on functions, see our video on functions.

A tuple can contain other tuples as well as other complex data types.

This is called nesting. We can access these elements using the standard indexing methods. If we select an index with a tuple, the same index convention applies. As such, we can then access values in the tuple. For example, we could access the second element. We can apply this indexing directly to the tuple variable NT. It is helpful to visualize this as a tree. We can visualize this nesting as a tree.

The tuple has the following indexes. If we consider indexes with other tuples, we see the tuple at index 2 contains a tuple with two elements. We can access those two indexes. The same convention applies to index 3. We can access the elements in those tuples as well. We can continue the process. We can even access deeper levels of the tree by adding another square bracket.

We can access different characters in the string or various elements in the second tuple contained in the first. Lists are also a popular data structure in Python. Lists are also an ordered sequence.

Here is a list, "L." A list is represented with square brackets. In many respects, lists are like tuples. One key difference is they are mutable. Lists can contain strings, floats, integers.

We can nest other lists. We also nest tuples and other data structures.

The same indexing conventions apply for nesting. Like tuples, each element of a list can be accessed via an index.

The following table represents the relationship between the index and the elements in the list. The first element can be accessed by the name of the list followed by a square bracket with the index number, in this case zero. We can access the second element as follows.

We can also access the last element. In Python, we can use a negative index; the relationship is as follows. The corresponding indexes are as follows. We can also perform slicing in lists. For example, if we want the last two elements in this list we use the following command.

Notice how the last index is one larger than the length of the list.

The index conventions for lists and tuples are identical.

Check the labs for more examples. We can concatenate or combine lists by adding them. The result is the following. The new list

has the following indices. Lists are mutable, therefore we can change them. For example, we apply the method extends by adding a dot followed by the name of the method then parentheses. The argument inside the parentheses is a new list that we are going to concatenate

to the original list. In this case, instead of creating a new list, "L1," the original list, "L," is modified by adding two new elements.

To learn more about methods check out our video on objects and classes.

Another similar method is append. If we apply append instead of extended, we add one element to the list. If we look at the index there is only one more element. Index 3 contains the list we appended.

Every time we apply a method, the list changes.

If we apply extend, we add two new elements to the list.

The list L is modified by adding two new elements.

If we append the string A, we further change the list, adding the string A. As lists are mutable we can change them.

For example, we can change the first element as follows.

The list now becomes hard rock 10 1.2. We can delete an element of a list using the del command. We simply indicate the list item we would like to remove as an argument. For example, if we would like to remove the first element the result becomes 10 1.2.

We can delete the second element. This operation removes the second element off the list. We can convert a string to a list using split. For example, the method split converts every group of characters separated by a space into an element of a list. We can use the split function to separate strings on a

specific character known, as a delimiter. We simply pass the delimiter we would like to split on as an argument, in this case a comma. The result is a list. Each element corresponds to a set of characters that have been separated by a comma. When we set one variable B equal to A, both A and B are referencing the same list.

Multiple names referring to the same object is known as aliasing.

We know from the list slide that the first element in B is set as hard rock.

If we change the first element in A to banana,
we get a side effect, the value of B will change as a consequence.
A and B are referencing the same list, therefore if we change A,
list B also changes. If we check the first element of B
after changing list A, we get banana instead of hard rock.

You can clone list A by using the following syntax.

Variable A references one list. Variable B
references a new copy or clone of the original list.

Now if you change A, B will not change. We can get more info on lists, tuples, and many other objects in Python using the help command. Simply pass in the list, tuple, or any other Python object.

See the labs for more things, you can do with lists.



Python

List and Tuples

Tuples

- Tuples are an ordered sequence
- Here is a Tuple “Ratings”
- Tuples are written as comma-separated elements within parentheses



Ratings = (10, 9, 6, 5, 10, 8, 9, 6, 2)

Tuples



tuple1 = ('disco', 10, 1.2)

type (tuple1)=tuple

Tuples

`Tuple1 =("disco", 10, 1.2)`

| | | |
|---|---------|---------------------------------|
| 0 | "disco" | <code>Tuple1[0]: "disco"</code> |
| 1 | 10 | <code>Tuple1[1]: 10</code> |
| 2 | 1.2 | <code>Tuple1[2]: 1.2</code> |

Tuples

`Tuple1 =("disco", 10, 1.2)`

| | | | |
|----|---|---------|----------------------------------|
| -3 | 0 | "disco" | <code>Tuple1[-3]: "disco"</code> |
| -2 | 1 | 10 | <code>Tuple1[-2]: 10</code> |
| -1 | 2 | 1.2 | <code>Tuple1[-1]: 1.2</code> |

Tuples

`("disco", 10, 1.2)`



`tuple2 = tuple1 + ("hard rock", 10)`

`("disco", 10, 1.2, "hard rock", 10)`

Tuples

(“disco”, 10, 1.2)



tuple2 = tuple1 + (“hard rock”, 10)

(“disco”, 10, 1.2, “hard rock”, 10)

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|

Tuples: Slicing

(“disco”, 10, 1.2, “hard rock”, 10)

| | | | | | |
|---|---|---|--|---|---|
| 0 | 1 | 2 | | 3 | 4 |
|---|---|---|--|---|---|

tuple2[0:3] : ('disco', 10, 1.2)

*Last index is 1 larger (so “2” becomes “3”)

Tuples: Slicing

```
len(("disco", 10, 1.2, "hard rock", 10))
```

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
| 1 | 2 | 3 | 4 | 5 |

Tuples: Immutable

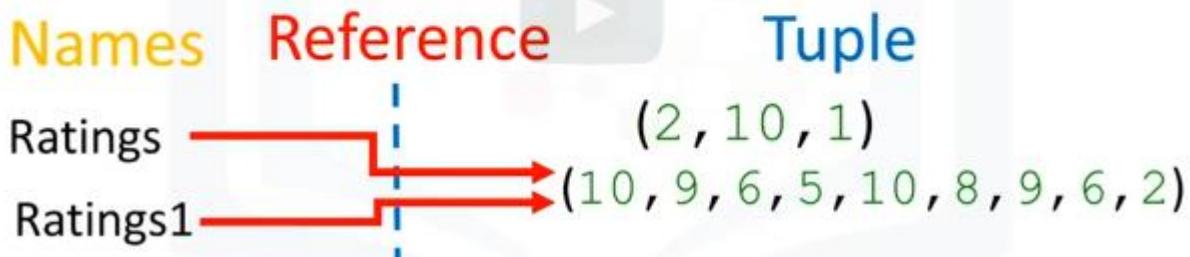
Ratings = (10, 9, 6, 5, 10, 8, 9, 6, 2)

Ratings1=Ratings



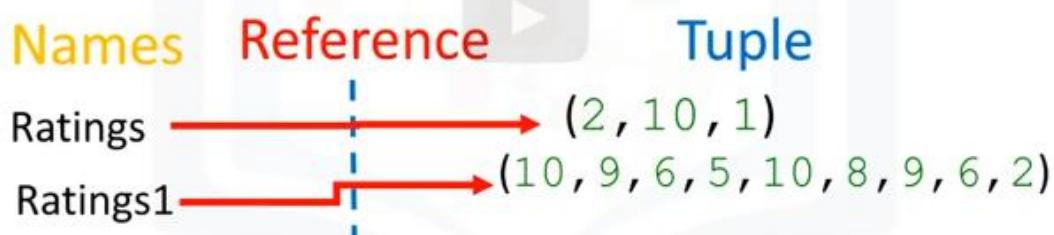
Tuples: Immutable

Ratings = (2, 10, 1)



Tuples: Immutable

Ratings = (2, 10, 1)



Tuples: Immutable

Ratings = (10, 9, 6, 5, 10, 8, 9, 6, 2)

RatingsSorted = sorted(Ratings)

[2, 5, 6, 6, 8, 9, 9, 10, 10]

Tuples: Nesting

NT = (1, 2, ("pop", "rock"), (3,4), ("disco", (1,2)))



NT[2]: ("pop", "rock") [1] = "rock" → NT[2][1] = "rock"



NT = (1, 2, ("pop", "rock"), (3,4), ("disco", (1,2)))



NT[2]

("pop", "rock")



NT[2][0]

NT[2][1]

NT[3]

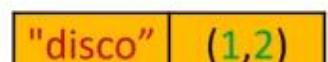
(3,4)



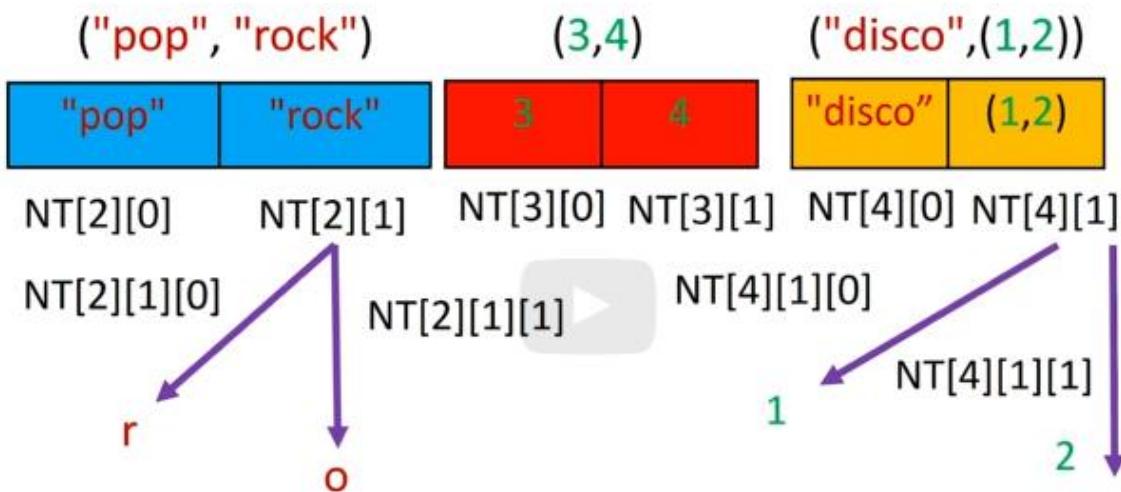
NT[3][0] NT[3][1]

NT[4]

("disco", (1,2))



NT[4][0] NT[4][1]



Lists

- Lists are also ordered sequences
- Here is a List “L”
- A List is represented with square brackets
- List **mutable**

L = [Michael Jackson, 10.1, 1982]

Lists

[Michael Jackson, 10.1, 1982, [1, 2], ('A', 1)]

Lists

```
L =["Michael Jackson", 10.1, 1982]
```

| | | |
|---|-------------------|-------------------------|
| 0 | "Michael Jackson" | L[0]: "Michael Jackson" |
| 1 | 10.1 | L[1]: 10.1 |
| 2 | 1982 | L[2]: 1982 |

Lists

```
L =["Michael Jackson", 10.1, 1982]
```

| | | | |
|----|---|-------------------|--------------------------|
| -3 | 0 | "Michael Jackson" | L[-3]: "Michael Jackson" |
| -2 | 1 | 10.1 | L[-2]: 10.1 |
| -1 | 2 | 1982 | L[-1]: 1982 |

Lists: Slicing

```
L =["Michael Jackson", 10.1, 1982, "MJ", 1]
```

```
0 1 2 3 4
```

```
L[3:5]:["MJ", 1]
```

Lists

```
L=[“Michael Jackson”, 10.1, 1982]
```

```
L1 = L+["pop", 10]
```

```
L1=[“Michael Jackson”, 10.1, 1982, “pop”, 10]
```

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|

Lists

```
L=[“Michael Jackson”, 10.1, 1982]
```

```
L.extend(["pop", 10])
```

```
L=[“Michael Jackson”, 10.1, 1982, “pop”, 10]
```

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|

Lists

```
L=[“Michael Jackson”, 10.1, 1982]
```

```
L.append ([“pop”, 10])
```

```
L=[“Michael Jackson”, 10.1, 1982, [“pop”, 10]]
```

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
|---|---|---|---|

Lists

```
L=[“Michael Jackson”, 10.1, 1982]
```

```
L.extend([“pop”, 10])
```

```
[“Michael Jackson”, 10.1, 1982, “pop”, 10]
```

```
L.append (“A”)
```

```
[“Michael Jackson”, 10.1, 1982, “pop”, 10, “A”]
```

Lists

```
A=[“disco”, 10, 1.2]
```

```
A[0]=“hard rock”
```

```
A=[“hard rock”, 10, 1.2]
```

Lists

```
A=["hard rock", 10, 1.2]
```

↑
del(A[0])



```
A:[10, 1.2]
```

Lists

```
A=["hard rock", 10, 1.2]
```

↑
del(A[1])



```
A: ["hard rock", 1.2]
```

Convert String to List

```
"hard rock".split()
```

```
["hard", "rock"]
```

Lists

"A,B,C,D".split(",")

[“A”, “B”, “C”, “D”]

Lists: Aliasing

A=[“hard rock”, 10, 1.2]

B=A



Lists: Aliasing

B[0]=“hard rock”

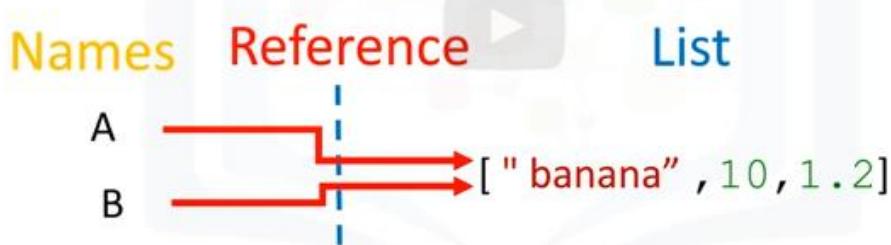
A[0]=“banana”



Lists: Aliasing

B[0]= "hard rock"

A[0]= " banana "



Lists: Aliasing

B[0]= "hard rock"

A[0]= " banana "



B[0]: " banana "



Lists: Clone

A=["hard rock", 10, 1.2]

B=A[:]





Lists in Python

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Perform list operations in Python, including indexing, list manipulation, and copy/clone list.

Table of Contents

- [About the Dataset](#)
 - [Lists](#)
 - [Indexing](#)
 - [List Content](#)
 - [List Operations](#)
 - [Copy and Clone List](#)
 - [Quiz on Lists](#)
-

About the Dataset

Imagine you received album recommendations from your friends and compiled all of the recommendations into a table, with specific information about each album.

The table has one row for each movie and several columns:

- **artist** - Name of the artist
- **album** - Name of the album
- **released_year** - Year the album was released
- **length_min_sec** - Length of the album (hours,minutes,seconds)
- **genre** - Genre of the album
- **music_recording_sales_millions** - Music recording sales (millions in USD) on [SONG://DATABASE](#)
- **claimed_sales_millions** - Album's claimed sales (millions in USD) on [SONG://DATABASE](#)
- **date_released** - Date on which the album was released
- **soundtrack** - Indicates if the album is the movie soundtrack (Y) or (N)
- **rating_of_friends** - Indicates the rating from your friends from 1 to 10

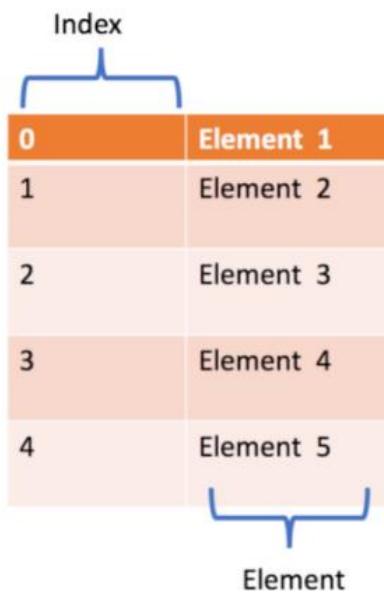
The dataset can be seen below:

| Artist | Album | Released | Length | Genre | Music recording sales (millions) | Claimed sales (millions) | Released | Soundtrack | Rating (friends) |
|-----------------|---------------------------------|----------|----------|-----------------------------|----------------------------------|--------------------------|-----------|------------|------------------|
| Michael Jackson | Thriller | 1982 | 00:42:19 | Pop, rock, R&B | 46 | 65 | 30-Nov-82 | | 10.0 |
| AC/DC | Back in Black | 1980 | 00:42:11 | Hard rock | 26.1 | 50 | 25-Jul-80 | | 8.5 |
| Pink Floyd | The Dark Side of the Moon | 1973 | 00:42:49 | Progressive rock | 24.2 | 45 | 01-Mar-73 | | 9.5 |
| Whitney Houston | The Bodyguard | 1992 | 00:57:44 | Soundtrack/R&B, soul, pop | 26.1 | 50 | 25-Jul-80 | Y | 7.0 |
| Meat Loaf | Bat Out of Hell | 1977 | 00:46:33 | Hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | | 7.0 |
| Eagles | Their Greatest Hits (1971-1975) | 1976 | 00:43:08 | Rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | | 9.5 |
| Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | Disco | 20.6 | 40 | 15-Nov-77 | Y | 9.0 |
| Fleetwood Mac | Rumours | 1977 | 00:40:01 | Soft rock | 27.9 | 40 | 04-Feb-77 | | 9.5 |

Lists

Indexing

We are going to take a look at lists in Python. A list is a sequenced collection of different objects such as integers, strings, and even other lists as well. The address of each element within a list is called an **index**. An index is used to access and refer to items within a list.



[Element 1 , Element 2 , Element 3 , Element 4, Element 5]

Index 0 1 2 3 4

To create a list, type the list within square brackets [], with your content inside the parenthesis and separated by commas. Let's try it!

```
[4]: # Create a list
L = ["Michael Jackson", 10.1, 1982]
L = ["Michael Jackson", 10.1, 1982]
```

We can use negative and regular indexing with a list:

```
L =["Michael Jackson", 10.1, 1982]
```

| | | | |
|----|---|-------------------|--------------------------|
| -3 | 0 | "Michael Jackson" | L[-3]: "Michael Jackson" |
| -2 | 1 | 10.1 | L[-2]: 10.1 |
| -1 | 2 | 1982 | L[-1]: 1982 |

```
[6]: # Print the elements on each index

print('the same element using negative and positive indexing:\n Postive:',L[0],'\n Negative:',L[-3])
print('the same element using negative and positive indexing:\n Postive:',L[1],'\n Negative:',L[-2])
print('the same element using negative and positive indexing:\n Postive:',L[2],'\n Negative:',L[-1])

the same element using negative and positive indexing:
Postive: Michael Jackson
Negative: Michael Jackson
the same element using negative and positive indexing:
Postive: 10.1
Negative: 10.1
the same element using negative and positive indexing:
Postive: 1982
Negative: 1982
```

List Content

Lists can contain strings, floats, and integers. We can nest other lists, and we can also nest tuples and other data structures. The same indexing conventions apply for nesting:

```
[29]: # Sample List

["Michael Jackson", 10.1, 1982, [1, 2], ("A", 1)]
```



```
[29]: ['Michael Jackson', 10.1, 1982, [1, 2], ('A', 1)]
```

List Operations

We can also perform slicing in lists. For example, if we want the last two elements, we use the following command:

```
[7]: # Sample List  
L = ["Michael Jackson", 10.1, 1982, "MJ", 1]  
L = ["Michael Jackson", 10.1, 1982, "MJ", 1]
```

L=[**"Michael Jackson"**, **10.1**, **1982**, **"MJ"**, **1**]

| | | | | |
|----------|----------|----------|----------|----------|
| 0 | 1 | 2 | 3 | 4 |
|----------|----------|----------|----------|----------|

```
[8]: # List slicing  
L[3:5]
```

```
[8]: ['MJ', 1]
```

We can use the method `extend` to add new elements to the list:

```
[10]: # Use extend to add elements to list  
L = ["Michael Jackson", 10.2]  
L.extend(['pop', 10])
```

Another similar method is `append`. If we apply `append` instead of `extend`, we add one element to the list:

```
[11]: # Use append to add elements to list  
L = ["Michael Jackson", 10.2]  
L.append(['pop', 10])  
L
```

```
[11]: ['Michael Jackson', 10.2, ['pop', 10]]
```

Each time we apply a method, the list changes. If we apply `extend` we add two new elements to the list. The list L is then modified by adding two new elements:

```
[13]: # Use extend to add elements to list  
  
L = ["Michael Jackson", 10.2]  
L.extend(['pop', 10])  
L
```

```
[13]: ['Michael Jackson', 10.2, 'pop', 10]
```

If we append the list `['a', 'b']` we have one new element consisting of a nested list:

```
[14]: # Use append to add elements to list  
  
L.append(['a', 'b'])  
L
```

```
[14]: ['Michael Jackson', 10.2, 'pop', 10, ['a', 'b']]
```

As lists are mutable, we can change them. For example, we can change the first element as follows:

```
[15]: # Change the element based on the index  
  
A = ["disco", 10, 1.2]  
print('Before change:', A)  
A[0] = 'hard rock'  
print('After change:', A)
```

```
Before change: ['disco', 10, 1.2]  
After change: ['hard rock', 10, 1.2]
```

We can also delete an element of a list using the `del` command:

```
[16]: # Delete the element based on the index  
  
print('Before change:', A)  
del(A[0])  
print('After change:', A)
```

```
Before change: ['hard rock', 10, 1.2]  
After change: [10, 1.2]
```

We can convert a string to a list using `split`. For example, the method `split` translates every group of characters separated by a space into an element in a list:

```
[17]: # Split the string, default is by space  
  
'hard rock'.split()
```

```
[17]: ['hard', 'rock']
```

We can use the split function to separate strings on a specific character which we call a **delimiter**. We pass the character we would like to split on into the argument, which in this case is a comma. The result is a list, and each element corresponds to a set of characters that have been separated by a comma:

```
[18]: # Split the string by comma  
  
'A,B,C,D'.split(',')
```

```
[18]: ['A', 'B', 'C', 'D']
```

Copy and Clone List

When we set one variable B equal to A, both A and B are referencing the same list in memory:

```
[19]: # Copy (copy by reference) the list A  
  
A = ["hard rock", 10, 1.2]  
B = A  
print('A:', A)  
print('B:', B)
```



```
A: ['hard rock', 10, 1.2]  
B: ['hard rock', 10, 1.2]
```



Initially, the value of the first element in **B** is set as "hard rock". If we change the first element in **A** to "**banana**", we get an unexpected side effect. As **A** and **B** are referencing the same list, if we change list **A**, then list **B** also changes. If we check the first element of **B** we get "banana" instead of "hard rock":

```
[20]: # Examine the copy by reference
print('B[0]:', B[0])
A[0] = "banana"
print('B[0]:', B[0])
```

B[0]: hard rock
B[0]: banana

This is demonstrated in the following figure:

B[0]: "hard rock" → B[0]: "banana"
A[0]= " banana "



You can clone list **A** by using the following syntax:

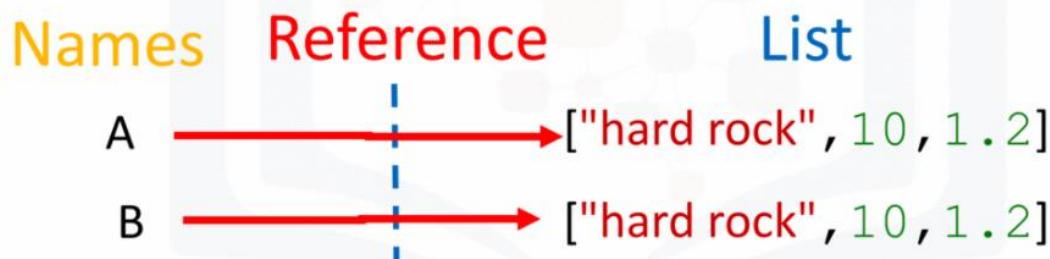
```
[21]: # Clone (clone by value) the list A
B = A[:]
B
```

[21]: ['banana', 10, 1.2]

Variable **B** references a new copy or clone of the original list. This is demonstrated in the following figure:

A=["hard rock", 10, 1.2]

B=A[:]



Now if you change A, B will not change:

```
[22]: print('B[0]:', B[0])
A[0] = "hard rock"
print('B[0]:', B[0])
```

B[0]: banana
B[0]: banana

Quiz on List

Create a list `a_list`, with the following elements `1`, `hello`, `[1,2,3]` and `True`.

```
[25]: # Write your code below and press Shift+Enter to execute
a_list = [1, "hello", [1,2,3], True]
a_list
```

```
[25]: [1, 'hello', [1, 2, 3], True]
```

▼ Click here for the solution

```
a_list = [1, 'hello', [1, 2, 3] , True]
a_list
```

Find the value stored at index 1 of `a_list`.

```
[26]: # Write your code below and press Shift+Enter to execute
a_list[1]
```

```
[26]: 'hello'
```

▼ Click here for the solution

```
a_list[1]
```

Retrieve the elements stored at index 1, 2 and 3 of `a_list`.

```
[27]: # Write your code below and press Shift+Enter to execute
a_list[1:4]
```

```
[27]: ['hello', [1, 2, 3], True]
```

▼ Click here for the solution

```
a_list[1:4]
```

Concatenate the following lists `A = [1, 'a']` and `B = [2, 1, 'd']`:

```
[28]: # Write your code below and press Shift+Enter to execute
A = [1, 'a'].
B = [2, 1, 'd']
A + B
```

```
[28]: [1, 'a', 2, 1, 'd']
```

▼ Click here for the solution

```
A = [1, 'a']
B = [2, 1, 'd']
A + B
```

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.



Tuples in Python

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Perform the basics tuple operations in Python, including indexing, slicing and sorting

Table of Contents

- About the Dataset
- Tuples
 - Indexing
 - Slicing
 - Sorting
- Quiz on Tuples

About the Dataset

Imagine you received album recommendations from your friends and compiled all of the recommendations into a table, with specific information about each album.

The table has one row for each movie and several columns:

- **artist** - Name of the artist
- **album** - Name of the album
- **released_year** - Year the album was released
- **length_min_sec** - Length of the album (hours,minutes,seconds)
- **genre** - Genre of the album
- **music_recording_sales_millions** - Music recording sales (millions in USD) on SONG://DATABASE
- **claimed_sales_millions** - Album's claimed sales (millions in USD) on SONG://DATABASE
- **date_released** - Date on which the album was released
- **soundtrack** - Indicates if the album is the movie soundtrack (Y) or (N)
- **rating_of_friends** - Indicates the rating from your friends from 1 to 10

The dataset can be seen below:

| Artist | Album | Released | Length | Genre | Music recording sales (millions) | Claimed sales (millions) | Released | Soundtrack | Rating (friends) |
|-----------------|---------------------------------|----------|----------|-----------------------------|----------------------------------|--------------------------|-----------|------------|------------------|
| Michael Jackson | Thriller | 1982 | 00:42:19 | Pop, rock, R&B | 46 | 65 | 30-Nov-82 | | 10.0 |
| AC/DC | Back in Black | 1980 | 00:42:11 | Hard rock | 26.1 | 50 | 25-Jul-80 | | 8.5 |
| Pink Floyd | The Dark Side of the Moon | 1973 | 00:42:49 | Progressive rock | 24.2 | 45 | 01-Mar-73 | | 9.5 |
| Whitney Houston | The Bodyguard | 1992 | 00:57:44 | Soundtrack/R&B, soul, pop | 26.1 | 50 | 25-Jul-80 | Y | 7.0 |
| Meat Loaf | Bat Out of Hell | 1977 | 00:46:33 | Hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | | 7.0 |
| Eagles | Their Greatest Hits (1971-1975) | 1976 | 00:43:08 | Rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | | 9.5 |
| Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | Disco | 20.6 | 40 | 15-Nov-77 | Y | 9.0 |
| Fleetwood Mac | Rumours | 1977 | 00:40:01 | Soft rock | 27.9 | 40 | 04-Feb-77 | | 9.5 |

Tuples

In Python, there are different data types: string, integer, and float. These data types can all be contained in a tuple as follows:



Now, let us create your first tuple with string, integer and float.

```
[1]: # Create your first tuple  
tuple1 = ("disco",10,1.2)
```

The type of variable is a **tuple**.

```
[2]: # Print the type of the tuple you created  
type(tuple1)  
[2]: tuple
```

Indexing

Each element of a tuple can be accessed via an index. The following table represents the relationship between the index and the items in the tuple. Each element can be obtained by the name of the tuple followed by a square bracket with the index number:

| | | |
|---|---------|--------------------|
| 0 | "disco" | Tuple1[0]= "disco" |
| 1 | 10 | Tuple1[1]= 10 |
| 2 | 1.2 | Tuple1[2]= 1.2 |

We can print out each value in the tuple:

```
[3]: # Print the variable on each index  
  
print(tuple1[0])  
print(tuple1[1])  
print(tuple1[2])  
  
disco  
10  
1.2
```

We can print out the **type** of each value in the tuple:

```
[4]: # Print the type of value on each index  
  
print(type(tuple1[0]))  
print(type(tuple1[1]))  
print(type(tuple1[2]))  
  
<class 'str'>  
<class 'int'>  
<class 'float'>
```

We can also use negative indexing. We use the same table above with corresponding negative values:

| | | | |
|----|---|---------|---------------------|
| -3 | 0 | “disco” | Tuple1[-3]= “disco” |
| -2 | 1 | 10 | Tuple1[-2]= 10 |
| -1 | 2 | 1.2 | Tuple1[-1]= 1.2 |

We can obtain the last element as follows (this time we will not use the print statement to display the values):

```
[5]: # Use negative index to get the value of the last element  
  
tuple1[-1]
```

```
[5]: 1.2
```

We can display the next two elements as follows:

```
[6]: # Use negative index to get the value of the second last element  
  
tuple1[-2]
```

```
[6]: 10
```

```
[7]: # Use negative index to get the value of the third last element  
  
tuple1[-3]
```

```
[7]: 'disco'
```

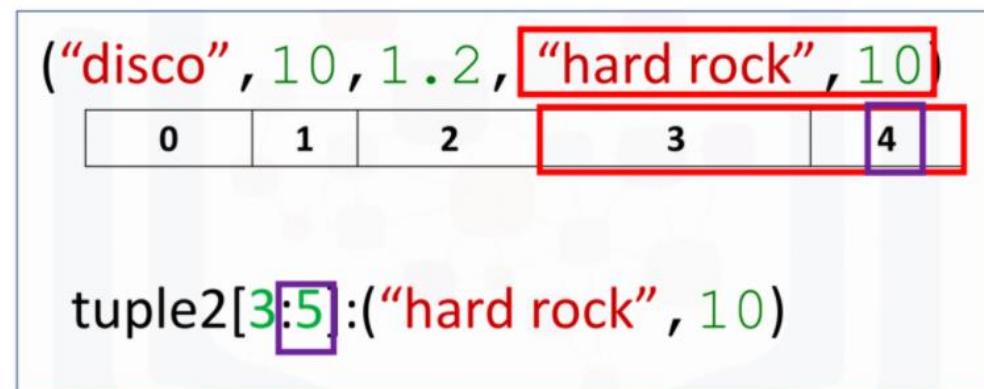
Concatenate Tuples

We can concatenate or combine tuples by using the `+` sign:

```
[8]: # Concatenate two tuples  
  
tuple2 = tuple1 + ("hard rock", 10)  
tuple2
```

```
[8]: ('disco', 10, 1.2, 'hard rock', 10)
```

We can slice tuples obtaining multiple values as demonstrated by the figure below:



Slicing

We can slice tuples, obtaining new tuples with the corresponding elements:

```
[10]: # Slice from index 0 to index 2  
tuple2[0:3]
```

```
[10]: ('disco', 10, 1.2)
```

We can obtain the last two elements of the tuple:

```
[11]: # Slice from index 3 to index 4  
tuple2[3:5]
```

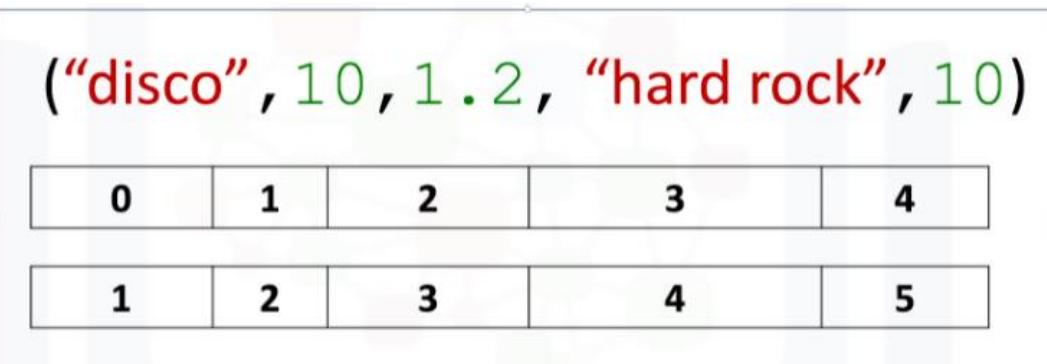
```
[11]: ('hard rock', 10)
```

We can obtain the length of a tuple using the length command:

```
[12]: # Get the length of tuple  
len(tuple2)
```

```
[12]: 5
```

This figure shows the number of elements:



Sorting

Consider the following tuple:

```
[13]: # A sample tuple  
  
Ratings = (0, 9, 6, 5, 10, 8, 9, 6, 2)
```

We can sort the values in a tuple and save it to a new tuple:

```
[14]: # Sort the tuple  
  
RatingsSorted = sorted(Ratings)  
RatingsSorted
```

```
[14]: [0, 2, 5, 6, 6, 8, 9, 9, 10]
```

Nested Tuple

A tuple can contain another tuple as well as other more complex data types. This process is called 'nesting'. Consider the following tuple with several elements:

```
[16]: # Create a nest tuple  
  
NestedT =(1, 2, ("pop", "rock") ,(3,4),("disco",(1,2)))
```

Each element in the tuple, including other tuples, can be obtained via an index as shown in the figure:

$$\text{NT} = (1, 2, (\text{"pop"}, \text{"rock"}), (3,4), (\text{"disco"}, (1,2)))$$



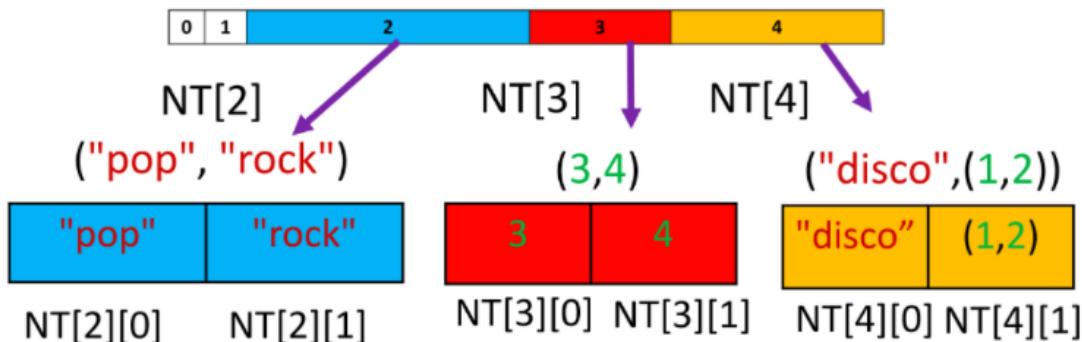
```
[17]: # Print element on each index

print("Element 0 of Tuple: ", NestedT[0])
print("Element 1 of Tuple: ", NestedT[1])
print("Element 2 of Tuple: ", NestedT[2])
print("Element 3 of Tuple: ", NestedT[3])
print("Element 4 of Tuple: ", NestedT[4])
```

```
Element 0 of Tuple: 1
Element 1 of Tuple: 2
Element 2 of Tuple: ('pop', 'rock')
Element 3 of Tuple: (3, 4)
Element 4 of Tuple: ('disco', (1, 2))
```

We can use the second index to access other tuples as demonstrated in the figure:

$NT = (1, 2, ("pop", "rock"), (3, 4), ("disco", (1, 2)))$



We can access the nested tuples:

```
[19]: # Print element on each index, including nest indexes

print("Element 2, 0 of Tuple: ", NestedT[2][0])
print("Element 2, 1 of Tuple: ", NestedT[2][1])
print("Element 3, 0 of Tuple: ", NestedT[3][0])
print("Element 3, 1 of Tuple: ", NestedT[3][1])
print("Element 4, 0 of Tuple: ", NestedT[4][0])
print("Element 4, 1 of Tuple: ", NestedT[4][1])
```

```
Element 2, 0 of Tuple: pop
Element 2, 1 of Tuple: rock
Element 3, 0 of Tuple: 3
Element 3, 1 of Tuple: 4
Element 4, 0 of Tuple: disco
Element 4, 1 of Tuple: (1, 2)
```

We can access strings in the second nested tuples using a third index:

```
[20]: # Print the first element in the second nested tuples
```

```
NestedT[2][1][0]
```

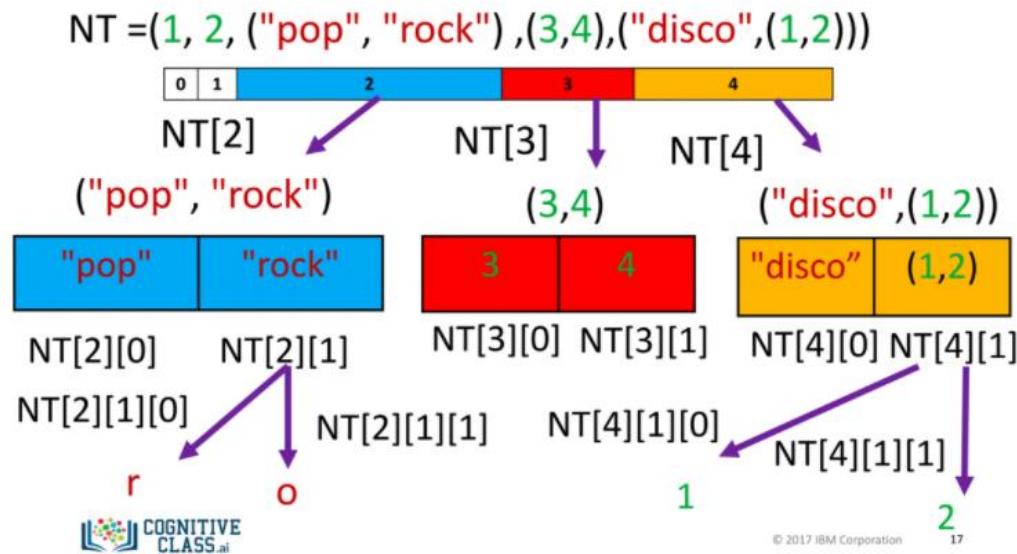
```
[20]: 'r'
```

```
[22]: # Print the second element in the second nested tuples
```

```
NestedT[2][1][1]
```

```
[22]: 'o'
```

We can use a tree to visualise the process. Each new index corresponds to a deeper level in the tree:



Similarly, we can access elements nested deeper in the tree with a third index:

```
[24]: # Print the first element in the second nested tuples
```

```
NestedT[4][1][0]
```

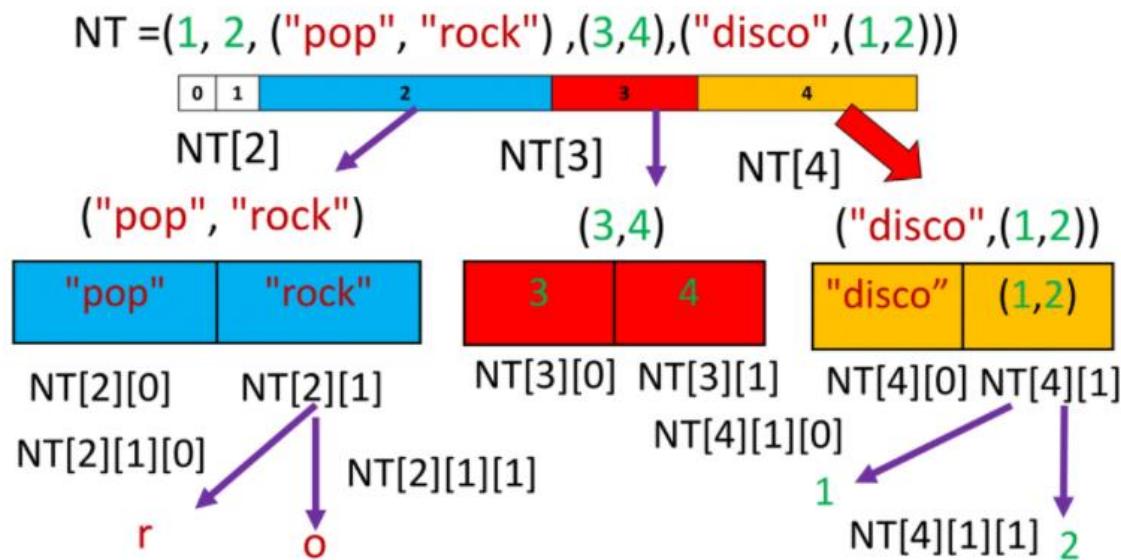
```
[24]: 1
```

```
[25]: # Print the second element in the second nested tuples
```

```
NestedT[4][1][1]
```

```
[25]: 2
```

The following figure shows the relationship of the tree and the element `NestedT[4][1][1]`:



Quiz on Tuples

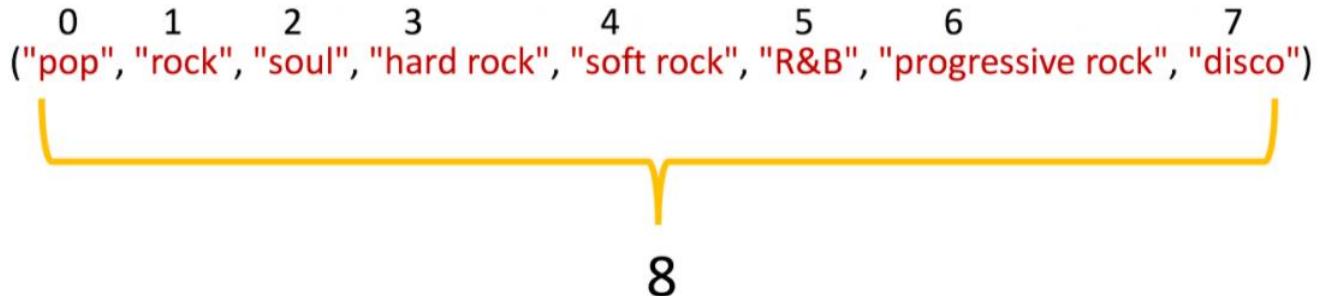
Consider the following tuple:

```
[26]: # sample tuple  
  
genres_tuple = ("pop", "rock", "soul", "hard rock", "soft rock",  
                 "R&B", "progressive rock", "disco")..
```

Find the length of the tuple, `genres_tuple`:

```
[28]: # Write your code below and press Shift+Enter to execute  
len(genres_tuple)
```

```
[28]: 8
```



▼ Click here for the solution

```
len(genres_tuple)
```

Access the element, with respect to index 3:

```
[31]: # Write your code below and press Shift+Enter to execute  
genres_tuple[3]
```

```
[31]: 'hard rock'
```

▼ Click here for the solution

```
genres_tuple[3]
```

Use slicing to obtain indexes 3, 4 and 5:

```
[32]: # Write your code below and press Shift+Enter to execute  
genres_tuple[3:6]
```

```
[32]: ('hard rock', 'soft rock', 'R&B')
```

▼ Click here for the solution

```
genres_tuple[3:6]
```

Find the first two elements of the tuple `genres_tuple`:

```
[33]: # Write your code below and press Shift+Enter to execute  
genres_tuple[0:2]
```

```
[33]: ('pop', 'rock')
```

▼ Click here for the solution

```
genres_tuple[0:2]
```

Find the first index of "disco":

```
[ ]: # Write your code below and press Shift+Enter to execute  
genres_tuple.index("disco")
```

▼ Click here for the solution

```
genres_tuple.index("disco")
```

Generate a sorted List from the Tuple `C_tuple=(-5, 1, -3)`:

```
[34]: # Write your code below and press Shift+Enter to execute  
C_tuple = (-5, 1, -3)  
C_list = sorted(C_tuple)  
C_list
```

```
[34]: [-5, -3, 1]
```

▼ Click here for the solution

```
C_tuple = (-5, 1, -3)  
C_list = sorted(C_tuple)  
C_list
```

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

Practice Quiz: Lists and Tuples

 [Bookmark this page](#)

Question 1

1/1 point (ungraded)

Consider the following tuple:

```
say_what=('say', 'what', 'you', 'will')
```

What is the result of the following `say_what[-1]` ?

'will'

'what'

'you'

'say'



Answer

Correct: Correct. An index of -1 corresponds to the last index of the tuple, in this case, the string 'will'.

Submit

You have used 1 of 2 attempts

Save

Correct (1/1 point)

Question 2

1/1 point (ungraded)

Consider the following tuple **A=(1,2,3,4,5)**, what is the result of the following:**A[1:4]**:

(2, 3, 4, 5)

(3, 4, 5)

(2, 3, 4)



Answer

Correct: Correct. These indexes correspond to elements 1,2 and 3 of the tuple.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 3

1/1 point (ungraded)

Consider the following tuple **A=(1,2,3,4,5)**. What is the result of the following: **len(A)**

6

5

4



Answer

Correct: Correct. The function len returns the number of items of a tuple.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 4

1/1 point (ungraded)

Consider the following list $B=[1,2,[3,'a'],[4,'b']]$. What is the result of the following: $B[3][1]$

"c"

"b"

[4,"b"]

**Answer**

Correct: Correct.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 5

1/1 point (ungraded)

What is the result of the following operation?

$[1,2,3]+[1,1,1]$

TypeError

[1, 2, 3, 1, 1, 1]

[2,3,4]

**Answer**

Correct: Correct. The addition operator corresponds to concatenating a list.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 6

1/1 point (ungraded)

What is the length of the list **A = [1]** after the following operation: **A.append([2,3,4,5])**

2

5



Answer

Correct: Correct. Append only adds one element to the list.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 7

1/1 point (ungraded)

What is the result of the following: "**Hello Mike**".split()

["H"]

["HelloMike"]

["Hello","Mike"]



Answer

Correct:

Correct. The method split separates a string into a list based on case the string is split using spaces.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Dictionaries

Let's cover Dictionaries in Python. Dictionaries are a type of collection in Python.

If you recall, a list is integer indexes. These are like addresses.

A list also has elements. A **dictionary** has keys and values.

The **key** is analogous to the index. They are like addresses, but they don't have to be integers. They are usually characters.

The **values** are similar to the element in a list and contain information. To create a dictionary, we use curly brackets.

The **keys** are the first elements. They must be immutable and unique.

Each key is followed by a value separated by a colon.

The values can be immutable, mutable, and duplicates. Each key and value pair is separated by a comma.

Consider the following example of a dictionary.

The album title is the key, and the value is the released data. We can use yellow to highlight the keys and leave the values in white. It is helpful to use the table to visualize

a dictionary where the first column represents the keys, and the second column represents the values. We can add a few more examples to the dictionary.

We can also assign the dictionary to a variable. The key is used to look at the value.

We use square brackets. The argument is the key. This outputs the value.

Using the key of "Back in Black," this returns the value of 1980. The key, "The Dark Side Of The Moon," gives us the value of 1973. Using the key, "The bodyguard,"

gives us the value 1992 and so on. We can add a new entry to the dictionary as follows.

This will add the value 2007 with a new key called "Graduation." We can delete an entry as follows. This gets rid of the key "Thriller" and its value.

We can verify if an element is in the dictionary using the "in" command as follows:

The command checks the keys. If they are in the dictionary, they return a true.

If we try the same command with a key that is not in the dictionary,

we get a false. In order to see all the keys in the dictionary, we can use the method keys to get the keys. The output is a list-like object with all the keys. In the same way, we can obtain the n the same way, we can obtain the values using the method values.



Python

Dictionaries

List

| Index | Element |
|-------|-----------|
| 0 | Element 1 |
| 1 | Element 2 |
| 2 | Element 3 |
| 3 | Element 4 |
| | |

Dictionary

| Key: is a index by label | Value |
|--------------------------|---------|
| Key 1 | Value 1 |
| Key 1 | Value 2 |
| Key 2 | Value 3 |
| Key 3 | Value 4 |
| | |

Dictionaries

- Dictionaries are denoted with curly Brackets {}
- The keys have to be immutable and unique
- The values can be can immutable, mutable and duplicates
- Each key and value pair is separated by a comma

```
{ "key1":1, "key2" :"2","key3" :[3,3,3], "key4":(4,4,4), ('key5'):5}
```

| | |
|-----------------------------|--------|
| "Thriller" | "1982" |
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |



Like **Search** if you type DICT["Back in Black"]:"1980"

| | |
|-----------------------------|--------|
| "Thriller" | "1982" |
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |
| "Bat Out of Hell" | "1977" |
| "Their Greatest..." | "1976" |
| Saturday Night Fever | "1977" |
| "Rumors" | "1977" |
| 'Graduation' | "2007" |

DICT['Graduation']='2007'

Add a key 'Graduation' with a value of '2007'

| | |
|-----------------------------|--------|
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |
| "Bat Out of Hell" | "1977" |
| "Their Greatest..." | "1976" |
| Saturday Night Fever | "1977" |
| "Rumors" | "1977" |

`del(DICT['Thriller'])`



Delete `del(DICT['Thriller'])` "Thriller" and its value "1982"

| | |
|-----------------------------|--------|
| "Thriller" | "1982" |
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |
| "Bat Out of Hell" | "1977" |
| "Their Greatest..." | "1976" |
| Saturday Night Fever | "1977" |
| "Rumors" | "1977" |

'The Bodyguard' in DICT

True



Can Verify if "The Bodyguard" is in the table using the "in". If it's there it will yield True.

| | |
|-----------------------------|--------|
| "Thriller" | "1982" |
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |
| "Bat Out of Hell" | "1977" |
| "Their Greatest..." | "1976" |
| Saturday Night Fever | "1977" |
| "Rumors" | "1977" |

"Starboy" in DICT

False



If "Starboy" is not in the table then it yield False

| | |
|-----------------------------|--------|
| "Thriller" | "1982" |
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |
| "Bat Out of Hell" | "1977" |
| "Their Greatest..." | "1976" |
| "Saturday Night Fever" | "1977" |
| "Rumors" | "1977" |

DICT.keys()=["Thriller", "Back in Black", "The Dark Side of the Moon", "The Bodyguard", "Bat Out of Hell", "Their Greatest...", "Saturday Night Fever", "Rumors"]

In order to see all the keys in the dictionary, DICT.keys() = ["Thriller", and so on.]

| | |
|-----------------------------|--------|
| "Thriller" | "1982" |
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |
| "Bat Out of Hell" | "1977" |
| "Their Greatest..." | "1976" |
| Saturday Night Fever | "1977" |
| "Rumors" | "1977" |

DICT.values() =["1982", "1980", "1973", "1992", "1977", "1976", "1977", "1977"]

In order to see all the values in the dictionary, DICT.values() = ["1982", and so on.]



IBM Developer
SKILLS NETWORK

Dictionaries in Python

Estimated time needed: **20** minutes

Objectives

After completing this lab you will be able to:

- Work with libraries in Python, including operations

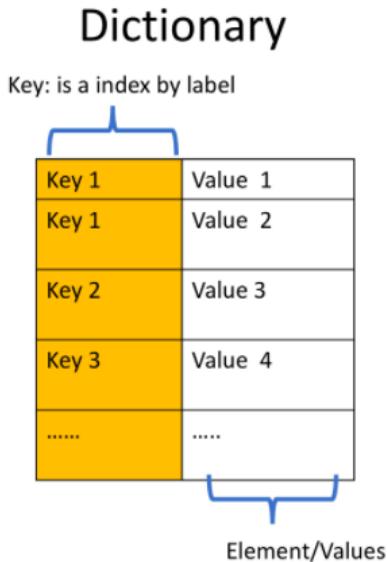
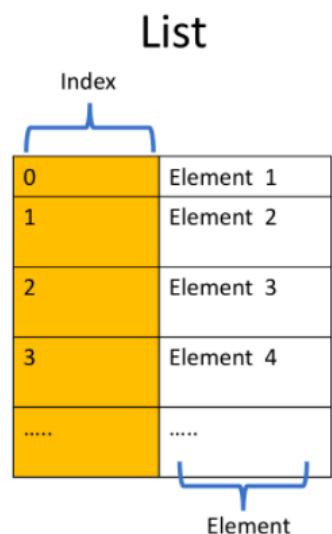
Table of Contents

- Dictionaries
 - What are Dictionaries?
 - Keys
- Quiz on Dictionaries

Dictionaries

What are Dictionaries?

A dictionary consists of keys and values. It is helpful to compare a dictionary to a list. Instead of the numerical indexes such as a list, dictionaries have keys. These keys are the keys that are used to access values within a dictionary.



An example of a Dictionary `Dict`:

```
In [1]: # Create the dictionary

Dict = {"key1": 1, "key2": "2", "key3": [3, 3, 3], "key4": (4, 4, 4), ('key5'): 5, (0, 1): 6}

Out[1]: {'key1': 1,
         'key2': '2',
         'key3': [3, 3, 3],
         'key4': (4, 4, 4),
         'key5': 5,
         (0, 1): 6}
```

The keys can be strings:

```
In [8]: # Access to the value by the key

a = Dict["key1"]
b = Dict["key4"]

print(a)
print(b)
```

```
1
(4, 4, 4)
```

Keys can also be any immutable object such as a tuple:

```
In [12]: # Access to the value by the key

Dict[(0, 1)]
```

```
6
```

Each key is separated from its value by a colon ":". Commas separate the items, and the whole dictionary is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this "{}".

```
In [13]: # Create a sample dictionary

release_year_dict = {"Thriller": "1982", "Back in Black": "1980", \
                     "The Dark Side of the Moon": "1973", "The Bodyguard": "1992", \
                     "Bat Out of Hell": "1977", "Their Greatest Hits (1971-1975)": "1976", \
                     "Saturday Night Fever": "1977", "Rumours": "1977"}
release_year_dict
```

```
Out[13]: {'Thriller': '1982',
          'Back in Black': '1980',
          'The Dark Side of the Moon': '1973',
          'The Bodyguard': '1992',
          'Bat Out of Hell': '1977',
          'Their Greatest Hits (1971-1975)': '1976',
          'Saturday Night Fever': '1977',
          'Rumours': '1977'}
```

In summary, like a list, a dictionary holds a sequence of elements. Each element is represented by a key and its corresponding value. Dictionaries are created with two curly braces containing keys and values separated by a colon. For every key, there can only be one single value, however, multiple keys can hold the same value. Keys can only be strings, numbers, or tuples, but values can be any data type.

It is helpful to visualize the dictionary as a table, as in the following image. The first column represents the keys, the second column represents the values.

| Key | Value |
|-----------------------------|--------|
| "Thriller" | "1982" |
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |
| "Bat Out of Hell" | "1977" |
| "Their Greatest..." | "1976" |
| Saturday Night Fever | "1977" |
| "Rumours" | "1977" |

Keys

You can retrieve the values based on the names:

In [14]:

```
# Get value by keys  
release_year_dict['Thriller']
```

Out[14]:

```
'1982'
```

This corresponds to:

| | |
|-----------------------------|--------|
| "Thriller" | "1982" |
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |
| "Bat Out of Hell" | "1977" |
| "Their Greatest..." | "1976" |
| "Saturday Night Fever" | "1977" |
| "Rumours" | "1977" |

Similarly for **The Bodyguard**

In [15]:

```
# Get value by key  
  
release_year_dict['The Bodyguard']
```

Out[15]:

| | |
|-----------------------------|--------|
| "Thriller" | "1982" |
| "Back in Black" | "1980" |
| "The Dark Side of the Moon" | "1973" |
| "The Bodyguard" | "1992" |
| "Bat Out of Hell" | "1977" |
| "Their Greatest..." | "1976" |
| "Saturday Night Fever" | "1977" |
| "Rumours" | "1977" |

Now let us retrieve the keys of the dictionary using the method `keys()`:

In [16]:

```
# Get all the keys in dictionary  
  
release_year_dict.keys()
```

Out[16]:

```
dict_keys(['Thriller', 'Back in Black', 'The Dark Side of the Moon', 'The Bodyguard', 'Bat Out of Hell', 'Their Greatest Hits (1971-1975)', 'Saturday Ni  
ght Fever', 'Rumours'])
```

You can retrieve the values using the method `values()`:

In [17]:

```
# Get all the values in dictionary  
  
release_year_dict.values()  
  
dict_values(['1982', '1980', '1973', '1992', '1977', '1976', '1977', '1977'])
```

We can add an entry:

```
In [18]: # Append value with key into dictionary  
  
release_year_dict['Graduation'] = '2007'  
release_year_dict
```

```
Out[18]: {'Thriller': '1982',  
          'Back in Black': '1980',  
          'The Dark Side of the Moon': '1973',  
          'The Bodyguard': '1992',  
          'Bat Out of Hell': '1977',  
          'Their Greatest Hits (1971-1975)': '1976',  
          'Saturday Night Fever': '1977',  
          'Rumours': '1977',  
          'Graduation': '2007'}
```

We can delete an entry:

```
In [19]: # Delete entries by key  
  
del(release_year_dict['Thriller'])  
del(release_year_dict['Graduation'])  
release_year_dict
```

```
Out[19]: {'Back in Black': '1980',  
          'The Dark Side of the Moon': '1973',  
          'The Bodyguard': '1992',  
          'Bat Out of Hell': '1977',  
          'Their Greatest Hits (1971-1975)': '1976',  
          'Saturday Night Fever': '1977',  
          'Rumours': '1977'}
```

We can verify if an element is in the dictionary:

```
In [23]: # Verify the key is in the dictionary  
  
'The Bodyguard' in release_year_dict
```

```
Out[23]: True
```

Quiz on Dictionaries

You will need this dictionary for the next two questions:

```
In [24]: # Question sample dictionary  
  
soundtrack_dic = {"The Bodyguard": "1992", "Saturday Night Fever": "1977"}  
soundtrack_dic  
  
Out[24]: {'The Bodyguard': '1992', 'Saturday Night Fever': '1977'}
```

a) In the dictionary `soundtrack_dic` what are the keys ?

```
In [25]: # Write your code below and press Shift+Enter to execute  
soundtrack_dic.keys()  
  
Out[25]: dict_keys(['The Bodyguard', 'Saturday Night Fever'])
```

Click here for the solution `python soundtrack_dic.keys() # The Keys "The Bodyguard" and "Saturday Night Fever"`

b) In the dictionary `soundtrack_dic` what are the values ?

```
In [29]: # Write your code below and press Shift+Enter to execute  
soundtrack_dic.values()  
  
Out[29]: dict_values(['1992', '1977'])
```

Click here for the solution `python soundtrack_dic.values() # The values are "1992" and "1977"`

You will need this dictionary for the following questions:

The Albums **Back in Black**, **The Bodyguard** and **Thriller** have the following music recording sales in millions 50, 50 and 65 respectively:

a) Create a dictionary `album_sales_dict` where the keys are the album name and the sales in millions are the values.

```
In [31]: # Write your code below and press Shift+Enter to execute  
album_sales_dict = { "Back in Black" : 50, "The Bodyguard" : 50, "Thriller" : 65}  
album_sales_dict  
  
Out[31]: {'Back in Black': 50, 'The Bodyguard': 50, 'Thriller': 65}
```

Click here for the solution `python album_sales_dict = {"The Bodyguard":50, "Back in Black":50, "Thriller":65}`

b) Use the dictionary to find the total sales of **Thriller**:

```
In [34]: # Write your code below and press Shift+Enter to execute  
album_sales_dict["Thriller"]  
  
Out[34]: 65
```

Click here for the solution `python album_sales_dict["Thriller"]`

c) Find the names of the albums from the dictionary using the method `keys()` :

c) Find the names of the albums from the dictionary using the method `keys()` :

In [35]:

```
# Write your code below and press Shift+Enter to execute  
album_sales_dict.keys()
```

Out[35]: `dict_keys(['Back in Black', 'The Bodyguard', 'Thriller'])`

[Click here for the solution](#) `~~~python album_sales_dict.keys()~~~`

d) Find the values of the recording sales from the dictionary using the method `values` :

In [36]:

```
# Write your code below and press Shift+Enter to execute  
album_sales_dict.values()
```

Out[36]: `dict_values([50, 50, 65])`

[Click here for the solution](#) `~~~python album_sales_dict.values()~~~`

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

Module 2 - Python Data Structures / Practice Quiz: Dictionaries

Question 1

1/1 point (ungraded)

What are the keys of the following dictionary: `{"a":1,"b":2}`

1,2

"a","b"



Answer

Correct: Correct, the key is the first element separated from its value by a colon.

Submit

You have used 2 of 2 attempts

Correct (1/1 point)

Question 2

1/1 point (ungraded)

Consider the following Python Dictionary: `Dict={"A":1,"B":2,"C":[3,3,3],"D":(4,4,4),'E':5,'F':6}` What is the result of the following operation: `Dict["D"]`

[3,3,3]

(4, 4, 4)

1



Answer

Correct: Correct, this corresponds to the key 'D' or `Dict['D']`

Submit

You have used 1 of 2 attempts

Save

Correct (1/1 point)

Sets

Let's cover sets. They are also a type of collection.

Sets are a type of collection. This means that like lists and tuples, you can input different Python types. Unlike lists and tuples,

- they are unordered. This means sets do not record element position.
- Sets only have unique elements. This means there is only one of a particular element in a set.

To define a set, you use curly brackets. You place the elements of a set within the curly brackets.

You notice there are duplicate items. When the actual set is created, duplicate items will not be present. You can convert a list to a set by using the function set, this is called type casting. You simply use the list as the input to the function set. The result will be a list converted to a set.

Let's go over an example. We start off with a list. We input the list to the function set. The function set returns a set. Notice how there are no duplicate elements.

Let's go over set operations. These could be used to change the set. Consider the set A.

Let's represent this set with a circle. If you are familiar with sets, this could be part of a venn diagram. A venn diagram is a tool that uses shapes usually to represent sets.

We can add an item to a set using the add-method. We just put the set name followed by a dot, then the add-method.

The argument is the new element of the set we would like to add, in this case, NSYNC.

The set A now has in NSYNC as an item. If we add the same item twice, nothing will happen as there can be no duplicates in a set. Let's say we would like to remove NSYNC from set A. We can also remove an item from a set using the remove-method. We just put the set name followed by a dot, then the remove-method.

The argument is the element of the set we would like to remove, in this case, NSYNC.

After the remove-method is applied to the set, set A does not contain the item NSYNC.

You can use this method for any item in the set. We can verify if an element is in the set using the in command as follows. The command checks that the item, in this case AC/DC, is in the set. If the item is in the set, it returns true. If we look for an item that is not in the set, in this case for the item Who, adds the item is not in the set, we will get a false.

These are types of mathematical set operations.

There are other operations we can do. There are lots of useful mathematical operations we can do between sets. Let's define the set album set one. We can represent it using a red circle or venn diagram. Similarly, we can define the set album set two. We can also represent it using a blue circle or venn diagram.

The intersection of two sets is a new set containing elements which are in both of those sets. It's helpful to use venn diagrams. The two circles that represent the sets combine, the overlap, represents the new set. As the overlap is comprised with the red circle and blue circle, we define the intersection in terms of and.

In Python, we use an ampersand to find the intersection of the two sets. If we overlay the values of the set over the circle placing the common elements in the overlapping area, we see the correspondence. After applying the intersection operation, all the items that are not in both sets disappear. In Python, we simply just place the ampersand between the two sets.

We see that both AC/DC and Back in Black are in both sets. The result is a new set album: set three containing all the elements in both albums set one and album set two.

The union of two sets is the new set of elements which contain all the items in both sets.

We can find the union of the sets album set one and album set two as follows. The result is a new set that has all the elements of album set one and album set two. This new set is represented in green.

Consider the new album set-album set three. The set contains the elements AC/DC and Back in Black. We can represent this with a Venn diagram, as all the elements and album set three are in album set one.

The circle representing album set one encapsulates the circle representing album set three.

We can check if a set is a subset using the is subset method.

As album set three is a subset of the album set one,
the result is true.

There is a lot more you can do with sets

Python

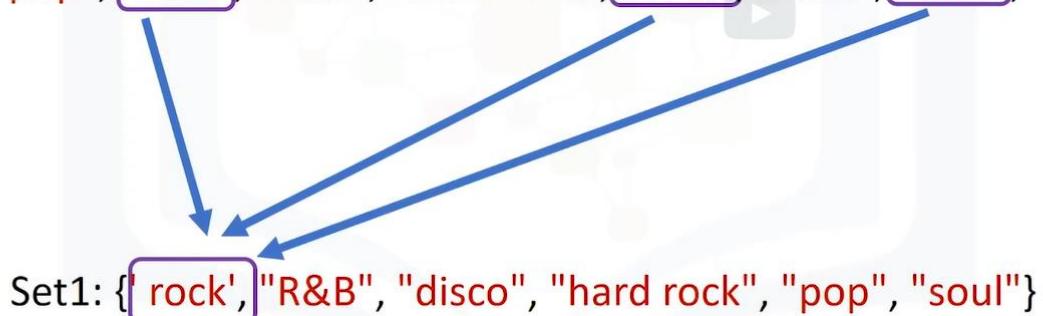
Sets

Sets

- Sets are a type of collection
 - This means that like lists and tuples you can input different Python types
- Unlike lists and tuples they are unordered
 - This means sets do not record element position
- Sets only have unique elements
 - This means there is only one of a particular element in a set

Sets: Creating a Set

Set1={"pop", "rock", "soul", "hard rock", "rock", "R&B", "rock", "disco" }



Sets: Creating a Set

```
album_list = ["Michael Jackson", "Thriller", "Thriller", 1982]
```

```
album_set = set(album_list)
```

```
album_set : {'Michael Jackson', 'Thriller', 1982}
```

set()

album_set

Sets: Creating a Set

```
album_list = ["Michael Jackson", "Thriller", "Thriller", 1982]
```

```
album_set = set(album_list)
```

album_list

set()

Sets: Creating a Set

```
album_list = ["Michael Jackson", "Thriller", "Thriller", 1982]
```

```
album_set = set(album_list)
```

```
album_set : {'Michael Jackson', 'Thriller', 1982}
```

set()

album_set

To ADD an Item

Set Operations

```
A :{"AC/DC", "Back in Black", "NSYNC", "Thriller"}
```

```
A.remove("NSYNC")
```

```
A:{ "AC/DC", "Back in Black", "Thriller"}
```

"Thriller", "Back in Black", "AC/DC"

Use **add** (**name of set.add**) if you want to add an element in a set. Nothing will happen if you add twice and so on, only once that it'll be included in the set.

To REMOVE an item

Set Operations

```
A :{"AC/DC", "Back in Black", "NSYNC", "Thriller"}
```

```
A.remove("NSYNC")
```

"NSYNC"

"Thriller", "Back in Black", "AC/DC"

Use **remove** (**name of set. remove**) to remove an item from the set.

To CHECK if an item is in the set

Set Operations

A:{“AC/DC”, “Back in Black”, “Thriller”}

“AC/DC” in A

True



Use the “in” to check if an item is in the set. If yes, it'll yield TRUE. If not in the set, you'll get a false.

Set Operations

A:{“AC/DC”, “Back in Black”, “Thriller”}

“Who” in A

False



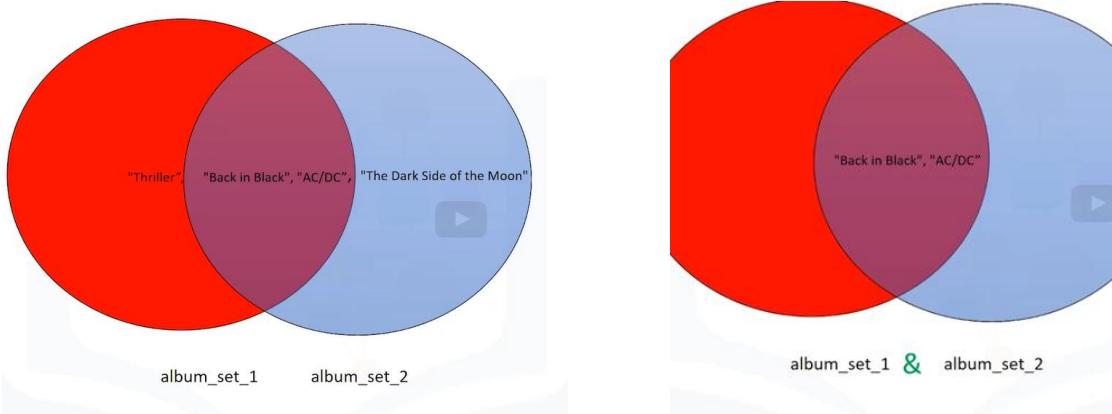
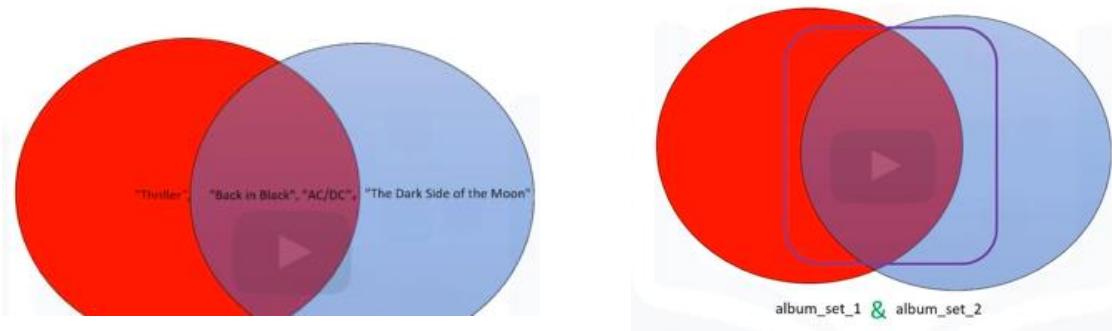
To find common element of 2 sets using “&”

Sets: Mathematical set operations

“AC/DC”, “Back in Black”, “Thriller”

album_set_1 = {"AC/DC", "Back in Black", "Thriller"}

Sets: Mathematical set operations



```
album_set_1 = {"AC/DC", "Back in Black", "Thriller"}
```

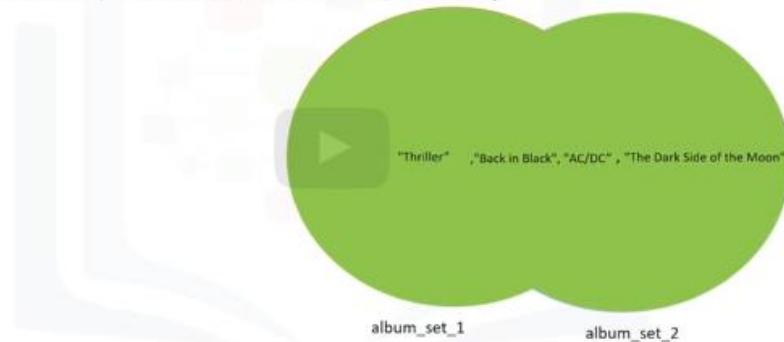
```
album_set_2 = {"AC/DC", "Back in Black", "The Dark Side of the Moon"}
```

```
album_set_3 = album_set_1 & album_set_2
```

```
album_set_3: {"AC/DC", "Back in Black"}
```

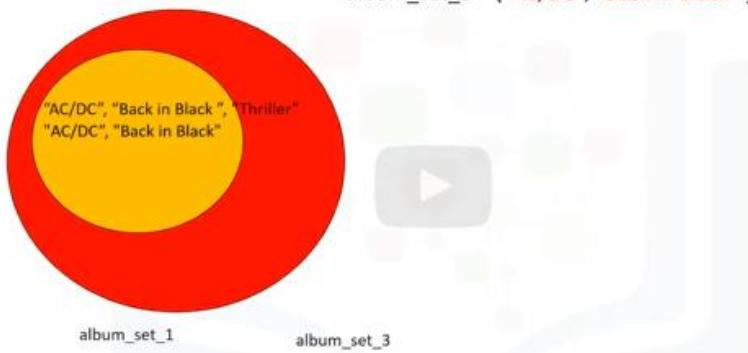
To combine 2 sets, use union - name of 1st set.union(name of 2nd set)

```
album_set_1.union(album_set_2)  
{AC/DC", "Back in Black", "The Dark Side of the Moon", "Thriller" }
```



To represent in a venn diagram of a 3rd set

```
album_set_1 = {"AC/DC", " Back in Black ", "Thriller }  
album_set_3 = {"AC/DC", " Back in Black "}
```



To check if a set is a subset, use issubset

```
album_set_1 = {"AC/DC", " Back in Black ", "Thriller }  
album_set_3 = {"AC/DC", " Back in Black "}  
album_set_3.issubset(album_set1)
```

True





IBM Developer
SKILLS NETWORK

Sets in Python

Estimated time needed: **20** minutes

Objectives

After completing this lab you will be able to:

- Work with sets in Python, including operations and logic operations.

Table of Contents

- Sets
 - Set Content
 - Set Operations
 - Sets Logic Operations
- Quiz on Sets

Sets

Set Content

A set is a unique collection of objects in Python. You can denote a set with a curly bracket {}. Python will automatically remove duplicate items:

Set1={"pop", "rock", "soul", "hard rock", "rock", "R&B", "rock", "disco" }

Set1: {"rock", "R&B", "disco", "hard rock", "pop", "soul"}

You can also create a set from a list as follows:

```
In [2]: # Convert list to set

album_list = [ "Michael Jackson", "Thriller", 1982, "00:42:19", \
               "Pop, Rock, R&B", 46.0, 65, "30-Nov-82", None, 10.0]
album_set = set(album_list)
album_set
```



```
Out[2]: {'00:42:19',
          10.0,
          1982,
          '30-Nov-82',
          46.0,
          65,
          'Michael Jackson',
          None,
          'Pop, Rock, R&B',
          'Thriller'}
```

Now let us create a set of genres:

```
In [3]: # Convert list to set

music_genres = set(["pop", "pop", "rock", "folk rock", "hard rock", "soul", \
                     "progressive rock", "soft rock", "R&B", "disco"])
music_genres
```



```
Out[3]: {'R&B',
          'disco',
          'folk rock',
          'hard rock',
          'pop',
          'progressive rock',
          'rock',
          'soft rock',
          'soul'}
```

Set Operations

Let us go over set operations, as these can be used to change the set. Consider the set A:

```
In [4]: # Sample set  
  
A = set(["Thriller", "Back in Black", "AC/DC"])  
A  
  
Out[4]: {'AC/DC', 'Back in Black', 'Thriller'}
```

We can add an element to a set using the `add()` method:

```
In [5]: # Add element to set  
  
A.add("NSYNC")  
A  
  
Out[5]: {'AC/DC', 'Back in Black', 'NSYNC', 'Thriller'}
```

If we add the same element twice, nothing will happen as there can be no duplicates in a set:

```
In [6]: # Try to add duplicate element to the set  
  
A.add("NSYNC")  
A  
  
Out[6]: {'AC/DC', 'Back in Black', 'NSYNC', 'Thriller'}
```

We can remove an item from a set using the `remove` method:

```
In [7]: # Remove the element from set  
  
A.remove("NSYNC")  
A  
  
Out[7]: {'AC/DC', 'Back in Black', 'Thriller'}
```

We can verify if an element is in the set using the `in` command:

```
In [8]: # Verify if the element is in the set  
  
"AC/DC" in A  
  
Out[8]: True
```

Sets Logic Operations

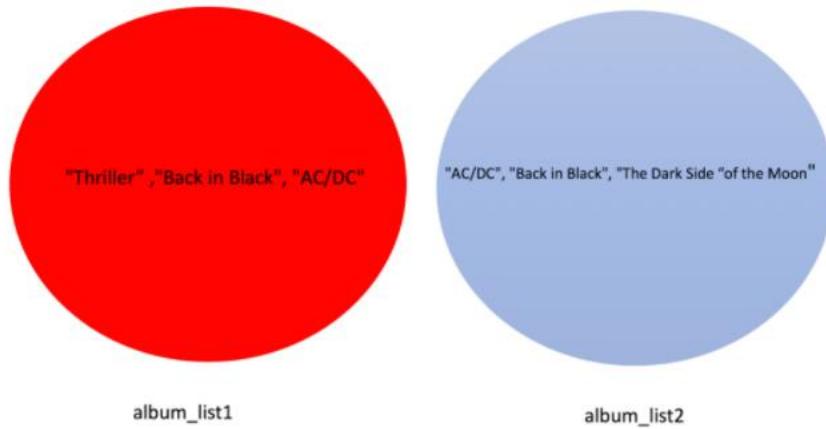
Remember that with sets you can check the difference between sets, as well as the symmetric difference, intersection, and union:

Consider the following two sets:

In [9]:

Sample Sets

```
album_set1 = set(["Thriller", 'AC/DC', 'Back in Black'])
album_set2 = set([ "AC/DC", "Back in Black", "The Dark Side of the Moon"])
```



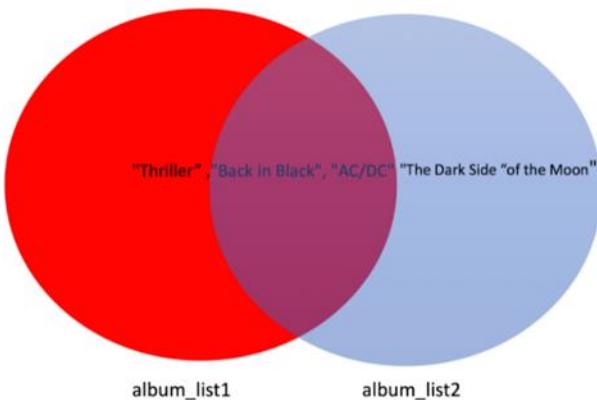
In [10]:

Print two sets

```
album_set1, album_set2
```

```
Out[10]: ({'AC/DC', 'Back in Black', 'Thriller'},
           {'AC/DC', 'Back in Black', 'The Dark Side of the Moon'})
```

As both sets contain **AC/DC** and **Back in Black** we represent these common elements with the intersection of two circles.



You can find the intersect of two sets as follow using `&`:

```
In [11]: # Find the intersections  
  
intersection = album_set1 & album_set2  
intersection
```

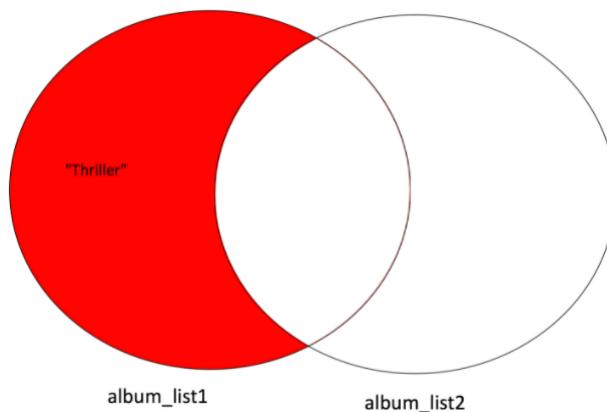
```
Out[11]: {'AC/DC', 'Back in Black'}
```

You can find all the elements that are only contained in `album_set1` using the `difference` method:

```
In [12]: # Find the difference in set1 but not set2  
  
album_set1.difference(album_set2)
```

```
Out[12]: {'Thriller'}
```

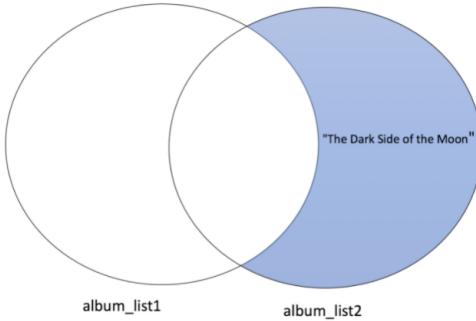
You only need to consider elements in `album_set1`; all the elements in `album_set2`, including the intersection, are not included.



The elements in `album_set2` but not in `album_set1` is given by:

```
In [13]: album_set2.difference(album_set1)
```

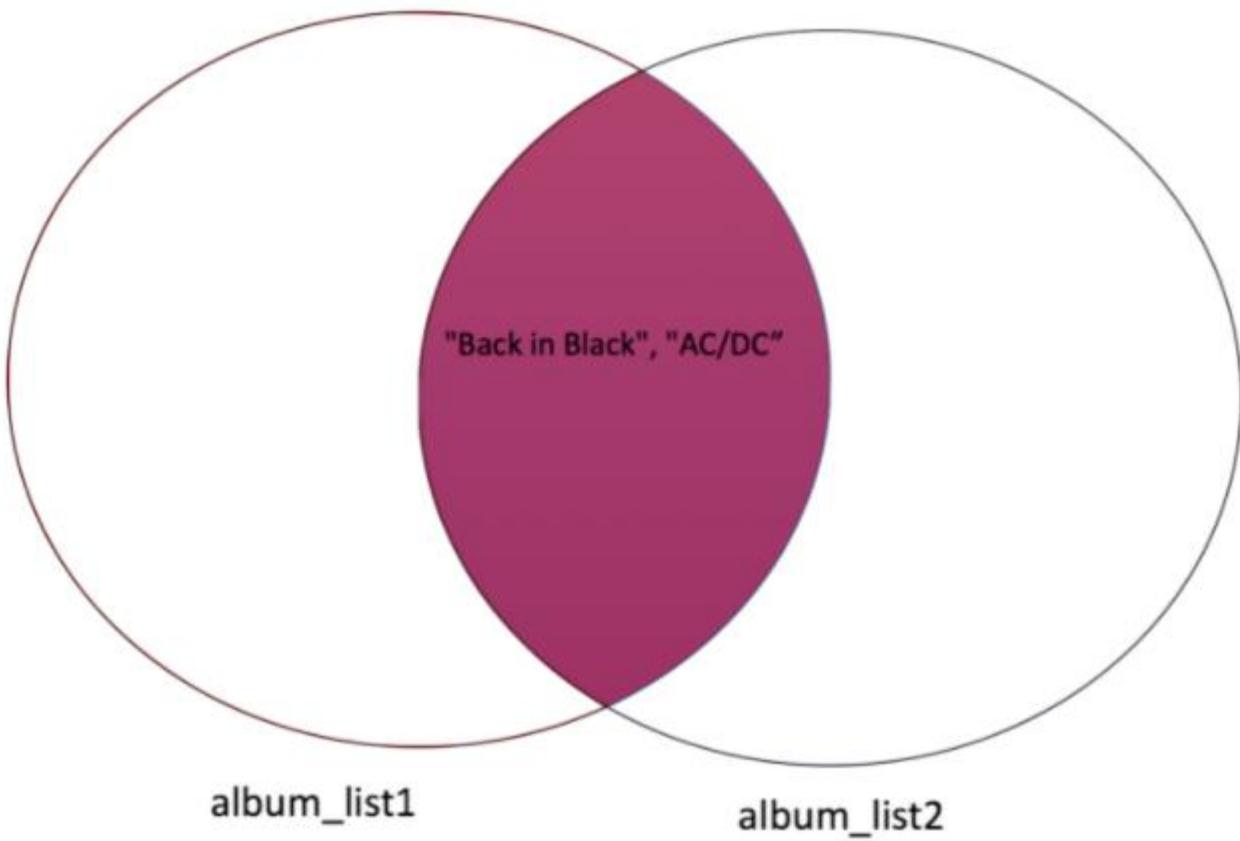
```
Out[13]: {'The Dark Side of the Moon'}
```



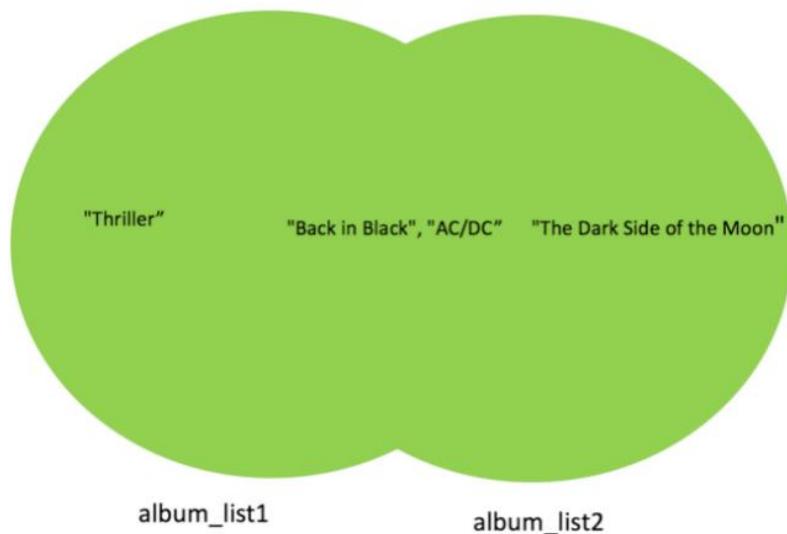
You can also find the intersection of `album_list1` and `album_list2`, using the `intersection` method:

```
In [14]: # Use intersection method to find the intersection of album_list1 and album_list2  
album_set1.intersection(album_set2)  
  
Out[14]: {'AC/DC', 'Back in Black'}
```

This corresponds to the intersection of the two circles:



The union corresponds to all the elements in both sets, which is represented by coloring both circles:



The union is given by:

```
In [15]: # Find the union of two sets  
  
album_set1.union(album_set2)  
  
Out[15]: {'AC/DC', 'Back in Black', 'The Dark Side of the Moon', 'Thriller'}
```

And you can check if a set is a superset or subset of another set, respectively, like this:

```
In [16]: # Check if superset  
  
set(album_set1).issuperset(album_set2)  
  
Out[16]: False
```

```
In [17]: # Check if subset  
  
set(album_set2).issubset(album_set1)  
  
Out[17]: False
```

Here is an example where `issubset()` and `issuperset()` return true:

```
In [18]: # Check if subset  
  
set({"Back in Black", "AC/DC"}).issubset(album_set1)  
  
Out[18]: True
```

```
In [19]: # Check if superset  
  
album_set1.issuperset({"Back in Black", "AC/DC"})  
  
Out[19]: True
```

Quiz on Sets

Convert the list `['rap', 'house', 'electronic music', 'rap']` to a set:

In [20]:

```
# Write your code below and press Shift+Enter to execute
set(['rap','house','electronic music', 'rap'])
```

Out[20]:

```
{'electronic music', 'house', 'rap'}
```

[Click here for the solution](#) `python set(['rap','house','electronic music','rap'])`

Consider the list `A = [1, 2, 2, 1]` and set `B = set([1, 2, 2, 1])`, does `sum(A) = sum(B)`

In [31]:

```
# Write your code below and press Shift+Enter to execute
A = [1, 2, 2, 1]
B = set([1, 2, 2, 1])

print("the sum of A is:", sum(A))
print("the sum of B is:", sum(B))

if sum(A) == sum(B):
    print("the sum of A and the sum of B is equal")
else:
    print("the sum of A and the sum of B is different")
```

```
the sum of A is: 6
the sum of B is: 3
the sum of A and the sum of B is different
```

[Click here for the solution](#) `python A = [1, 2, 2, 1] B = set([1, 2, 2, 1]) print("the sum of A is:", sum(A)) print("the sum of B is:", sum(B))`

Create a new set `album_set3` that is the union of `album_set1` and `album_set2`:

In [32]:

```
# Write your code below and press Shift+Enter to execute

album_set1 = set(["Thriller", 'AC/DC', 'Back in Black'])
album_set2 = set([ "AC/DC", "Back in Black", "The Dark Side of the Moon"])

album_set3 = album_set1.union(album_set2)
album_set3
```

Out[32]:

```
{'AC/DC', 'Back in Black', 'The Dark Side of the Moon', 'Thriller'}
```

[Click here for the solution](#) `python album_set3 = album_set1.union(album_set2)` `album_set3`

Find out if `album_set1` is a subset of `album_set3`:

In []:

```
# Write your code below and press Shift+Enter to execute
```

[Click here for the solution](#) `python album_set1.issubset(album_set3)`

Find out if `album_set1` is a subset of `album_set3`:

[25]:

```
# Write your code below and press Shift+Enter to execute
album_set1.issubset(album_set3)
```

[25]:

▼ [Click here for the solution](#)
`album_set1.issubset(album_set3)`

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

S



Practice Quiz: Sets

Bookmark

Question 1

1/1 point (ungraded)

Consider the following set: `{"A","A"}`. What will the result be when the set is created?

{"A"}

{"A","A"}



Answer

Correct: Correct, there are no duplicate values in a set.

Submit

You have used 1 of 2 attempts

Save

✓ Correct (1/1 point)

Question 2

1/1 point (ungraded)

What is the result of the following: `type(set([1,2,3]))`

list

set



Answer

Correct: Correct, the function set casts the list to a set before we apply the type function.

Submit

You have used 2 of 2 attempts

✓ Correct (1/1 point)

Question 3

1/1 point (ungraded)

What method do you use to add an element to a set?

add

extend

append

**Answer**

Correct: Correct.

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 4

1/1 point (ungraded)

What is the result of the following operation: `{'a','b'} & {'a'}`

`{'a','b'}`

`{'a'}`

**Answer**

Correct: Correct, the intersection operation finds the elements that are in both sets.

Submit

You have used 2 of 2 attempts

Correct (1/1 point)

Graded Quiz: Python Data Structures

Bookmarked

Graded Quiz due Jan 11, 2022 12:55 +08

Question 1

1/1 point (graded)

Consider the tuple **tuple1=("A","B","C")**, what is the result of the following operation **tuple1[-1]**?

"A"

"C"

"B"



Answer

Correct: correct, the index -1 corresponds to the last element of the tuple.

Submit

You have used 1 of 2 attempts

Reset

Question 2

1/1 point (graded)

Consider the tuple **A=((1),[2,3],[4])**, that contains a tuple and list. What is the result of the following operation **A[2]** ?

[2,3]

[4]

1



Answer

Correct: correct, the index 2 corresponds to the third element in the tuple, which contains another list.

Submit

You have used 2 of 2 attempts

✓ Correct (1/1 point)

Question 3

1/1 point (graded)

Consider the following dictionary:

{ "The Bodyguard": "1992", "Saturday Night Fever": "1977" }

select the values

"1977"

"1992"

"The Bodyguard"

"Saturday Night Fever"



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 4

1/1 point (graded)

The variable **release_year_dict** is a Python Dictionary, what is the result of applying the following method:
release_year_dict.keys() ?

retrieve the keys of the dictionary

retrieves the values of the dictionary



Answer

Correct: correct, the method returns the keys

Submit

You have used 2 of 2 attempts

✓ Correct (1/1 point)

Question 5

1/1 point (graded)

What is the result of the following: '1' in {'1','2'}?

True

False



Answer

Correct: correct

You have used 1 of 1 attempt

 Correct (1/1 point)

Module 3 - Python Programming Fundamentals

- ✓ Module Introduction and Learning Objectives 1 min
 - ✓ Video: Conditions and Branching (10:17) 11 min
 - ✓ Hands-on Lab: Conditions and Branching 1 min + 1 activity
 - ✓ Practice Quiz: Conditions and Branching 1 activity
 - ✓ Video: Loops (6:45) 7 min
 - ✓ Hands-on Lab: Loops 1 min + 1 activity
 - ✓ Practice Quiz: Loops 1 activity
 - ✓ Video: Functions (13:31) 14 min
 - ✓ Hands-on Lab: Functions 1 min + 1 activity
 - ✓ Practice Quiz: Functions 1 activity
 - ✓ Video: Exception Handling (3:49) 4 min
 - ✓ Hands-On Lab: Exception Handling 1 min + 1 activity
 - ✓ Practice Quiz: Exception Handling 1 activity
 - ✓ Video: Objects and Classes (10:52) 11 min
 - ✓ Hands-on Lab: Objects and Classes 1 min + 1 activity
 - ✓ Discussion Prompt: Classes in Python 1 min + 1 activity
 - ✓ Practice Quiz: Objects and Classes 1 activity
 - ✓ Graded Quiz: Python Programming Fundamentals (5 Questions) 1 activity
-



Conditions in Python

Estimated time needed: **20** minutes

Objectives

After completing this lab you will be able to:

- work with condition statements in Python, including operators, and branching.

Table of Contents

- Condition Statements
 - Comparison Operators
 - Branching
 - Logical operators
- Quiz on Condition Statement

Condition Statements

Comparison Operators

Comparison operations compare some value or operand and, based on a condition, they produce a Boolean. When comparing two values you can use these operators:

- equal: `==`
- not equal: `!=`
- greater than: `>`
- less than: `<`
- greater than or equal to: `>=`
- less than or equal to: `<=`

Let's assign `a` a value of 5. Use the equality operator denoted with two equal `==` signs to determine if two values are equal. The case below compares the variable `a` with 6.

```
In [1]: # Condition Equal
```

```
a = 5  
a == 6
```

```
Out[1]: False
```

The result is **False**, as 5 does not equal to 6.

Consider the following equality comparison operator `i > 5`. If the value of the left operand, in this case the variable `i`, is greater than the value of the right operand, in this case 5, then the statement is **True**. Otherwise, the statement is **False**. If `i` is equal to 6, because 6 is larger than 5, the output is **True**.

```
In [2]: # Greater than Sign
```

```
i = 6  
i > 5
```

```
Out[2]: True
```

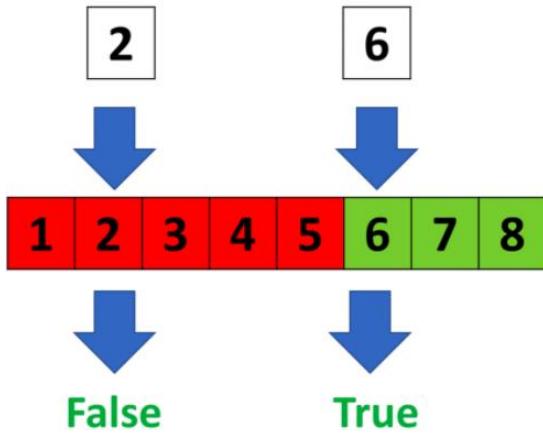
Set `i = 2`. The statement is false as 2 is not greater than 5:

```
In [3]: # Greater than Sign
```

```
i = 2  
i > 5
```

```
Out[3]: False
```

Let's display some values for `i` in the figure. Set the values greater than 5 in green and the rest in red. The green region represents where the condition is **True**, the red where the statement is **False**. If the value of `i` is 2, we get **False** as the 2 falls in the red region. Similarly, if the value for `i` is 6 we get a **True** as the condition falls in the green region.



The inequality test uses an exclamation mark preceding the equal sign, if two operands are not equal then the condition becomes **True**. For example, the following condition will produce **True** as long as the value of *i* is not equal to 6:

```
In [4]: # Inequality Sign
i = 2
i != 6
```

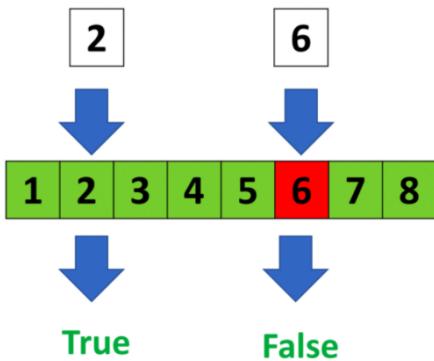
```
Out[4]: True
```

When *i* equals 6 the inequality expression produces **False**.

```
In [5]: # Inequality Sign
i = 6
i != 6
```

```
Out[5]: False
```

See the number line below. when the condition is **True** the corresponding numbers are marked in green and for where the condition is **False** the corresponding number is marked in red. If we set *i* equal to 2 the operator is true as 2 is in the green region. If we set *i* equal to 6, we get a **False** as the condition falls in the red region.



We can apply the same methods on strings. For example, use an equality operator on two different strings. As the strings are not equal, we get a **False**.

```
In [6]: # Use Equality sign to compare the strings
         "ACDC" == "Michael Jackson"
Out[6]: False
```

If we use the inequality operator, the output is going to be **True** as the strings are not equal.

```
In [7]: # Use Inequality sign to compare the strings
         "ACDC" != "Michael Jackson"
Out[7]: True
```

Inequality operation is also used to compare the letters/words/symbols according to the ASCII value of letters. The decimal value shown in the following table represents the order of the character:

For example, the ASCII code for ! is 33, while the ASCII code for + is 43. Therefore + is larger than ! as 43 is greater than 33.

Similarly, the value for **A** is 65, and the value for **B** is 66 therefore:

```
In [8]: # Compare characters  
  
'B' > 'A'
```

```
Out[8]: True
```

When there are multiple letters, the first letter takes precedence in ordering:

```
In [9]: # Compare characters  
  
'BA' > 'AB'
```

```
Out[9]: True
```

Note: Upper Case Letters have different ASCII code than Lower Case Letters, which means the comparison between the letters in python is case-sensitive.

Branching

Branching allows us to run different statements for different inputs. It is helpful to think of an **if statement** as a locked room, if the statement is **True** we can enter the room and your program will run some predefined tasks, but if the statement is **False** the program will ignore the task.

For example, consider the blue rectangle representing an ACDC concert. If the individual is older than 18, they can enter the ACDC concert. If they are 18 or younger than 18 they cannot enter the concert.

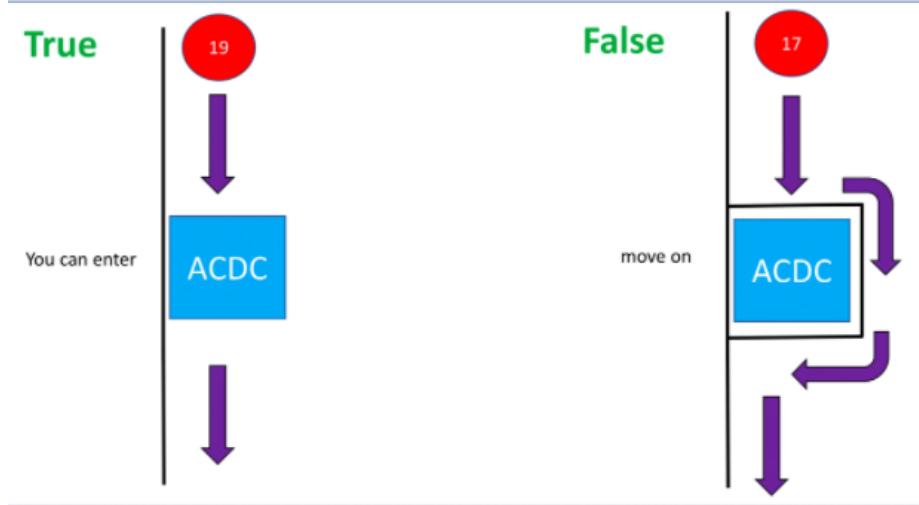
Use the condition statements learned before as the conditions need to be checked in the **if statement**. The syntax is as simple as `if condition statement :`, which contains a word if, any condition statement, and a colon at the end. Start your tasks which need to be executed under this condition in a new line with an indent. The lines of code after the colon and with an indent will only be executed when the **if statement** is **True**. The tasks will end when the line of code does not contain the indent.

In the case below, the tasks executed `print("you can enter")` only occurs if the variable `age` is greater than 18 is a True case because this line of code has the indent. However, the execution of `print("move on")` will not be influenced by the if statement.

```
In [10]: # If statement example  
  
age = 19  
age = 18  
  
#expression that can be true or false  
if age > 18:  
  
    #within an indent, we have the expression that is run if the condition is true  
    print("you can enter")  
  
    #The statements after the if statement will run regardless if the condition is true or false  
    print("move on")  
  
you can enter  
move on
```

Try uncommenting the age variable

It is helpful to use the following diagram to illustrate the process. On the left side, we see what happens when the condition is **True**. The person enters the ACDC concert representing the code in the indent being executed; they then move on. On the right side, we see what happens when the condition is **False**; the person is not granted access, and the person moves on. In this case, the segment of code in the indent does not run, but the rest of the statements are run.



The else statement runs a block of code if none of the conditions are **True** before this else statement. Let's use the ACDC concert analogy again. If the user is 17 they cannot go to the ACDC concert, but they can go to the Meatloaf concert. The syntax of the else statement is similar as the syntax of the if statement, as else :: Notice that, there is no condition statement for else. Try changing the values of age to see what happens:

```
In [11]: # Else statement example

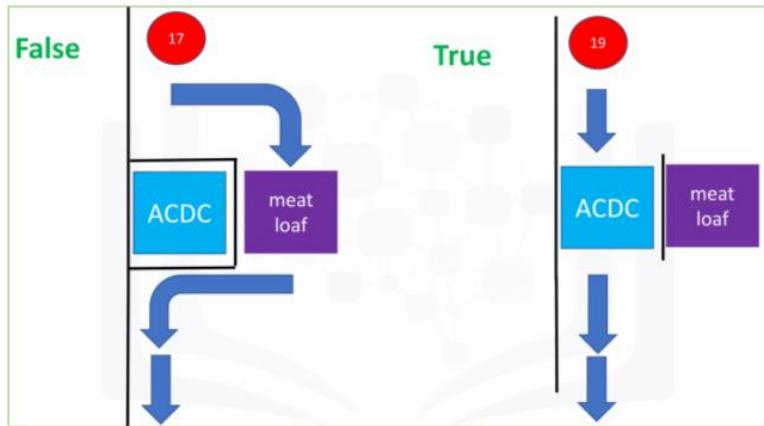
age = 18
# age = 19

if age > 18:
    print("you can enter" )
else:
    print("go see Meat Loaf" )

print("move on")

go see Meat Loaf
move on
```

The process is demonstrated below, where each of the possibilities is illustrated on each side of the image. On the left is the case where the age is 17, we set the variable age to 17, and this corresponds to the individual attending the Meatloaf concert. The right portion shows what happens when the individual is over 18, in this case 19, and the individual is granted access to the concert.



The `elif` statement, short for else if, allows us to check additional conditions if the condition statements before it are **False**. If the condition for the `elif` statement is **True**, the alternate expressions will be run. Consider the concert example, where if the individual is 18 they will go to the Pink Floyd concert instead of attending the ACDC or Meat-loaf concert. The person of 18 years of age enters the area, and as they are not older than 18 they can not see ACDC, but as they are 18 years of age, they attend Pink Floyd. After seeing Pink Floyd, they move on. The syntax of the `elif` statement is similar in that we merely change the `if` in `if` statement to `elif`.

```
In [12]: # Elif statement example

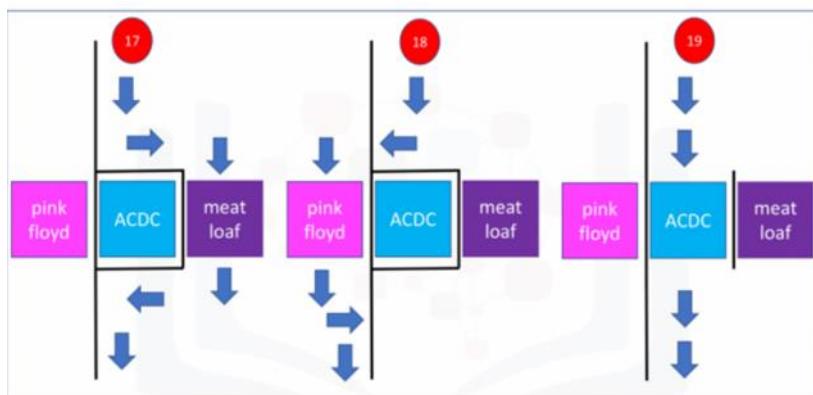
age = 18

if age > 18:
    print("you can enter" )
elif age == 18:
    print("go see Pink Floyd")
else:
    print("go see Meat Loaf" )

print("move on")
```

go see Pink Floyd
move on

The three combinations are shown in the figure below. The left-most region shows what happens when the individual is less than 18 years of age. The central component shows when the individual is exactly 18. The rightmost shows when the individual is over 18.



Look at the following code:

```
In [14]: # Condition statement example

album_year = 1983
# album_year = 1970

if album_year > 1980:
    print("Album year is greater than 1980")

print('do something..')
```

```
Album year is greater than 1980
do something..
```

Feel free to change `album_year` value to other values -- you'll see that the result changes!

Notice that the code in the above **indented** block will only be executed if the results are **True**.

As before, we can add an else block to the if block. The code in the else block will only be executed if the result is **False**.

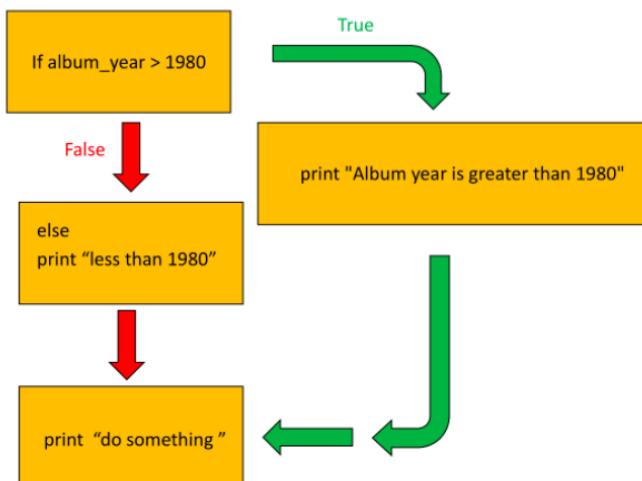
Syntax:

```
if (condition):
    # do something
```

```
else:
```

```
    # do something else
```

If the condition in the `if` statement is **False**, the statement after the `else` block will execute. This is demonstrated in the figure:



```
In [15]: # Condition statement example

album_year = 1983
#album_year = 1970

if album_year > 1980:
    print("Album year is greater than 1980")
else:
    print("less than 1980")

print('do something..')

Album year is greater than 1980
do something..
```

Feel free to change the `album_year` value to other values -- you'll see that the result changes based on it!

Logical operators

Sometimes you want to check more than one condition at once. For example, you might want to check if one condition and another condition is **True**. Logical operators allow you to combine or modify conditions.

- `and`
- `or`
- `not`

These operators are summarized for two variables using the following truth tables:

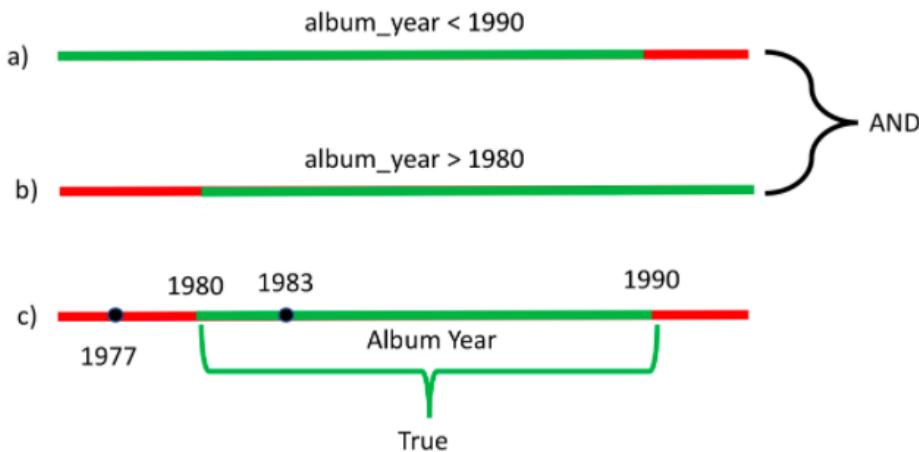
| A | B | A & B |
|-------|-------|-------|
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

| A | B | A or B |
|-------|-------|--------|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

| A | A! |
|-------|-------|
| False | True |
| True | False |

The and statement is only **True** when both conditions are true. The or statement is true if one condition is **True**. The not statement outputs the opposite truth value.

Let's see how to determine if an album was released after 1979 (1979 is not included) and before 1990 (1990 is not included). The time periods between 1980 and 1989 satisfy this condition. This is demonstrated in the figure below. The green on lines **a** and **b** represents periods where the statement is **True**. The green on line **c** represents where both conditions are **True**, this corresponds to where the green regions overlap.



The block of code to perform this check is given by:

```
In [16]: # Condition statement example
          album_year = 1980

          if(album_year > 1979) and (album_year < 1990):
              print ("Album year was in between 1980 and 1989")

              print("")
              print("Do Stuff..")
```

Album year was in between 1980 and 1989
Do Stuff..

To determine if an album was released before 1980 (~ - 1979) or after 1989 (1990 - ~), an **or** statement can be used. Periods before 1980 (~ - 1979) or after 1989 (1990 - ~) satisfy this condition. This is demonstrated in the following figure, the color green in **a** and **b** represents periods where the statement is true. The color green in **c** represents where at least one of the conditions are true.



The block of code to perform this check is given by:

```
In [17]: # Condition statement example

album_year = 1990

if(album_year < 1980) or (album_year > 1989):
    print ("Album was not made in the 1980's")
else:
    print("The Album was made in the 1980's ")

Album was not made in the 1980's
```

The `not` statement checks if the statement is false:

```
In [18]: # Condition statement example

album_year = 1983

if not (album_year == '1984'):
    print ("Album year is not 1984")

Album year is not 1984
```

Quiz on Conditions

Write an if statement to determine if an album had a rating greater than 8. Test it using the rating for the album “**Back in Black**” that had a rating of 8.5. If the statement is true print “This album is Amazing!”

```
In [19]: # Write your code below and press Shift+Enter to execute
rating = 8.5

if (rating > 8):
    print("This album is Amazing!")

This album is Amazing!
```

Click here for the solution ``python rating = 8.5 if rating > 8: print ("This album is Amazing!") ``

Write an if-else statement that performs the following. If the rating is larger then eight print “this album is amazing”. If the rating is less than or equal to 8 print “this album is ok”.

In [21]:

```
# Write your code below and press Shift+Enter to execute
rating = 7.5

if (rating > 8):
    print("This album is amazing!")
else:
    print("This album is ok.")
```

This album is ok.

Click here for the solution ```python rating = 8.5 if rating > 8: print ("this album is amazing") else: print ("this album is ok")```

Write an if statement to determine if an album came out before 1980 or in the years: 1991 or 1993. If the condition is true print out the year the album came out.

In [24]:

```
# Write your code below and press Shift+Enter to execute
album_year = 1977

if (album_year < 1980) or (album_year == 1991) or (album_year == 1993):
    print("This album came out in ",album_year)
```

This album came out in 1977

Click here for the solution ```python album_year = 1979 if album_year < 1980 or album_year == 1991 or album_year == 1993: print ("This album came out in year %d" %album_year)```

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

Python Arithmetic Operators

Assume variable a holds 10 and variable b holds 20, then –

[Show Example ]

| Operator | Description | Example |
|---------------|---|--|
| + Addition | Adds values on either side of the operator. | $a + b = 30$ |
| - Subtraction | Subtracts right hand operand from left hand operand. | $a - b = -10$ |
| * | Multiplication | $a * b = 200$ |
| / Division | Divides left hand operand by right hand operand | $b / a = 2$ |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | $b \% a = 0$ |
| ** Exponent | Performs exponential (power) calculation on operators | $a^{**}b = 10 \text{ to the power } 20$ |
| // | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) | $9//2 = 4 \text{ and } 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0$ |

Python Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

Assume variable a holds 10 and variable b holds 20, then –

[Show Example ]

| Operator | Description | Example |
|-----------------------|---|---|
| <code>==</code> | If the values of two operands are equal, then the condition becomes true. | <code>(a == b)</code> is not true. |
| <code>!=</code> | If values of two operands are not equal, then condition becomes true. | <code>(a != b)</code> is true. |
| <code><></code> | If values of two operands are not equal, then condition becomes true. | <code>(a <> b)</code> is true. This is similar to <code>!=</code> operator. |
| <code>></code> | If the value of left operand is greater than the value of right operand, then condition becomes true. | <code>(a > b)</code> is not true. |
| <code><</code> | If the value of left operand is less than the value of right operand, then condition becomes true. | <code>(a < b)</code> is true. |
| <code>>=</code> | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | <code>(a >= b)</code> is not true. |
| <code><=</code> | If the value of left operand is less than or equal to the value of right operand, then condition becomes true. | <code>(a <= b)</code> is true. |

Python Assignment Operators

Assume variable a holds 10 and variable b holds 20, then –

[Show Example ↗]

| Operator | Description | Example |
|--------------------|--|---|
| = | Assigns values from right side operands to left side operand | c = a + b assigns value of a + b into c |
| += Add AND | It adds right operand to the left operand and assign the result to left operand | c += a is equivalent to c = c + a |
| -= Subtract AND | It subtracts right operand from the left operand and assign the result to left operand | c -= a is equivalent to c = c - a |
| *= Multiply AND | It multiplies right operand with the left operand and assign the result to left operand | c *= a is equivalent to c = c * a |
| /= Divide AND | It divides left operand with the right operand and assign the result to left operand | c /= a is equivalent to c = c / a |
| %= Modulus AND | It takes modulus using two operands and assign the result to left operand | c %= a is equivalent to c = c % a |
| **= Exponent AND | Performs exponential (power) calculation on operators and assign value to the left operand | c **= a is equivalent to c = c ** a |
| //= Floor Division | It performs floor division on operators and assign value to the left operand | c // a is equivalent to c = c // a |

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if $a = 60$; and $b = 13$; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands –

$a = 0011\ 1100$

$b = 0000\ 1101$

$a \& b = 0000\ 1100$

$a | b = 0011\ 1101$

$a ^ b = 0011\ 0001$

$\sim a = 1100\ 0011$

| Operator | Description | Example |
|-------------------------------|---|--|
| & Binary AND | Operator copies a bit to the result if it exists in both operands | $(a \& b)$ (means 0000 1100) |
| Binary OR | It copies a bit if it exists in either operand. | $(a b) = 61$ (means 0011 1101) |
| ^ Binary XOR | It copies the bit if it is set in one operand but not both. | $(a ^ b) = 49$ (means 0011 0001) |
| \sim Binary Ones Complement | It is unary and has the effect of 'flipping' bits. | $(\sim a) = -61$ (means 1100 0011 in 2's complement form due to a signed binary number.) |
| << Binary Left Shift | The left operand's value is moved left by the number of bits specified by the right operand. | $a << 2 = 240$ (means 1111 0000) |
| >> Binary Right Shift | The left operand's value is moved right by the number of bits specified by the right operand. | $a >> 2 = 15$ (means 0000 1111) |

Python Logical Operators

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then

[Show Example ]

| Operator | Description | Example |
|-----------------------|--|------------------------|
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

Python Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below –

[Show Example ↗]

| Operator | Description | Example |
|----------|---|--|
| in | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a 1 if x is a member of sequence y. |
| not in | Evaluates to true if it does not find a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below –

[Show Example ↗]

| Operator | Description | Example |
|----------|---|---|
| is | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here is results in 1 if id(x) equals id(y). |
| is not | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here is not results in 1 if id(x) is not equal to id(y). |

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

[Show Example ↗]

| Sr.No. | Operator & Description |
|--------|---|
| 1 | <code>**</code> Exponentiation (raise to the power) |
| 2 | <code>~ + -</code> Complement, unary plus and minus (method names for the last two are <code>+@</code> and <code>-@</code>) |
| 3 | <code>* / % //</code> Multiply, divide, modulo and floor division |
| 4 | <code>+ -</code> Addition and subtraction |
| 5 | <code>>> <<</code> Right and left bitwise shift |

| | |
|----|---|
| 6 | & Bitwise 'AND' |
| 7 | ^ Bitwise exclusive 'OR' and regular 'OR' |
| 8 | <= < > >= Comparison operators |
| 9 | <> == != Equality operators |
| 10 | = %= /= //= -= += *= **= Assignment operators |
| 11 | is is not Identity operators |
| 12 | in not in Membership operators |
| 13 | not or and Logical operators |



Module Introduction and Learning Objectives

[Bookmark this page](#)

This module discusses Python fundamentals and begins with the concepts of conditions and branching. Continue through the module and learn how to implement loops to iterate over sequences, create functions to perform a specific task, perform exception handling to catch errors, and how classes are needed to create objects.

Learning Objectives

In this lesson you will learn about:

- Classify conditions and branching by identifying structured scenarios with outputs.
- Understand loops by using visual examples and comparing them to tuples and lists.
- Understand functions by building a function using inputs and outputs.
- Explain objects and classes by identifying data types and creating a class.

Conditions and Branching

Comparison operations compares some value or operand. Then based on some condition, they produce a Boolean. Let's say we assign a value of `a` to six. We can use the equality operator denoted with two equal signs to determine if two values are equal. In this case, if seven is equal to six.

In this case, as six is not equal to seven, the result is false. If we performed an equality test for the value six, the two values would be equal. As a result, we would get a true. Consider the following equality comparison operator:

If the value of the left operand, in this case, the variable `i` is greater than the value of the right operand, in this case five, the condition becomes true or else we get a false. Let's display some values for `i` on the left. Let's see the value is greater than five in green and the rest in red. If we set `i` equal to six, we see that six is larger than five and as a result, we get a true. We can also apply the same operations to floats. If we modify the operator as follows, if the left operand `i` is greater than or equal to the value of the right operand, in this case five, then the condition becomes true. In this case, we include the value of five in the number line and the color changes to green accordingly. If we set the value of `i` equal to five, the operand will produce a true. If we set the value of `i` to two, we would get a false because two is less than five. We can change the inequality if the value of the left operand, in this case, `i` is less than the value of the right operand, in this case, six.

Then condition becomes true. Again, we can represent this with a colored number line. The areas where the inequality is true are marked in green and red where the inequality is false. If the value for *i* is set to two, the result is a true. As two is less than six. The inequality test uses an explanation mark preceding the equal sign. If two operands are not equal, then the condition becomes true.

We can use a number line. When the condition is true, the corresponding numbers are marked in green and red for where the condition is false.

If we set *i* equal to two, the operator is true as two is not equal to six. We compare strings as well.

Comparing ACDC and Michael Jackson using the equality test, we get a false, as the strings are not the same.

Using the inequality test, we get a true, as the strings are different. See the Lapps for more examples.

Branching allows us to run different statements for a different input. It's helpful to think of an if statement as a locked room. If this statement is true, you can enter the room and your program can run some predefined task. If the statement is false, your program will skip the task. For example, consider the blue rectangle representing an ACDC concert. If the individual is 18 or older, they can enter the ACDC concert. If they are under the age of 18, they cannot enter the concert. Individual proceeds to the concert their age is 17, therefore, they are not granted access to the concert and they must move on. If the individual is 19, the condition is true. They can enter the concert then they can move on. This is the syntax of the if statement from our previous example. We have the if statement. We have the expression that can be true or false. The brackets are not necessary. We have a colon. Within an indent, we have the expression that is run if the condition is true. The statements after the if statement will run regardless if the condition is true or false. For the case where the age is 17, we set the value of the variable age to 17. We check the if statement, the statement is false. Therefore the program will not execute the statement to print, "you will enter". In this case, it will just print "move on". For the case where the age is 19, we set the value of the variable age to 19. We check the if statement. The statement is true.

Therefore, the program will execute the statement to print "you will enter". Then it will just print "move on".

The else statement will run a different block of code if the same condition is false. Let's use the ACDC concert analogy again. If the user is 17, they cannot go to the ACDC concert but they can go to the Meat Loaf concert represented by the purple square. If the individual is 19, the condition is true, they can enter the ACDC concert then they can move on as before. The syntax of the else statement is similar. We simply append the statement else.

We then add the expression we would like to execute with an indent. For the case where the age is 17, we set the value of the variable age to 17. We check the if statement, the statement is false.

Therefore, we progress to the else statement. We run the statement in the indent.

This corresponds to the individual attending the Meat Loaf concert. The program will then continue running.

For the case where the age is 19, we set the value of the variable age to 19. We check the if statement, the statement is true. Therefore, the program will execute the statement to print "you will enter".

The program skips the expressions in the else statement and continues to run the rest of the expressions.

The elif statement, short for else if, allows us to check additional conditions if the preceding condition is false.

If the condition is true, the alternate expressions will be run. Consider the concert example, if the individual is 18, they will go to the Pink Floyd concert instead of attending the ACDC or Meat Loaf concerts.

The person of 18 years of age enters the area as they are not over 19 years of age. They cannot see ACDC but as their 18 years, they attend Pink Floyd. After seeing Pink Floyd, they move on. The syntax of the elif statement is similar. We simply add the statement elif with the condition. We, then add the expression we would like to execute if the statement is true with an indent. Let's illustrate the code on the left. An 18 year old enters.

They are not older than 18 years of age. Therefore, the condition is false. So the condition of the elif statement is checked. The condition is true. So then we would print "go see Pink Floyd". Then we would move on as before.

If the variable age was 17, the statement "go see Meat Loaf" would print. Similarly, if the age was greater than 18, the statement "you can enter" would print. Check the Lapps for more examples. Now let's take a look at logic operators. Logic operations take Boolean values and produce different Boolean values. The first operation is the not operator. If the input is true, the result is a false. Similarly, if the input is false, the result is a true. Let A and B represent Boolean variables. The OR operator takes in the two values and produces a new Boolean value.

We can use this table to represent the different values. The first column represents the possible values of A. The second column represents the possible values of B. The final column represents the result of applying the OR operation. We see the OR operator only produces a false if all the Boolean values are false. The following lines of code will print out: "This album was made in the 70s' or 90's", if the variable album year does not fall in the 80s. Let's see what happens when we set the album year to 1990. The colored number line is green when the condition is true and red when the condition is false. In this case, the condition is false. Examining the second condition, we see that 1990 is greater than 1989. So the condition is true. We can verify by examining the corresponding second number line. In the final number line, the green region indicates, where the area is true. This region corresponds to where at least one statement is true. We see that 1990 falls in the area. Therefore, we execute the statement.

Let A and B represent Boolean variables.

The AND operator takes in the two values and produces a new Boolean value.

We can use this table to represent the different values. The first column represents the possible values of A.

The second column represents the possible values of B. The final column represents the result of applying the AND operation.

We see the OR operator only produces a true if all the Boolean values are true. The following lines of code will print out "This album was made in the 80's" if the variable album year is between 1980 and 1989. Let's see what happens when we set the album year to 1983. As before, we can use the colored number line to examine where the condition is true. In this case, 1983 is larger than 1980, so the condition is true.

Examining the second condition, we see that 1990 is greater than 1983. So, this condition is also true.

We can verify by examining the corresponding second number line. In the final number line, the green region indicates where the area is true. Similarly, this region corresponds to where both statements are true. We see that 1983 falls in the area. Therefore, we execute the statement.

Branching allows us to run different statements for different inputs.



Comparison Operators



`==` determines if two values are equal

Comparison Operators

6==7

False

a=6

a==7

False

Comparison Operators

6==6

True

a=6

a==6

True

i=6

i>5

True



True

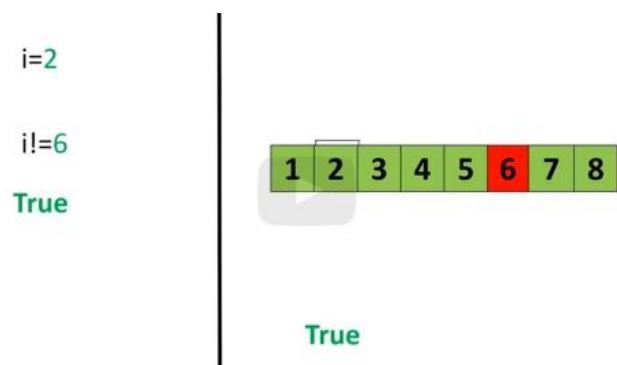
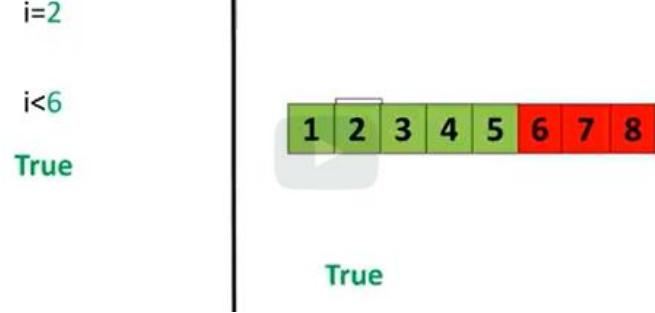
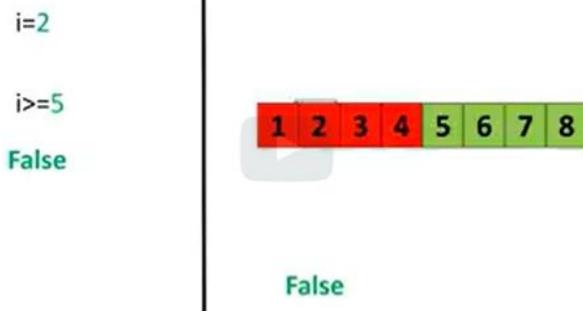
i=5

i>=5

True



True



"AC/DC" == "Michael Jackson"

False

"AC/DC" != "Michael Jackson"

True

Branching

The if Statement

False

ACDC

17

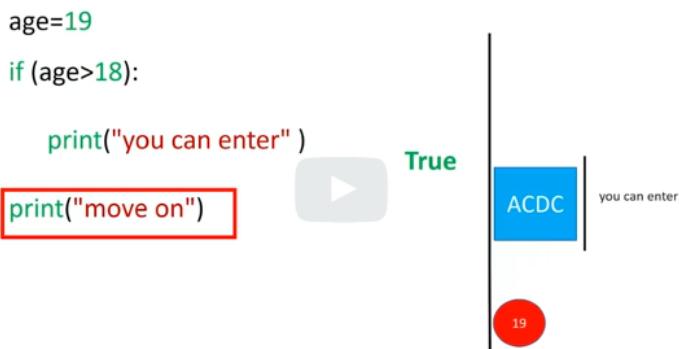
The if Statement

True

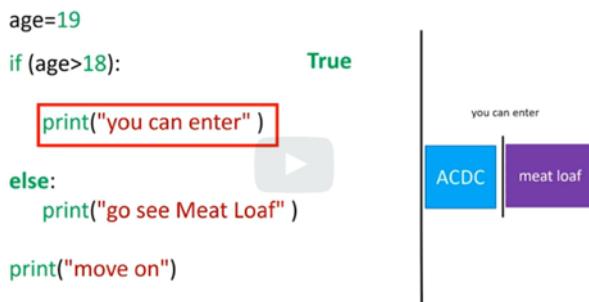
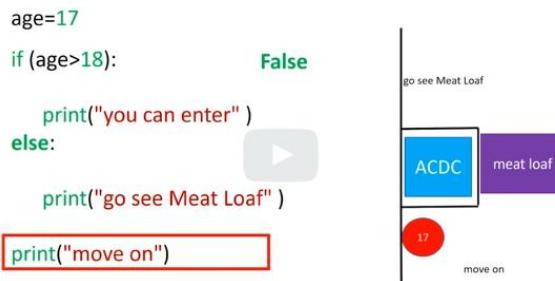
ACDC

19

Syntax



ELSE statement



The elif Statement

```
age=18
```

```
if (age>18):
```

```
    print("you can enter" )
```

```
elif(age==18):
```

```
    print("go see Pink Floyd" )
```

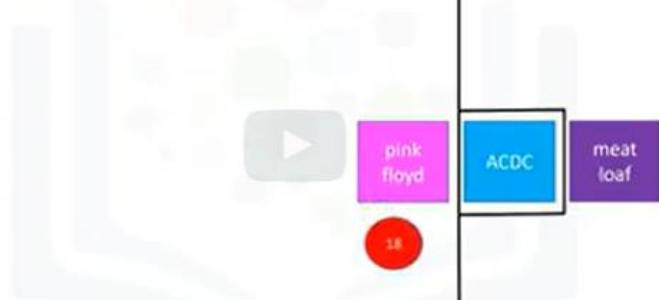
```
else:
```

```
    print("go see Meat Loaf" )
```

```
print("move on")
```

True

go see Pink Floyd



LOGIC OPERATORS

Logic Operators

LOGIC
OPERATORS

Boolean

Logic Operators

True

not

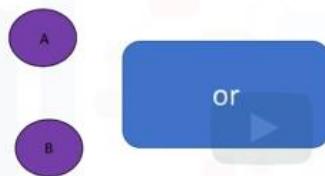
not(True)

Logic Operators

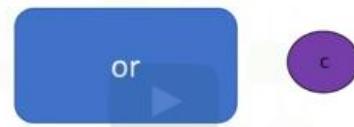
Logic Operators



Logic Operators: OR



Logic Operators: OR



Logic Operators: OR

| A | B | A or B |
|-------|-------|--------|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

```
album_year = 1990
```



```
If (album_year < 1980) or (album_year > 1989):
```

```
    print ("The Album was made in the 70 's or 90's")
```

```
else:
```

```
    print("The Album was made in the 1980's ")
```

```
album_year = 1990
```



```
If (album_year < 1980) or [album_year > 1989]:
```

```
    print ("The Album was made in the 70 's or 90's")
```

```
else:
```

```
    print("The Album was made in the 1980's ")
```

```
album_year = 1990
```



```
If (album_year < 1980) or (album_year > 1989):
```

```
    print ("The Album was made in the 70 's or 90's")
```

```
else:
```

```
    print("The Album was made in the 1980's ")
```



True

True

Logic Operators: AND



Logic Operators: AND

Logic Operators: AND

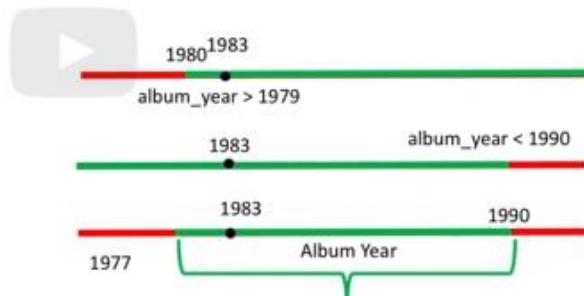
| A | B | A & B (AND) |
|-------|-------|-------------|
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

```
album_year = 1983
```

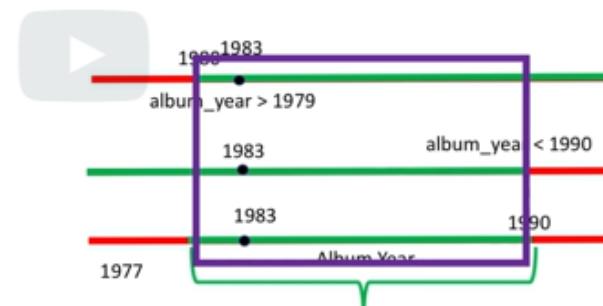
```
if(album_year > 1979) and (album_year < 1990):  
    print ("This album was made in the 80's ")
```



```
album_year = 1983  
  
if(album_year > 1979) and (album_year < 1990):  
  
    print ("This album was made in the 80's ")
```



```
album_year = 1983  
  
if(album_year > 1979) and (album_year < 1990):  
  
    print ("This album was made in the 80's ")
```



Practice Quiz: Python Programming Fundamentals

 Bookmark this page

Question 1

0/1 point (ungraded)

What is the result of the following: `1=2`

True

SyntaxError:can't assign to literal

False



Answer

Incorrect: Incorrect, a comparison operator is denoted as follows: ==

 Submit

You have used 2 of 2 attempts

Question 2

1/1 point (ungraded)

What is the output of the following code segment:

`i=6`

`i<5`

SyntaxError: can't assign to literal

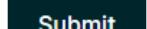
False

True



Answer

Correct: Correct

 Submit

You have used 1 of 2 attempts

 Correct (1/1 point)

Question 3

1/1 point (ungraded)

What is the result of the following: `5!=5`

True

False

**Answer**

Correct: Correct, this is the inequality operator.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 4

1/1 point (ungraded)

What is the output of the following code segment:`'a'=='A'`

True

False

**Answer**

Correct: Correct, the equality operator is case sensitive.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 5

1/1 point (ungraded)

In the video, if `age=18` what would be the result.

You can enter

Move on

**Answer**

Correct: Correct

Submit

You have used 2 of 2 attempts

✓ Correct (1/1 point)

Question 6

1/1 point (ungraded)

In the video what would be the result if we set the variable age as follows: **age= -10**

you can enter
move on

go see Meat Loaf
move on



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 7

1/1 point (ungraded)

What is the result of the following: **True or False**

False

True, an **or** statement is only False if all the Boolean values are False.



Answer

Correct: Correct, an or statement is only False if all the Boolean values are False.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

<https://github.com/brendensong/IBM-Data-Science-Professional-Certificate/wiki/4.1.Python-Basics>

Loops

In this video we will cover Loops in particular for loops and while loops. We will use many visual examples in this video.

See the labs for examples with data. Before we talk about loops,

let's go over the range function. The range function outputs an ordered sequence as a list I.

If the input is a positive integer, the output is a sequence. The sequence contains the same number of elements as the input but starts at zero.

For example, if the input is three the output is the sequence zero, one, two. If the range function has two inputs where the first input is smaller than the second input, the output is a sequence that starts at the first input.

Then the sequence iterates up to but not including the second number. For the input 10 and 15 we get the following sequence. See the labs for more capabilities of the range function.

Please note, if you use Python three, the range function will not generate a list explicitly like in Python two.

In this section, we will cover for loops. We will focus on lists, but many of the procedures can be used on tuples.

Loops perform a task over and over. Consider the group of colored squares. Let's say we would like to replace each colored square with a white square. Let's give each square a number to make things a little easier and refer to all the group of squares as squares. If we wanted to tell someone to replace square zero with a white square, we would say equals replace square zero with a white square or we can say four squares zero in squares square zero equals white square. Similarly, for the next square we can say for square one in squares, square one equals white square. For the next square we can say for square two in squares, square two equals white square. We repeat the process for each square. The only thing that changes is the index of the square we are referring to. If we're going to perform a similar task in Python we cannot use actual squares.

So let's use a list to represent the boxes. Each element in the list is a string representing the color.

We want to change the name of the color in each element to white. Each element in the list has the following index. This is a syntax to perform a loop in Python. Notice the indent, the range function generates a list.

The code will simply repeat everything in the indent five times. If you were to change the value to six it would do it 6 times. However, the value of I is incremented by one each time. In this segment we change the I element of the list to the string white.

The value of I is set to zero. Each iteration of the loop starts at the beginning of the indent.

We then run everything in the indent. The first element in the list is set to white. We then go to the start of the indent, we progress down each line. When we reach the line to change the value of the list,

we set the value of index one to white. The value of I increases by one. We repeat the process for index two.

The process continues for the next index, until we've reached the final element. We can also iterate through a list or tuple directly in python, we do not even need to use indices.

Here is the list squares. Each iteration of the list we pass one element of the list squares to the variable square.

Lets display the value of the variable square on this section. For the first iteration, the value of square is red, we then start the second iteration. For the second iteration, the value of square is yellow. We then start the third iteration. For the final iteration, the value of square is green, a useful function for iterating data is enumerate.

It can be used to obtain the index and the element in the list. Let's use the box analogy with the numbers representing the index of each square. This is the syntax to iterate through a list and provide the index of each element.

We use the list squares and use the names of the colors to represent the colored squares. The argument of the function enumerate is the list. In this case squares the variable I is the index and the variable square is the corresponding element in the list. Let's use the left part of the screen to display the different values of the variable square and I for the various iterations of the loop. For the first iteration, the value of the variable is red corresponding to the zeroth index, and the value for I is zero for the second iteration.

The value of the variable square is yellow, and the value of I corresponds to its index i.e. 1. We repeat the process for the last index. While loops are similar to for loops but instead of executing a statement a set number of times a while loop will only run if a condition is met. Let's say we would like to copy all the orange squares from the list squares to the list New squares. But we would like to stop if we encounter a non-orange square.

We don't know the value of the squares beforehand. We would simply continue the process while the square is orange or see if the square equals orange. If not, we would stop. For the first example, we would check if the square was orange. It satisfies the conditions so we would copy the square.

We repeat the process for the second square. The condition is met. So we copy the square. In the next iteration, we encounter a purple square. The condition is not met. So we stop the process.

This is essentially what a while loop does. Let's use the figure on the left to represent the code. We will use a list with the names of the color to represent the different squares. We create an empty list of new squares. In reality the list is of indeterminate size. We start the index at zero the while statement will repeatedly execute the statements within the indent until the condition inside the bracket is false.

We append the value of the first element of the list squares to the list new squares.

We increase the value of I by one.

We append the value of the second element of the list squares to the list new squares.

We increment the value of I.

Now the value in the array squares is purple;

therefore, the condition for the while statement is false and we exit the loop.

range(N)



$[0, \dots, N - 1]$

range(3)



$[0, 1, 2]$

range(10,15)

range(10,15)



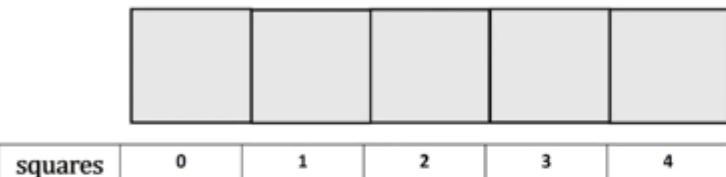
$[10, 11, 12, 13, 14]$

if the 1st index is lower than the 2nd index, start with the lower then last is minus 1

for loops



Replace square 0 with a white square



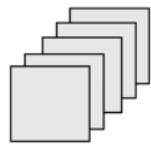
For square 0 in squares, square 0 = white square
 For square 1 in squares, square 1 = white square
 For square 2 in squares, square 2 = white square
 For square 3 in squares, square 3 = white square
 For square 4 in squares, square 4 = white square

| | | | | | |
|---------|-----|--------|-------|--------|------|
| | red | yellow | green | purple | blue |
| squares | 0 | 1 | 2 | 3 | 4 |

squares=[“red”, “yellow”, “green”, “purple”, “blue”]

for i in range(0,5):

squares[i]=“white”



| | | | | | |
|---------|-------|--------|-------|--------|------|
| squares | white | yellow | green | purple | blue |
| | 0 | 1 | 2 | 3 | 4 |

squares=[“red”, “yellow”, “green”, “purple”, “blue”]

for i in range(0,5):

squares[1]=“white”



and so on:



| | | | | | |
|---------|-------|-------|-------|--------|------|
| squares | white | white | white | purple | blue |
| | 0 | 1 | 2 | 3 | 4 |

squares=[“red”, “yellow”, “green”, “purple”, “blue”]

for i in range(0,5):

squares[4]=“white”



and so on until:



squares=[“red”, “yellow”, “green”]

for square in squares:

square



square
red



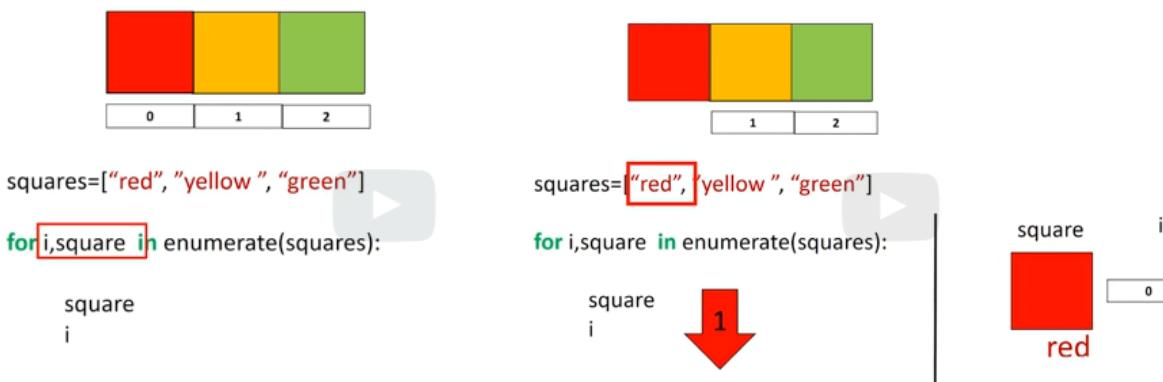
squares=[“red”, “yellow”, “green”]

for square in squares:

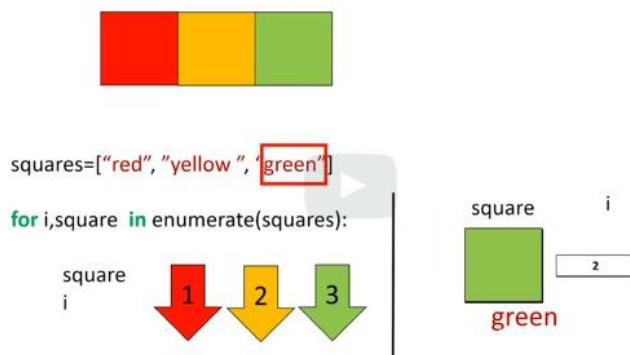
square



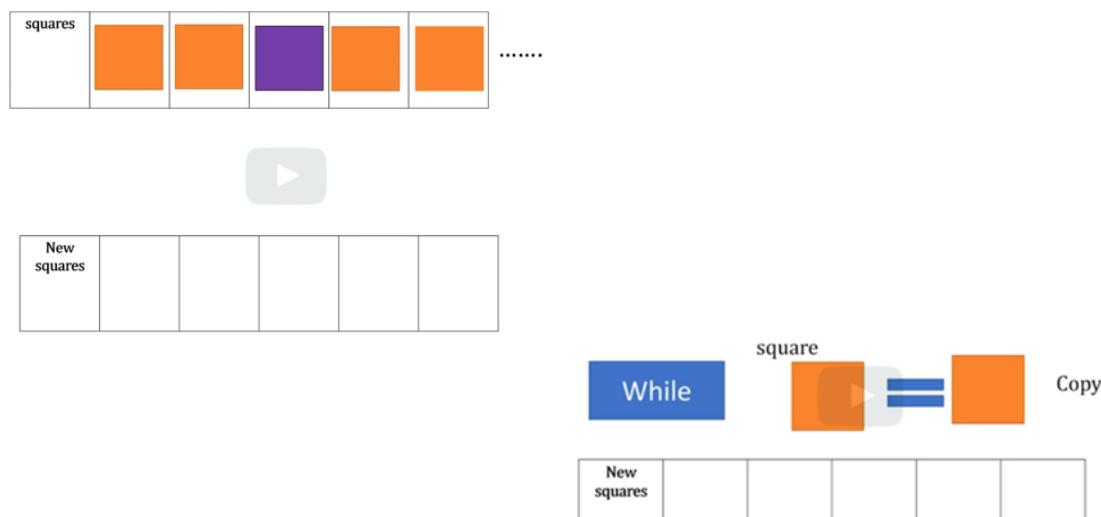
square
green

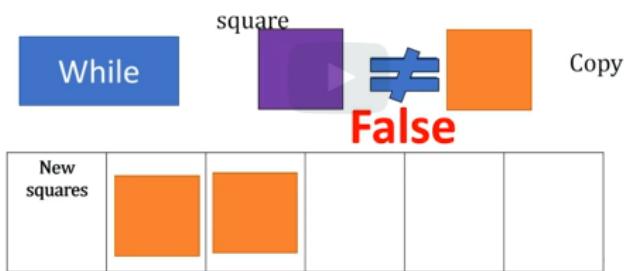
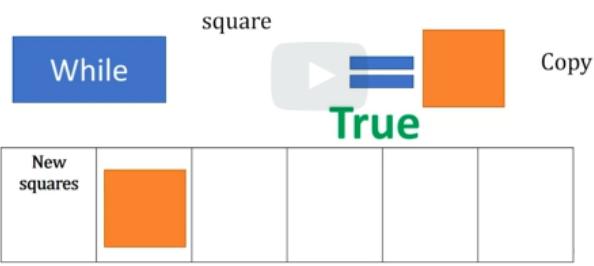


and so on:

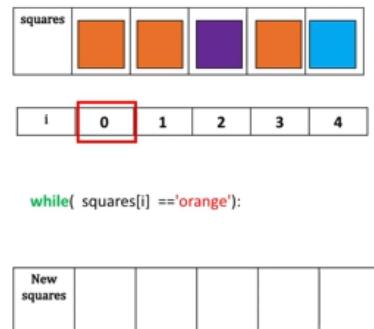


while loops





```
squares=['orange','orange','purple','orange','blue']
Newsquares=[]
i=0
while(squares[i]=='orange'):
    Newsquares.append(squares[i])
    i=i+1
```



```
squares=['orange','orange','purple','orange','blue']
```

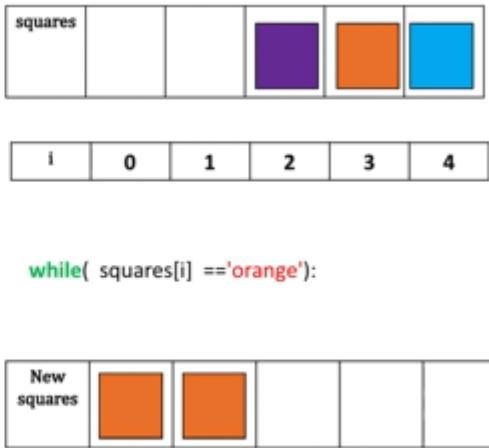
```
Newsquares=[]
```

```
i=0
```

```
while(squares[i]=='orange'):
```

```
    Newsquares.append(squares[i])
```

```
i=i+1
```



```
squares=['orange','orange','purple','orange','blue']
```

```
Newsquares=[]
```

```
i=0
```

```
while(squares[i]=='orange'):
```

```
    Newsquares.append(squares[i])
```

```
    i=i+1
```



| | | | | | | |
|---------|--|--|--|--|--------|------|
| squares | | | | | | |
| | | | | | orange | blue |

| | | | | | |
|---|---|---|---|---|---|
| i | 0 | 1 | 2 | 3 | 4 |
| | | | 2 | | |

```
while( squares[i] =='orange'):
```

| | | | | | | |
|-------------|--------|--------|--|--|--|--|
| New squares | | | | | | |
| | orange | orange | | | | |



Loops in Python

Estimated time needed: **20** minutes

Objectives

After completing this lab you will be able to:

- work with the loop statements in Python, including for-loop and while-loop.

Loops in Python

Welcome! This notebook will teach you about the loops in the Python Programming Language. By the end of this lab, you'll know how to use the loop statements in Python, including for loop, and while loop.

Table of Contents

- Loops
 - Range
 - What is `for` loop?
 - What is `while` loop?
- Quiz on Loops

Loops

Range

Sometimes, you might want to repeat a given operation many times. Repeated executions like this are performed by **loops**. We will look at two types of loops, for loops and while loops.

Before we discuss loops lets discuss the range object. It is helpful to think of the range object as an ordered list. For now, let's look at the simplest case. If we would like to generate an object that contains elements ordered from 0 to 2 we simply use the following command:

```
In [1]: # Use the range
```

```
range(3)
```

```
Out[1]: range(0, 3)
```

range(3)



range(0,3)

NOTE: While in Python 2.x it returned a list as seen in video lessons, in 3.x it returns a range object.

What is for loop?

The for loop enables you to execute a code block multiple times. For example, you would use this if you would like to print out every element in a list.

Let's try to use a for loop to print all the years presented in the list dates:

This can be done as follows:

```
In [2]: # For Loop example
```

```
dates = [1982,1980,1973]
N = len(dates)

for i in range(N):
    print(dates[i])
```

```
1982
1980
1973
```

The code in the indent is executed N times, each time the value of i is increased by 1 for every execution. The statement executed is to print out the value in the list at index i as shown here:

```

for i in range(N):
    print(dates[i])

Dates=[1982,1980,1973]
↑ ↑ ↑

i=0          i=0
print(dates[0])      print(1982)

i=1          i=1
print(dates[1])      print(1980)

i=2          i=2
print(dates[1])      print(1973)

```

In this example we can print out a sequence of numbers from 0 to 7:

In [3]:

```

# Example of for Loop

for i in range(0, 8):
    print(i)

```

```

0
1
2
3
4
5
6
7

```

In Python we can directly access the elements in the list as follows:

In [4]:

```

# Example of for Loop, loop through list

for year in dates:
    print(year)

```

```

1982
1980
1973

```

For each iteration, the value of the variable years behaves like the value of dates[i] in the first example:

```

for year in dates:
    print(year)

Dates=[1982,1980,1973]
    ↑
    year
i=0          i=0
print(year)   print(1982)

i=1          i=1
print(year)   print(1980)

i=2          i=2
print(year)   print(1973)

```

We can change the elements in a list:

```

In [5]: # Use for Loop to change the elements in list

squares = ['red', 'yellow', 'green', 'purple', 'blue']

for i in range(0, 5):
    print("Before square ", i, 'is', squares[i])
    squares[i] = 'white'
    print("After square ", i, 'is', squares[i])

Before square  0 is red
After square  0 is white
Before square  1 is yellow
After square  1 is white
Before square  2 is green
After square  2 is white
Before square  3 is purple
After square  3 is white
Before square  4 is blue
After square  4 is white

```

We can access the index and the elements of a list as follows:

```
In [6]: # Loop through the list and iterate on both index and element value

squares=['red', 'yellow', 'green', 'purple', 'blue']

for i, square in enumerate(squares):
    print(i, square)

0 red
1 yellow
2 green
3 purple
4 blue
```

What is while loop?

As you can see, the for loop is used for a controlled flow of repetition. However, what if we don't know when we want to stop the loop? What if we want to keep executing a code block until a certain condition is met? The while loop exists as a tool for repeated execution based on a condition. The code block will keep being executed until the given logical condition returns a **False** boolean value.

Let's say we would like to iterate through list dates and stop at the year 1973, then print out the number of iterations. This can be done with the following block of code:

```
In [8]: # While Loop Example

dates = [1982, 1980, 1973, 2000]

i = 0
year = dates[0]

while(year != 1973):
    print(year)
    i = i + 1
    year = dates[i]

print("It took ", i , "repetitions to get out of loop.")

1982
1980
It took 2 repetitions to get out of loop.
```

A while loop iterates merely until the condition in the argument is not met, as shown in the following figure:

```

albums = 250
total_albums = 0
i=0;
while( year!=1973):    True
    year=dates[i]
    i=i+1
    print(year)

print("it took",i, "outloop")

```

| | |
|---------------|---------------|
| i=0 | i=1 |
| year=dates[0] | year=dates[0] |
| i=0+1 | i=1+1 |
| print(1982) | print(1980) |

```

albums = 250
total_albums = 0
i=0;
while( year!=1973):    False
    year=dates[i]
    i=i+1
    print(year)

print("it took",i, "outloop")

```

| | | |
|---------------|---------------|---------------|
| i=0 | i=1 | i=2 |
| year=dates[0] | year=dates[0] | year=dates[0] |
| i=0+1 | i=1+1 | i=2+1 |
| print(1982) | print(1980) | print(1973) |

Quiz on Loops

Write a for loop the prints out all the element between **-5** and **5** using the range function.

In [10]:

```
# Write your code below and press Shift+Enter to execute
for i in range(-5, 6):
    print(i)
```

```

-5
-4
-3
-2
-1
0
1
2
3
4
5

```

Click here for the solution ```python for i in range(-5, 6): print(i) ````

Print the elements of the following list: Genres=['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop'] Make sure you follow Python conventions.

In [15]:

```
# Write your code below and press Shift+Enter to execute
Genres=['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']

for Genre in Genres:
    print(Genre)

# N = len(Genres)

# for i in range(N):
#     print(Genres[i])
```

```
rock
R&B
Soundtrack
R&B
soul
pop
```

Click here for the solution `python Genres = ['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop'] for Genre in Genres:
print(Genre)`

Write a for loop that prints out the following list: squares=['red', 'yellow', 'green', 'purple', 'blue']

In [17]:

```
# Write your code below and press Shift+Enter to execute
squares=['red', 'yellow', 'green', 'purple', 'blue']

for square in squares:
    print(square)

# Using while loop
# i = 0
# N = len(squares)

# while (i < N):
#     print(squares[i])
#     i = i + 1
```

```
red
yellow
green
purple
blue
```

Click here for the solution `python squares=['red', 'yellow', 'green', 'purple', 'blue'] for square in squares:
print(square)`

Write a while loop to display the values of the Rating of an album playlist stored in the list PlayListRatings. If the score is less than 6, exit the loop. The list PlayListRatings is given by: PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]

```
In [26]: # Write your code below and press Shift+Enter to execute
PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]

i = 0
N = len(PlayListRatings)

while (i < N) and (PlayListRatings[i] > 5):
    print(PlayListRatings[i])
    i = i + 1
```

```
10
9.5
10
8
7.5
```

Click here for the solution ````python PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10] i = 1 Rating = PlayListRatings[0] while(i < len(PlayListRatings) and Rating >= 6): print(Rating) Rating = PlayListRatings[i] i = i + 1````

Write a while loop to copy the strings 'orange' of the list squares to the list new_squares. Stop and exit the loop if the value on the list is not 'orange':

```
In [29]: # Write your code below and press Shift+Enter to execute

squares = ['orange', 'orange', 'purple', 'blue ', 'orange']
new_squares = []

i = 0
N = len(squares)

while (i < N) and (squares[i] == 'orange'):
    new_squares.append(squares[i])
    i = i + 1

print(new_squares)
```

```
['orange', 'orange']
```

Click here for the solution ````python squares = ['orange', 'orange', 'purple', 'blue ', 'orange'] new_squares = [] i = 0 while(i < len(squares) and squares[i] == 'orange'): new_squares.append(squares[i]) i = i + 1 print (new_squares)````

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

Quiz on Loops

Write a `for` loop that prints out all the elements between -5 and 5 using the range function.

```
[0]: # Write your code below and press Shift+Enter to execute
for i in range(-5, 6):
    print(i)
```

```
-5
-4
-3
-2
-1
0
1
2
3
4
5
```

► [Click here for the solution](#)

Print the elements of the following list: `Genres=['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']` Make sure you follow Python conventions.

```
[13]: # Write your code below and press Shift+Enter to execute
Genres=['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']

for Genre in Genres:
    print(Genre)
```

```
['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']
```

▼ [Click here for the solution](#)

```
Genres = ['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']
for Genre in Genres:
    print(Genre)
```

Write a for loop that prints out the following list: `squares=['red', 'yellow', 'green', 'purple', 'blue']`

```
[14]: # Write your code below and press Shift+Enter to execute
squares=['red','yellow','green','purple','blue']

for square in squares:
    print(square)

# Write your code below and press Shift+Enter to execute
```

red
yellow
green
purple
blue

▼ Click here for the solution

```
squares=['red', 'yellow', 'green', 'purple', 'blue']
for square in squares:
    print(square)
```

Write a while loop to display the values of the Rating of an album playlist stored in the list PlayListRatings. If the score is less than 6, exit the loop. The list PlayListRatings is given by: PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]

```
[15]: # Write your code below and press Shift+Enter to execute
PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]
i = 0
Rating = PlayListRatings[0]
while(i < len(PlayListRatings) and Rating >= 6):
    Rating = PlayListRatings[i]
    print(Rating)
    i = i + 1
```

10
9.5
10
8
7.5
5

▼ Click here for the solution

```
PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]
i = 0
Rating = PlayListRatings[0]
while(i < len(PlayListRatings) and Rating >= 6):
    Rating = PlayListRatings[i]
    print(Rating)
    i = i + 1
```

Write a while loop to copy the strings 'orange' of the list squares to the list new_squares. Stop and exit the loop if the value on the list is not 'orange'.

```
[16]: # Write your code below and press Shift+Enter to execute

squares = ['orange', 'orange', 'purple', 'blue ', 'orange']
new_squares = []

squares = ['orange', 'orange', 'purple', 'blue ', 'orange']
new_squares = []
i = 0
while(i < len(squares) and squares[i] == 'orange'):
    new_squares.append(squares[i])
    i = i + 1
print(new_squares)
-----
```

```
['orange', 'orange']
```

▼ Click here for the solution

```
squares = ['orange', 'orange', 'purple', 'blue ', 'orange']
new_squares = []
i = 0
while(i < len(squares) and squares[i] == 'orange'):
    new_squares.append(squares[i])
    i = i + 1
print (new_squares)
```

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python.



Loops in Python

Estimated time needed: **20** minutes

Objectives

After completing this lab you will be able to:

- work with the loop statements in Python, including for-loop and while-loop.

Loops in Python

Welcome! This notebook will teach you about the loops in the Python Programming Language. By the end of this lab, you'll know how to use the loop statements in Python, including for loop, and while loop.

Table of Contents

- Loops
 - Range
 - What is `for` loop?
 - What is `while` loop?
- Quiz on Loops

Loops

Range

Sometimes, you might want to repeat a given operation many times. Repeated executions like this are performed by **loops**. We will look at two types of loops, for loops and while loops.

Before we discuss loops lets discuss the range object. It is helpful to think of the range object as an ordered list. For now, let's look at the simplest case. If we would like to generate an object that contains elements ordered from 0 to 2 we simply use the following command:

```
In [1]: # Use the range  
  
range(3)
```

```
Out[1]: range(0, 3)
```

range(3)



range(0,3)

NOTE: While in Python 2.x it returned a list as seen in video lessons, in 3.x it returns a range object.

What is for loop?

The for loop enables you to execute a code block multiple times. For example, you would use this if you would like to print out every element in a list.

Let's try to use a for loop to print all the years presented in the list dates:

This can be done as follows:

```
In [2]: # For Loop example  
  
dates = [1982,1980,1973]  
N = len(dates)  
  
for i in range(N):  
    print(dates[i])
```

```
1982  
1980  
1973
```

The code in the indent is executed N times, each time the value of i is increased by 1 for every execution. The statement executed is to print out the value in the list at index i as shown here:

```
for i in range(N):  
    print(dates[i])  
  
Dates=[1982,1980,1973]  
      ↑↑↑  
  
i=0    print(dates[0])    i=0    print(1982)  
      ↓          ↓  
i=1    print(dates[1])    i=1    print(1980)  
      ↓          ↓  
i=2    print(dates[1])    i=2    print(1973)
```

In this example we can print out a sequence of numbers from 0 to 7:

In [3]:

```
# Example of for Loop

for i in range(0, 8):
    print(i)
```

```
0
1
2
3
4
5
6
7
```

In Python we can directly access the elements in the list as follows:

In [4]:

```
# Example of for Loop, Loop through List

for year in dates:
    print(year)
```

```
1982
1980
1973
```

For each iteration, the value of the variable years behaves like the value of dates[i] in the first example:

```
for year in dates:
```

```
    print(year)
```

```
Dates=[1982,1980,1973]
```



```
i=0
    print(year)
```

```
i=0
    print(1982)
```

```
i=1
    print(year)
```

```
i=1
    print(1980)
```

```
i=2
    print(year)
```

```
i=2
    print(1973)
```

We can change the elements in a list:

```
In [5]: # Use for loop to change the elements in list

squares = ['red', 'yellow', 'green', 'purple', 'blue']

for i in range(0, 5):
    print("Before square ", i, "is", squares[i])
    squares[i] = 'white'
    print("After square ", i, "is", squares[i])
```

```
Before square 0 is red
After square 0 is white
Before square 1 is yellow
After square 1 is white
Before square 2 is green
After square 2 is white
Before square 3 is purple
After square 3 is white
Before square 4 is blue
After square 4 is white
```

We can access the index and the elements of a list as follows:

```
In [6]: # Loop through the list and iterate on both index and element value

squares=['red', 'yellow', 'green', 'purple', 'blue']

for i, square in enumerate(squares):
    print(i, square)
```

```
0 red
1 yellow
2 green
3 purple
4 blue
```

What is while loop?

As you can see, the for loop is used for a controlled flow of repetition. However, what if we don't know when we want to stop the loop? What if we want to keep executing a code block until a certain condition is met? The while loop exists as a tool for repeated execution based on a condition. The code block will keep being executed until the given logical condition returns a False boolean value.

Let's say we would like to iterate through list dates and stop at the year 1973, then print out the number of iterations. This can be done with the following block of code:

```
In [8]: # While Loop Example

dates = [1982, 1980, 1973, 2000]

i = 0
year = dates[0]

while(year != 1973):
    print(year)
    i = i + 1
    year = dates[i]

print("It took ", i , "repetitions to get out of loop.")
```

```
1982
1980
It took 2 repetitions to get out of loop.
```

A while loop iterates merely until the condition in the argument is not met, as shown in the following figure:

```
albums = 250
total_albums = 0
i=0;                                     dates=[1982,1980,1973,1992]
while( year!=1973):
    year=dates[i]                         ↑
    i=i+1
    print(year)

print("it took",i, "outloop")               i=0          i=1          i=2
                                            year=dates[0]  year=dates[0]  year=dates[0]
                                            i=0+1        i=1+1       i=2+1
                                            print(1982)   print(1980)   print(1973)
```

```
albums = 250
total_albums = 0
i=0;                                     dates=[1982,1980,1973,1992]
while( year!=1973):  False                ↑
    year=dates[i]
    i=i+1
    print(year)

print("it took",i, "outloop")               i=0          i=1          i=2
                                            year=dates[0]  year=dates[0]  year=dates[0]
                                            i=0+1        i=1+1       i=2+1
                                            print(1982)   print(1980)   print(1973)
```

Quiz on Loops

Write a for loop that prints out all the elements between **-5** and **5** using the range function.

In [10]:

```
# Write your code below and press Shift+Enter to execute
for i in range(-5, 6):
    print(i)
```

```
-5
-4
-3
-2
-1
0
1
2
3
4
5
```

Click here for the solution ````python for i in range(-5, 6): print(i)````

Print the elements of the following list: Genres=['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop'] Make sure you follow Python conventions.

In [15]:

```
# Write your code below and press Shift+Enter to execute
Genres=[ 'rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']

for Genre in Genres:
    print(Genre)

# N = len(Genres)

# for i in range(N):
#     print(Genres[i])
```

```
rock
R&B
Soundtrack
R&B
soul
pop
```

Click here for the solution ````python Genres = ['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop'] for Genre in Genres:
print(Genre)````

Write a for loop that prints out the following list: squares=['red', 'yellow', 'green', 'purple', 'blue']

In [17]:

```
# Write your code below and press Shift+Enter to execute
squares=['red', 'yellow', 'green', 'purple', 'blue']

for square in squares:
    print(square)

# Using while loop
# i = 0
# N = Len(squares)

# while (i < N):
#     print(squares[i])
#     i = i + 1
```

```
red
yellow
green
purple
blue
```

Click here for the solution [```python squares=\['red', 'yellow', 'green', 'purple', 'blue'\] for square in squares: print\(square\)```](#)

Write a while loop to display the values of the Rating of an album playlist stored in the list PlayListRatings. If the score is less than 6, exit the loop. The list PlayListRatings is given by: PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]

In [26]:

```
# Write your code below and press Shift+Enter to execute
PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]

i = 0
N = len(PlayListRatings)

while (i < N) and (PlayListRatings[i] > 5):
    print(PlayListRatings[i])
    i = i + 1
```

```
10
9.5
10
8
7.5
```

Click here for the solution [```python PlayListRatings = \[10, 9.5, 10, 8, 7.5, 5, 10, 10\] i = 1 Rating = PlayListRatings\[0\] while\(i < len\(PlayListRatings\) and Rating >= 6\): print\(Rating\) Rating = PlayListRatings\[i\] i = i + 1```](#)

Write a while loop to copy the strings 'orange' of the list squares to the list new_squares. Stop and exit the loop if the value on the list is not 'orange':

In [29]:

```
# Write your code below and press Shift+Enter to execute

squares = ['orange', 'orange', 'purple', 'blue ', 'orange']
new_squares = []

i = 0
N = len(squares)

while (i < N) and (squares[i] == 'orange'):
    new_squares.append(squares[i])
    i = i + 1

print(new_squares)
```

```
['orange', 'orange']
```

```
['orange', 'orange']
```

Click here for the solution [```python squares = \['orange', 'orange', 'purple', 'blue ', 'orange'\] new_squares = \[\] i = 0 while\(i < len\(squares\) and squares\[i\] == 'orange'\): new_squares.append\(squares\[i\]\) i = i + 1 print \(new_squares\)```](#)

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

Practice Quiz: Loops

Bookmarked

Question 1

1/1 point (ungraded)

What will be the output of the following:

for x in range(0,3):

print(x)

- 0
- 1
- 2
- 3

- 0
- 1
- 2



Answer

Correct: Correct.

Question 2

1/1 point (ungraded)

What is the output of the following:

`for x in ['A','B','C']:`

`print(x+'A')`

A

B

C

AA

BA

CA



Answer

Correct: Correct

Submit

You have used 2 of 2 attempts

✓ Correct (1/1 point)

Question 3

1/1 point (ungraded)

What is the output of the following:

`for i,x in enumerate(['A','B','C']):`

`print(i,x)`

AA

BB

CC

0 A

1 B

2 C



Answer

Correct: Correct

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Functions

In this video we will cover functions. You will learn how to use some of Python's built-in functions as well as how to build your own functions.

Functions take some input then produce some output or change. The function is just a piece of code you can reuse.

You can implement your own function, but in many cases, you use other people's functions.

In this case, you just have to know how the function works and in some cases how to import the functions. Let the orange and yellow squares represent similar blocks of code. We can run the code using some input and get an output. If we define a function to do the task we just have to call the function. Let the small squares represent the lines of code used to call the function. We can replace these long lines of code by just calling the function a few times. Now we can just call the function; our code is much shorter. The code performs the same task. You can think of the process like this: when we call the function f1, we pass an input to the function. These values are passed to all those lines of code you wrote.

This returns a value; you can use the value. For example, you can input this value to a new function f2. When we call this new function f2, the value is passed to another set of lines of code. The function returns a value.

The process is repeated passing the values to the function you call.

You can save these functions and reuse them, or use other people's functions.

Python has many built-in functions; you don't have to know how those functions work internally, but simply what task those functions perform. The function len takes in an input of type sequence, such as a string or list, or type collection, such as a dictionary or set, and returns the length of that sequence or collection. Consider the following list.

The len function takes this list as an argument, and we assign the result to the variable L.

The function determines there are 8 items in the list, then returns the length of the list, in this case, 8. The function sum takes in an iterable like a tuple or list and returns the total of all the elements.

Consider the following list. We pass the list into the sum function and assign the result to the variable S. The function determines the total of all the elements, then returns it, in this case, the value is 70.

There are **two ways to sort a list**. The first is using the function sorted. We can also use the list method sort. Methods are similar to functions.

Let's use this as an example to illustrate the difference. The function sorted Returns a new sorted list or tuple. Consider the list album ratings. We can apply the function sorted to the list album ratings and get a new list sorted album rating.

The result is a new sorted list. If we look at the list album ratings, nothing has changed. Generally, functions take an input, in this case, a list. They produce a new output, in this instance, a sorted list.

If we use the method sort, the list album ratings will change and no new list will be created. Let's use this diagram to help illustrate the process. In this case, the rectangle represents the list album ratings. When we apply the method sort to the list, the list album rating changes. Unlike the previous case, we see that the list album rating has changed. In this case, no new list is created.

Now that we have gone over how to use functions in Python, let's see how to build our own functions. We will now get you started on building your own functions in python. This is an example of a function in python that returns its input value + 1. To define a function, we start with the keyword **def**. The name of the function should be descriptive

of what it does. We have the function formal parameter "A" in parentheses. Followed by a colon. We have a code block with an indent, for this case, we add 1 to "A" and assign it to B.

We return or output the value for b. After we define the function, we can call it. The function will add 1 to 5 and return a 6. We can call the function again; this time assign it to the variable "c" The value for 'c' is 11. Let's explore this further. Let's go over an example when you call a function. It should be noted that this is a simplified model of Python, and Python does not work like this under the hood. We call the function giving it an input, 5. It helps to think of the value of 5 as being passed to the function.

Now the sequences of commands are run, the value of "A" is 5. "B" would be assigned a value of 6. We then return the value of b, in this case, as b was assigned a value of 6, the function returns a 6. If we call the function again, the process starts from scratch; we pass in an 8.

The subsequent operations are performed. Everything that happened in the last call

will happen again with a different value of "A" The function returns a value, in this case, 9.

Again, this is just a helpful analogy. Let's try and make this function more complex.

It's customary to document the function on the first few lines; this tells anyone who uses the function what it does. This documentation is surrounded in triple

quotes. You can use the help command on the function to display the documentation as follows. This will printout the function name and the documentation. We will not include the documentation in the rest of the examples. A function can have multiple parameters.

The function `mult` multiplies two numbers; in other words, it finds their product.

If we pass the integers 2 and 3, the result is a new integer. If we pass the integer 10 and the float 3.14, the result is a float 31.4. If we pass in the integer two and the string "Michael Jackson," the string Michael Jackson is repeated two times. This is because the multiplication symbol can also mean repeat a sequence. If you accidentally multiply an integer and

a String instead of two integers, you won't get an error. Instead, you will get a String, and your program will progress, potentially failing later because you have a String where you expected an integer. This property will make coding simpler, but you must test your code more thoroughly. In many cases a function does not have a return statement. In these cases, Python will return the special "None" object. Practically speaking, if your function has

no return statement, you can treat it as if the function returns nothing at all. The function MJ simply prints the name 'Michael Jackson'. We call the function. The function prints "Michael Jackson." Let's define the function "No work" that performs no task.

Python doesn't allow a function to have an empty body, so we can use the keyword `pass`,

which doesn't do anything, but satisfies the requirement of a non-empty body.

If we call the function and print it out, the function returns a None. In the background, if the return statement is not called, Python will automatically return a None. It is helpful to view the function No Work with the following return statement. Usually, functions perform more than one task.

This function prints a statement then returns a value. Let's use this table to represent the different values as the function is called. We call the function with an input of 2. We find the value of b. The function prints the statement with the values of a and b.

Finally, the function returns the value of b, in this case, 3. We can use loops in functions. This function prints out the values and indexes of a loop or tuple. We call the function with the list album ratings as an input. Let's display the list on the right with its corresponding index. Stuff is used as an input to the function enumerate. This operation will pass the index to i and the value in the list to "s". The function would begin to iterate through the loop. The function will print the first index and the first value in the list. We continue iterating through the loop.

The values of i and s are updated. The print statement is reached. Similarly, the next values of the list and index are printed. The process is repeated. The values of i and s are updated.

We continue iterating until the final values in the list are printed out. Variadic parameters allow us to input a variable number of elements. Consider the following function; the function has an asterisk on the parameter names. When we call the function, three parameters are packed into the tuple names. We then iterate through the loop; the values are printed out accordingly. If we call the same function with only two parameters as inputs, the variable names only contain two elements. The result is only two values are printed out. The scope of a variable is the part of the program where that variable is accessible. Variables that are defined outside of any function are said to be within the global scope, meaning they can be accessed anywhere after they are defined. Here we have a function that adds the string DC to the parameter x. When we reach the part where the value of

x is set to AC, this is within the **global scope**, meaning x is accessible anywhere after

it is defined. A variable defined in the global scope is called a **global variable**. When we call the function, we enter a new scope or the scope of AddDC. We pass as an argument to the AddDC function, in this case, AC. Within the scope of the function, the value

of x is set to ACDC. The function returns the value and is assigned to z. Within the global scope, the value z is set to ACDC After the value is returned, the scope of the function is deleted. Local variables only exist within the scope of a function. Consider the function thriller; the local variable Date is set to 1982. When we call the function, we create a new

scope. Within that scope of the function, the value of the date is set to 1982. The value of date does not exist within the global scope. Variables inside the global scope can have

the same name as variables in the local scope with no conflict. Consider the function thriller; the local variable Date is set to 1982. The global variable date is set to 2017. When we call the function, we create a new scope. Within that scope, the value of the date is set to 1982. If we call the function, it returns the value of Date in the local scope, in this case, 1982. (click6) When we print in the global scope, we use the global variable value.

The global value of the variable is 2017. Therefore, the value is set to 2017. If a variable is not defined within a function, Python will check the global scope. Consider the function "AC-DC". The function has the variable rating, with no value assigned. If we define the variable rating in the global scope, then call the function, Python will see there is no value for the variable Rating. As a result, python will leave the scope and check if the variable Ratings exists in the global scope. It will use this value of Ratings in the global scope within the scope of "AC-DC". In the function, will print out a 9. The value of z in the global scope will be 10, as we added one.

The value of rating will be unchanged within the global scope. Consider the function Pink Floyd. If we define the variable Claimed Sales with the keyword global, the variable will be a global variable. We call the function Pink Floyd. The variable claimed sales is set to the string "45 million" in the global scope. When we print the variable, we get a value of "45 million." There is a lot more you can do with functions.

Functions

Functions

Developer

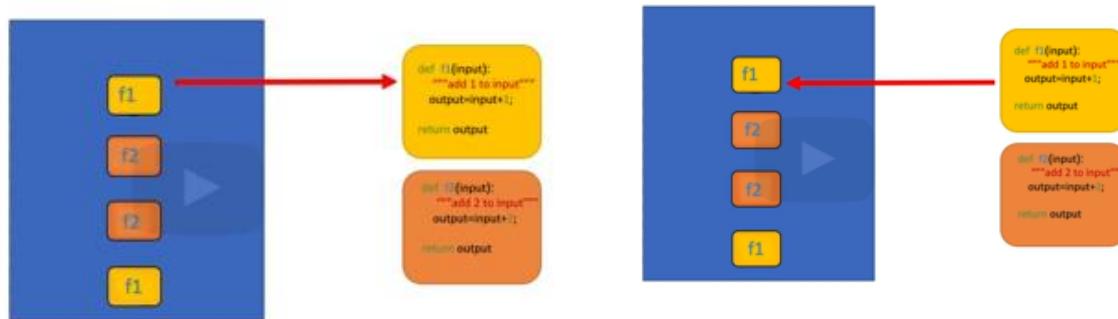
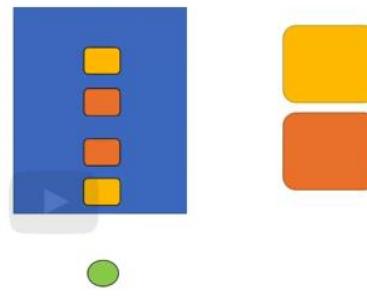
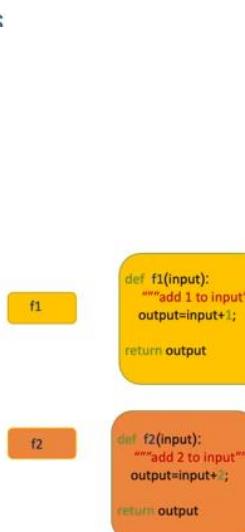
SKILL

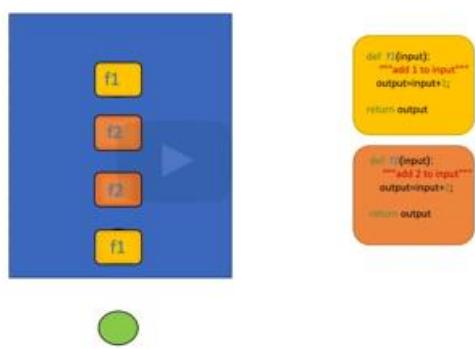
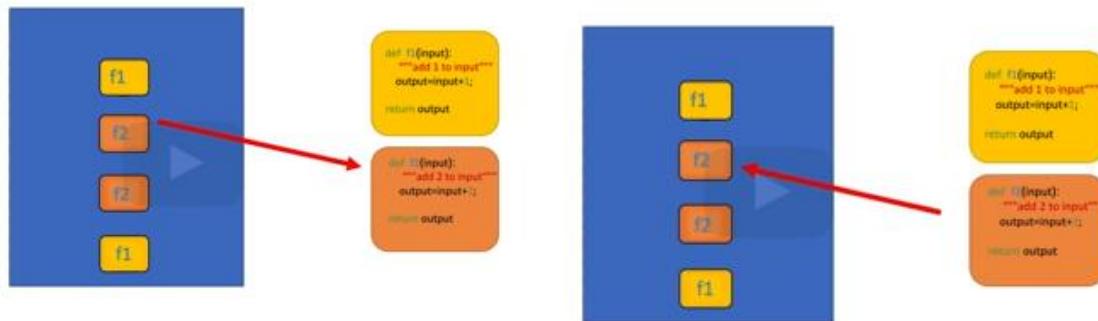


IBM Developer

SKILL

IBMD





Python Built-in functions

Len

album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]

L=len(album_ratings)

album_ratings

len

Len

album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|

L=len(album_ratings)

L:8

len

Sum

```
album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
```

```
S=sum(album_ratings)
```



```
album_ratings
```

```
sum
```

Sum

```
album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
```

```
10.0+8.5+9.5+7.0+7.0+9.5+9.0+9.5
```

```
S=sum(album_ratings)
```



```
S:70
```

```
sum
```

```
70
```

Sorted vs Sort

```
album_ratings
```

```
sorted
```

Sorted vs Sort

```
album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
```

```
sorted_album_rating = sorted(album_ratings)
```

```
sorted_album_rating:
```

```
[7.0, 7.0, 8.5, 9.0, 9.5, 9.5, 9.5, 10.0]
```

```
sorted
```

```
album_ratings :
```

```
[10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
```

```
[7.0, 7.0, 8.5, 9.0, 9.5, 9.5, 9.5, 10.0]
```

Sorted vs Sort

```
album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
```

```
album_ratings.sort()
```

```
sort()
```

```
album_ratings
```

Sorted vs Sort

```
album_ratings = [10.0,8.5,9.5,7.0,7.0,9.5,9.0,9.5]
```

```
album_ratings.sort()
```

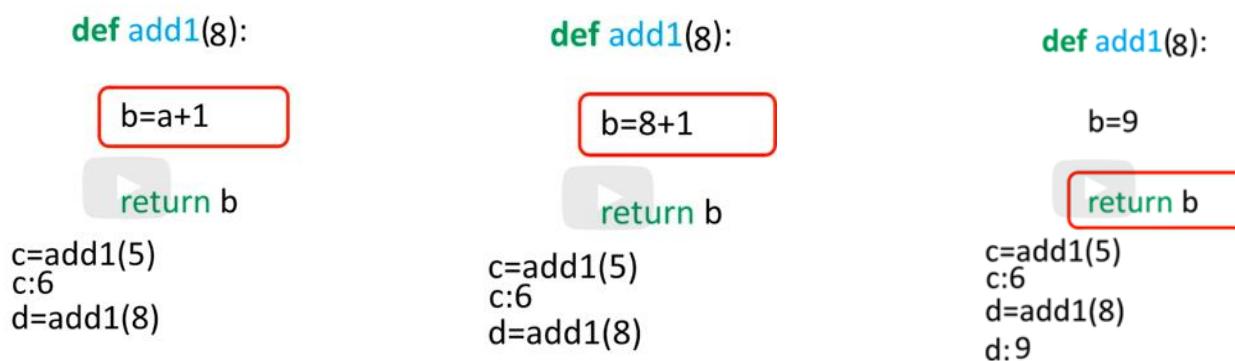
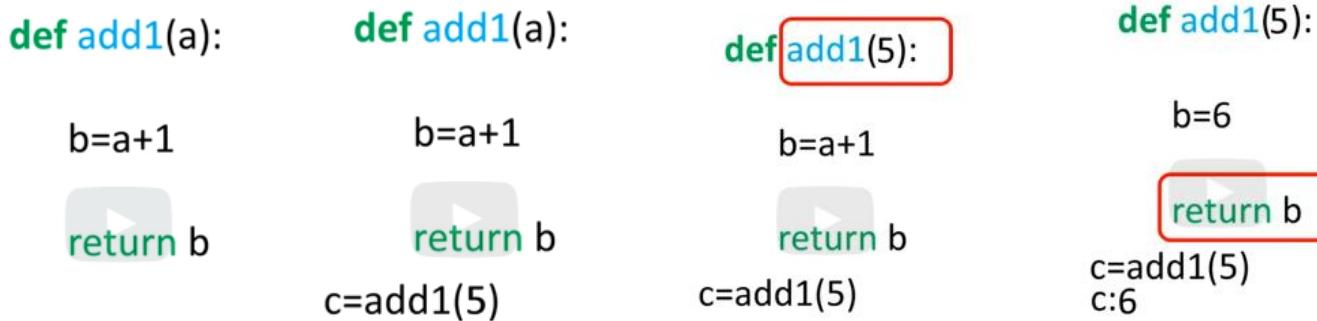
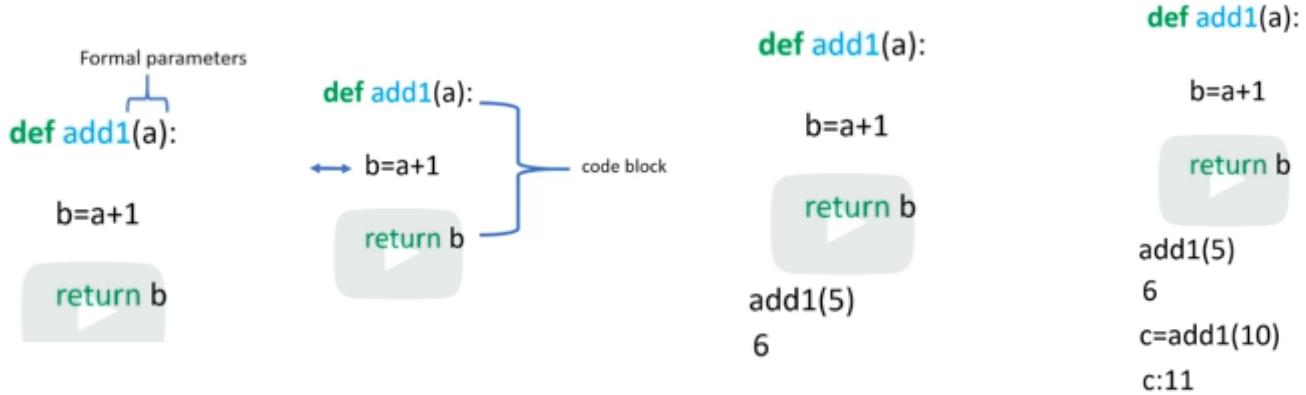
```
album_ratings:
```

```
[7.0, 7.0, 8.5, 9.0, 9.5, 9.5, 9.5, 10.0]
```



```
album_ratings
```

Making Functions



```

def add1(a):
    """
    add 1 to a
    """
    b=a+1;
    return b

```

```

def add1(a):
    """
    add 1 to a
    """
    b=a+1;
    return b

```

Documentation String

Documentation String

`help(add1)`

Help on function add1 in module __main__: add1(a) add 1 to a

Multiple Parameters

- A function can have multiple parameters

```
def Mult(a,b):
    c=a*b
    return c
```

Mult(2,3)

6

Mult(10,3.14)

31.4

```
def Mult(a,b):
    c=a*b
    return c
```

Mult(2,"Michael Jackson ")

"Michael Jackson Michael Jackson "

2*"Michael Jackson "

"Michael Jackson Michael Jackson "

```
def MJ():
    print('Michael Jackson')
MJ()
'Michael Jackson'
```



```
def NoWork():
    pass
print(NoWork())
None
```

```
def NoWork():
    pass
print(NoWork())
```

None



```
def NoWork():
    pass
return None
```



```
def printStuff(Stuff):
    for i,s in enumerate(Stuff):
        print("Album", i , "Rating is ", s)
```

Stuff: [10.0, 8.5, 9.5]

Index: 0 1 2

```
album_ratings = [10.0,8.5,9.5]
printStuff(album_ratings)
```

Album 0 Rating is 10
Album 1 Rating is 8.5
Album 2 Rating is 9.5

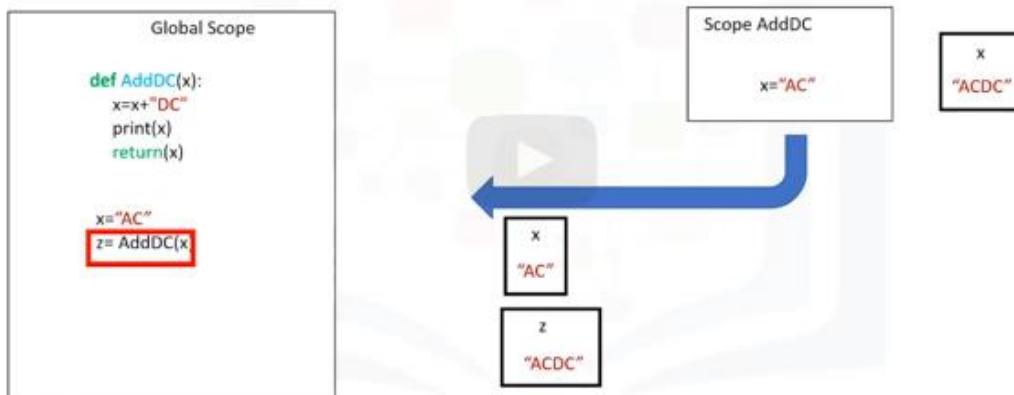
Collecting arguments

```
def ArtistNames(*names):  
    for name in names:  
        print(name) → names=("Michael Jackson","AC/DC","Pink Floyd")  
  
ArtistNames("Michael Jackson","AC/DC","Pink Floyd")  
  
Michael Jackson  
AC/DC  
Pink Floyd
```

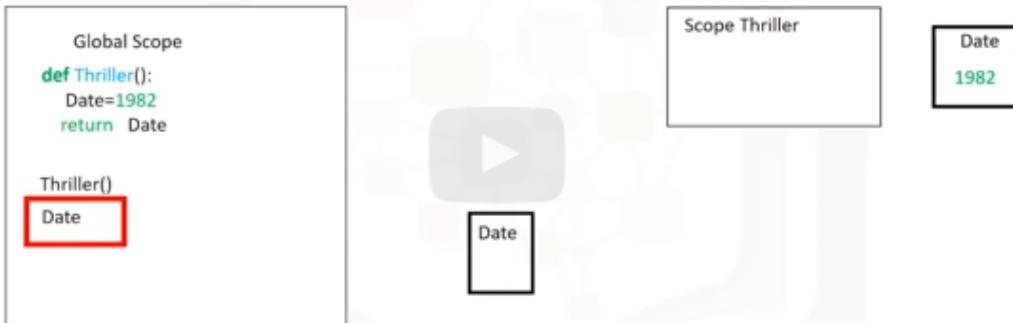
Collecting arguments

```
def ArtistNames(*names):  
    for name in names:  
        print(name) → names=("Michael Jackson","AC/DC")  
  
ArtistNames("Michael Jackson","AC/DC")  
  
Michael Jackson  
AC/DC
```

Scope



Scope: Local Variables



Scope: Local Variables



Scope: Variables



Scope: Local Variables

```
Global Scope  
def PinkFloyd():  
    global ClaimedSales  
    ClaimedSales ='45 million'  
    return ClaimedSales  
  
PinkFloyd()  
print(ClaimedSales)  
45 million
```

Claimed Sales
'45 millions'



IBM Developer
SKILLS NETWORK

Functions in Python

Estimated time needed: **40** minutes

Objectives

After completing this lab you will be able to:

- Understand functions and variables
- Work with functions and variables

Functions in Python

Welcome! This notebook will teach you about the functions in the Python Programming Language. By the end of this lab, you'll know the basic concepts about function, variables, and how to use functions.

Table of Contents

- Functions
 - What is a function?
 - Variables
 - Functions Make Things Simple
- Pre-defined functions
- Using `if / else` Statements and Loops in Functions
- Setting default argument values in your custom functions
- Global variables
- Scope of a Variable
- Collections and Functions
- Quiz on Loops

Functions

A function is a reusable block of code which performs operations specified in the function. They let you break down tasks and allow you to reuse your code in different programs.

There are two types of functions :

- **Pre-defined functions**
- **User defined functions**

What is a Function?

You can define functions to provide the required functionality. Here are simple rules to define a function in Python:

- Functions blocks begin def followed by the function name and parentheses () .
- There are input parameters or arguments that should be placed within these parentheses.
- You can also define parameters inside these parentheses.
- There is a body within every function that starts with a colon (:) and is indented.
- You can also place documentation before the body.
- The statement return exits a function, optionally passing back a value.

An example of a function that adds one to the parameter a prints and returns the output as b:

An example of a function that adds one to the parameter a prints and returns the output as b :

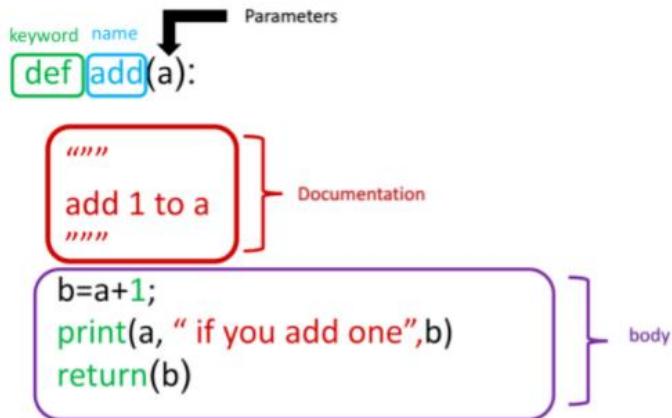
```
In [2]: # First function example: Add 1 to a and store as b

def add(a):
    b = a + 1
    print(a, "if you add one", b)
    return(b)

add(3)

3 if you add one 4
4
```

The figure below illustrates the terminology:



`add(1)`

We can obtain help about a function :

```
In [3]: # Get a help on add function
help(add)

Help on function add in module __main__:

add(a)
```

We can call the function:

```
In [4]: # Call the function add()
add(1)

1 if you add one 2
2
```

Out[4]:

If we call the function with a new input we get a new result:

```
In [5]: # Call the function add()
add(2)

2 if you add one 3
3
```

Out[5]:

We can create different functions. For example, we can create a function that multiplies two numbers. The numbers will be represented by the variables a and b:

```
In [6]: # Define a function for multiple two numbers

def Mult(a, b):
    c = a * b
    return(c)
    print('This is not printed')

result = Mult(12,2)
print(result)
```

24

The same function can be used for different data types. For example, we can multiply two integers:

```
In [7]: # Use mult() multiply two integers

Mult(2, 3)
```

Out[7]: 6

Note how the function terminates at the `return` statement, while passing back a value. This value can be further assigned to a different variable as desired.

The same function can be used for different data types. For example, we can multiply two integers:

Two Floats:

```
In [8]: # Use mult() multiply two floats

Mult(10.0, 3.14)
```

Out[8]: 31.400000000000002

We can even replicate a string by multiplying with an integer:

```
In [11]: # Use mult() multiply two different type values together

Mult(2, "Michael Jackson ")
```

Out[11]: 'Michael Jackson Michael Jackson '

Variabls

The input to a function is called a formal parameter.

A variable that is declared inside a function is called a local variable. The parameter only exists within the function (i.e. the point where the function starts and stops).

A variable that is declared outside a function definition is a global variable, and its value is accessible and modifiable throughout the program. We will discuss more about global variables at the end of the lab.

```
In [13]: # Function Definition

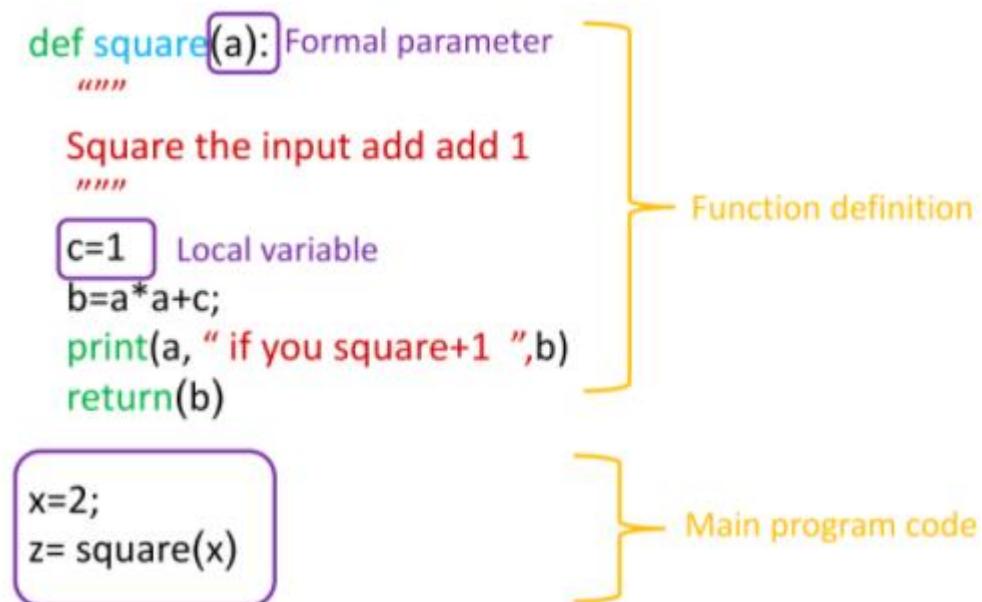
def square(a):

    # Local variable b
    b = 1
    c = a * a + b
    print(a, "if you square + 1", c)
    return(c)

square(4)
```

```
Out[13]: 4 if you square + 1 17
17
```

The labels are displayed in the figure:



We can call the function with an input of **3**:

```
In [14]: # Initializes Global variable  
  
x = 3  
# Makes function call and return function a y  
y = square(x)  
y  
  
3 if you square + 1 10  
Out[14]: 10
```

We can call the function with an input of **2** in a different manner:

```
In [15]: # Directly enter a number as parameter  
  
square(2)  
  
2 if you square + 1 5  
Out[15]: 5
```

If there is no return statement, the function returns None. The following two functions are equivalent:

```
In [16]: # Define functions, one with return value None and other without return value  
  
def MJ():  
    print('Michael Jackson')  
  
def MJ1():  
    print('Michael Jackson')  
    return(None)
```

```
In [17]: # See the output
```

```
MJ()
```

Michael Jackson

```
In [18]: # See the output
```

```
MJ1()
```

Michael Jackson

Printing the function after a call reveals a **None** is the default return statement:

```
In [19]: # See what functions returns are  
  
print(MJ())  
print(MJ1())
```

```
Michael Jackson  
None  
Michael Jackson  
None
```

Create a function `con` that concatenates two strings using the addition operation:

```
In [20]: # Define the function for combining strings  
  
def con(a, b):  
    return(a + b)
```

```
In [24]: # Test on the con() function  
  
con("This ", "is")
```

```
Out[24]: 'This is'
```

[Tip] How do I learn more about the pre-defined functions in Python?

We will be introducing a variety of pre-defined functions to you as you learn more about Python. There are just too many functions, so there's no way we can teach them all in one sitting. But if you'd like to take a quick peek, here's a short reference card for some of the commonly-used pre-defined functions: [Reference](#)

Functions Make Things Simple

Consider the two lines of code in **Block 1** and **Block 2**: the procedure for each block is identical. The only thing that is different is the variable names and values.

Block 1:

```
In [25]: # a and b calculation block1  
  
a1 = 4  
b1 = 5  
c1 = a1 + b1 + 2 * a1 * b1 - 1  
if(c1 < 0):  
    c1 = 0  
else:  
    c1 = 5  
c1
```

```
Out[25]: 5
```

Block 2:

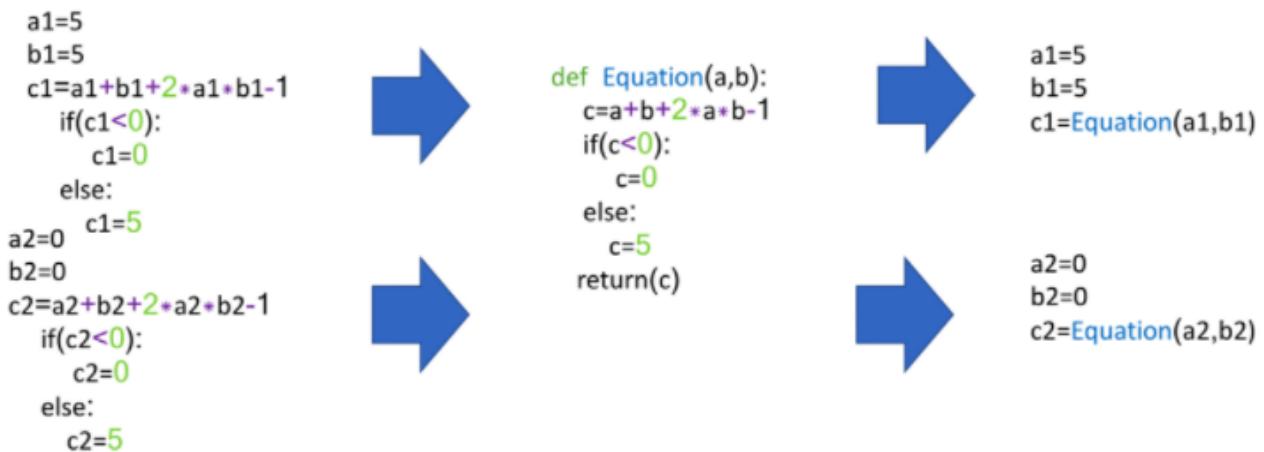
```
In [26]: # a and b calculation block2  
  
a2 = 0  
b2 = 0  
c2 = a2 + b2 + 2 * a2 * b2 - 1  
if(c2 < 0):  
    c2 = 0  
else:  
    c2 = 5  
c2
```

```
Out[26]: 0
```

We can replace the lines of code with a function. A function combines many instructions into a single line of code. Once a function is defined, it can be used repeatedly. You can invoke the same function many times in your program. You can save your function and use it in another program or use someone else's function. The lines of code in code **Block 1** and code **Block 2** can be replaced by the following function:

```
In [27]: # Make a Function for the calculation above  
  
def Equation(a,b):  
    c = a + b + 2 * a * b - 1  
    if(c < 0):  
        c = 0  
    else:  
        c = 5  
    return(c)
```

This function takes two inputs, a and b, then applies several operations to return c. We simply define the function, replace the instructions with the function, and input the new values of a1, b1 and a2, b2 as inputs. The entire process is demonstrated in the figure:



Code **Blocks 1** and **Block 2** can now be replaced with code **Block 3** and code **Block 4**.

Block 3:

```

In [28]: a1 = 4
          b1 = 5
          c1 = Equation(a1, b1)
          c1

Out[28]: 5

```

Block 4:

```

In [29]: a2 = 0
          b2 = 0
          c2 = Equation(a2, b2)
          c2

Out[29]: 0

```

Pre-defined functions

There are many pre-defined functions in Python, so let's start with the simple ones.

The `print()` function:

In [30]:

```
# Build-in function print()

album_ratings = [10.0, 8.5, 9.5, 7.0, 7.0, 9.5, 9.0, 9.5]
print(album_ratings)
```

```
[10.0, 8.5, 9.5, 7.0, 7.0, 9.5, 9.0, 9.5]
```

The `sum()` function adds all the elements in a list or tuple:

In [31]:

```
# Use sum() to add every element in a list or tuple together

sum(album_ratings)
```

Out[31]: 70.0

The `len()` function returns the length of a list or tuple:

In [32]:

```
# Show the length of the list or tuple

len(album_ratings)
```

Out[32]: 8

Using `if/else` Statements and Loops in Functions

The `return()` function is particularly useful if you have any IF statements in the function, when you want your output to be dependent on some condition:

```
In [33]: # Function example

def type_of_album(artist, album, year_released):

    print(artist, album, year_released)
    if year_released > 1980:
        return "Modern"
    else:
        return "Oldie"

x = type_of_album("Michael Jackson", "Thriller", 1980)
print(x)
```

```
Michael Jackson Thriller 1980
Oldie
```

We can use a loop in a function. For example, we can `print` out each element in a list:

```
In [34]: # Print the list using for Loop

def PrintList(the_list):
    for element in the_list:
        print(element)
```

```
In [35]: # Implement the printlist function

PrintList(['1', 1, 'the man', "abc"])
```

```
1
1
the man
abc
```

Setting default argument values in your custom functions

You can set a default value for arguments in your function. For example, in the `isGoodRating()` function, what if we wanted to create a threshold for what we consider to be a good rating? Perhaps by default, we should have a default rating of 4:

```
In [36]: # Example for setting param with default value

def isGoodRating(rating=4):
    if(rating < 7):
        print("this album sucks it's rating is",rating)

    else:
        print("this album is good its rating is",rating)
```



```
In [37]: # Test the value with default value and with input

isGoodRating()
isGoodRating(10)
```

this album sucks it's rating is 4
this album is good its rating is 10

Global variables

So far, we've been creating variables within functions, but we have not discussed variables outside the function. These are called global variables.

Let's try to see what printer1 returns:

```
In [41]: # Example of global variable

artist = "Michael Jackson"
def printer1(artist):
    internal_var1 = artist
    print(artist, "is an artist")

printer1(artist)
# try running the following code
#printer1(internal_var1)
```

Michael Jackson is an artist

We got a Name Error: name 'internal_var' is not defined. Why?

It's because all the variables we create in the function is a **local variable**, meaning that the variable assignment does not persist outside the function.

But there is a way to create **global variables** from within a function as follows:

```
In [42]: artist = "Michael Jackson"

def printer(artist):
    global internal_var
    internal_var= "Whitney Houston"
    print(artist,"is an artist")

printer(artist)
printer(internal_var)
```

Michael Jackson is an artist
Whitney Houston is an artist

Scope of a Variable

The scope of a variable is the part of that program where that variable is accessible. Variables that are declared outside of all function definitions, such as the myFavouriteBand variable in the code shown here, are accessible from anywhere within the program. As a result, such variables are said to have global scope, and are known as global variables. myFavouriteBand is a global variable, so it is accessible from within the getBandRating function, and we can use it to determine a band's rating. We can also use it outside of the function, such as when we pass it to the print function to display it:

```
In [43]: # Example of global variable

myFavouriteBand = "AC/DC"

def getBandRating(bandname):
    if bandname == myFavouriteBand:
        return 10.0
    else:
        return 0.0

print("AC/DC's rating is:", getBandRating("AC/DC"))
print("Deep Purple's rating is:",getBandRating("Deep Purple"))
print("My favourite band is:", myFavouriteBand)

AC/DC's rating is: 10.0
Deep Purple's rating is: 0.0
My favourite band is: AC/DC
```

Take a look at this modified version of our code. Now the myFavouriteBand variable is defined within the getBandRating function. A variable that is defined within a function is said to be a local variable of that function. That means that it is only accessible from within the function in which it is defined.

Our getBandRating function will still work, because myFavouriteBand is still defined within the function. However, we can no longer print myFavouriteBand outside our function, because it is a local variable of our getBandRating function; it is only defined within the getBandRating function:

In [44]:

```
# Example of Local variable

def getBandRating(bandname):
    myFavouriteBand = "AC/DC"
    if bandname == myFavouriteBand:
        return 10.0
    else:
        return 0.0

print("AC/DC's rating is: ", getBandRating("AC/DC"))
print("Deep Purple's rating is: ", getBandRating("Deep Purple"))
print("My favourite band is", myFavouriteBand)
```

```
AC/DC's rating is: 10.0
Deep Purple's rating is: 0.0
My favourite band is AC/DC
```

Finally, take a look at this example. We now have two myFavouriteBand variable definitions. The first one of these has a global scope, and the second of them is a local variable within the getBandRating function. Within the getBandRating function, the local variable takes precedence. **Deep Purple** will receive a rating of 10.0 when passed to the getBandRating function. However, outside of the getBandRating function, the getBandRating's local variable is not defined, so the myFavouriteBand variable we print is the global variable, which has a value of **AC/DC**:

In [45]:

```
# Example of global variable and local variable with the same name

myFavouriteBand = "AC/DC"

def getBandRating(bandname):
    myFavouriteBand = "Deep Purple"
    if bandname == myFavouriteBand:
        return 10.0
    else:
        return 0.0

print("AC/DC's rating is:",getBandRating("AC/DC"))
print("Deep Purple's rating is: ",getBandRating("Deep Purple"))
print("My favourite band is:",myFavouriteBand)
```

```
AC/DC's rating is: 0.0
Deep Purple's rating is: 10.0
My favourite band is: AC/DC
```

Collections and Functions

When the number of arguments are unknown for a function, They can all be packed into a tuple as shown:

```
In [46]: def printAll(*args): # All the arguments are 'packed' into args which can be treated like a tuple
    print("No of arguments:", len(args))
    for argument in args:
        print(argument)
#printAll with 3 arguments
printAll('Horsefeather','Adonis','Bone')
#printAll with 4 arguments
printAll('Sidecar','Long Island','Mudslide','Carriage')
```

```
No of arguments: 3
Horsefeather
Adonis
Bone
No of arguments: 4
Sidecar
Long Island
Mudslide
Carriage
```

Similarly, The arguments can also be packed into a dictionary as shown:

```
In [47]: def printDictionary(**args):
    for key in args:
        print(key + " : " + args[key])

printDictionary(Country='Canada',Province='Ontario',City='Toronto')
```

```
Country : Canada
Province : Ontario
City : Toronto
```

Functions can be incredibly powerful and versatile. They can accept (and return) data types, objects and even other functions as arguments. Consider the example below:

```
In [48]: def addItems(list):
    list.append("Three")
    list.append("Four")

myList = ["One","Two"]

addItems(myList)

myList
```

```
Out[48]: ['One', 'Two', 'Three', 'Four']
```

Note how the changes made to the list are not limited to the functions scope. This occurs as it is the lists **reference** that is passed to the function - Any changes made are on the original instance of the list. Therefore, one should be cautious when passing mutable objects into functions.

Quiz on Functions

Come up with a function that divides the first input by the second input:

```
In [50]: # Write your code below and press Shift+Enter to execute
def div(a, b):
    return(a / b)

div(4, 2)
```

```
Out[50]: 2.0
```

[Click here for the solution](#) `python def div(a, b): return(a/b)`

Use the function `con` for the following question.

```
In [51]: # Use the con function for the following question

def con(a, b):
    return(a + b)
```

Can the `con` function we defined before be used to add two integers or strings?

```
In [57]: # Write your code below and press Shift+Enter to execute
twoStrings = con("This is", " an example")
twoIntegers = con(3, 5)

print(twoStrings)
print(twoIntegers)
```

```
This is an example
8
```

[Click here for the solution](#) `python Yes, for example: con(2, 2)`

```
In [58]: # Write your code below and press Shift+Enter to execute
conLists = con((1,2), (3,4))
conTuples = con([('a','b'),[3,4,'c'])]

print(conLists)
print(conTuples)
```

```
(1, 2, 3, 4)
['a', 'b', 3, 4, 'c']
```

[Click here for the solution](#) `python Yes, for example: con(['a', 1], ['b', 1])`

In [58]:

```
# Write your code below and press Shift+Enter to execute
conLists = con((1,2), (3,4))
conTuples = con(['a','b'],[3,4,'c'])

print(conLists)
print(conTuples)
```

```
(1, 2, 3, 4)
['a', 'b', 3, 4, 'c']
```

Click here for the solution [```python Yes, for example: con\(\['a', 1\], \['b', 1\]\)```](#)

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

Exception Handling

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Understand exceptions
- Handle the exceptions

Table of Contents

- What is an Exception?
 - Exception Handling
-

What is an Exception?

In this section you will learn about what an exception is and see examples of them.

Definition

An exception is an error that occurs during the execution of code. This error causes the code to raise an exception and if not prepared to handle it will halt the execution of the code.

Examples

Run each piece of code and observe the exception raised

```
In [1]: 1/0
```

```
-----
ZeroDivisionError                                     Traceback (most recent call last)
<ipython-input-1-9e1622b385b6> in <module>
----> 1 1/0

ZeroDivisionError: division by zero
ZeroDivisionError occurs when you try to divide by zero.
```

```
In [2]: y = a + 5
```

```
-----
NameError                                         Traceback (most recent call last)
<ipython-input-2-6ddcec040107> in <module>
----> 1 y = a + 5

NameError: name 'a' is not defined
NameError in this case means that you tried to use the variable a when it was not defined.
```

```
In [3]: a = [1, 2, 3]
a[10]
```

```
-----
IndexError                                         Traceback (most recent call last)
<ipython-input-3-3f911ca4e3d3> in <module>
      1 a = [1, 2, 3]
----> 2 a[10]

IndexError: list index out of range
```

IndexError in this case occurs when you try to access data from a list using an index that does not exist for this list.

There are many more exceptions that are built into Python, here is a list of them <https://docs.python.org/3/library/exceptions.html>

Exception Handling

In this section you will learn how to handle exceptions to perform certain tasks and not halt the execution of your code.

Try Except

A try except will allow you to execute code that might raise an exception and in the case of any exception or a specific one we can handle or catch the exception and execute specific code. This will allow us to continue the execution of our program even if there is an exception.

Python tries to execute the code in the try block. In this case if there is any exception raised by the code in the try block it will be caught and the code block in the except block will be executed. After the code that comes after the try except will be executed.

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except:
    # code to execute if there is an exception

# code that will still execute if there is an exception
```

Try Except Example

In this example we are trying to divide a number given by the user, save the outcome in the variable a, and then we would like to print the result of the operation. When taking user input and dividing a number by it there are a couple of exceptions that can be raised. For example if we divide by zero, try running the following block of code, with b as a number an exception will only be Raised if b, is zero

```
In [5]: a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
    print("Success a=",a)
except:
    print("There was an error")
```

There was an error

Try Except Specific

A specific try except allows you to catch certain exceptions and also execute certain code depending on the exception. This is useful if you do not want to deal with some exceptions and the execution should halt, it can also help you find errors in your code that you might not know about, and it can help you differentiate responses to different exceptions. In this case the code after the try except might not run depending on the error.

do not run, just to illustrate

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except (ZeroDivisionError, NameError):
    # code to execute if there is an exception of the given types

# code that will execute if there is no exception or a one that we are handling
```

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError

# code that will execute if there is no exception or a one that we are handling
```

You can also have an empty except at the end to catch an unexpected exception:

do not run, just to illustrate

```
-----
```

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if ther is any exception

# code that will execute if there is no exception or a one that we are handling
```

Try Except Specific Example

This is the same example as above, but now we will add differentiated messages depending on the exception letting the user know what is wrong with the input.

In [15]:

```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
    print("Success a=",a)
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
```

You did not provide a number

Try Except Else and Finally

else allows one to check if there was no exception when executing the try block. This is useful when we want to execute something only if there were no errors.

do not run, just to illustrate

In []:

```
# potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if ther is any exception
else:
    # code to execute if there is no exception

# code that will execute if there is no exception or a one that we are handling
```

finally allows us to always execute something even if there is an exception or not. This is usually used to signify the end of the try except.

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if ther is any exception
else:
    # code to execute if there is no exception
finally:
    # code to execute at the end of the try except no matter what

# code that will execute if there is no exception or a one that we are handling
```

Try Except Else and Finally Example

You might have noticed that even if there is an error the value of a is always printed. Lets use the else and print the value of a only if there is no error.

```
- - - - -
In [ ]: a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
else:
    print("success a=",a)
```

Now lets let the user know that we are done processing their answer. Using the finally lets add a print.

In []:

```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
else:
    print("success a=",a)
finally:
    print("Processing Complete")
```

Practice Quiz: Functions

 Bookmarked

Question 1

1/1 point (ungraded)

What does the following function return: `len(['A','B','1'])` ?

4

2

3



Answer

Correct: Correct, the function returns the number of elements in the list, in this case 3.

Submit

You have used 1 of 2 attempts

 Correct (1/1 point)

Question 2

1/1 point (ungraded)

What does the following function return: `len([sum([0,0,1])])` ?

3

0

1



Answer

Correct: Correct.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 3

1/1 point (ungraded)

What is the value of list L after the following code segment is run?

L=[1,3,2]

`sorted(L)`

L:[0,0,0]

L:[1,2,3]

L:[1,3,2]



Answer

Correct: Correct, `sorted` is a function and returns a new list, it does not change the list L.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 4

1/1 point (ungraded)

From the video what is the value of c after the following?

c=add1(2)

c=add1(10)

14

11

3

**Answer**

Correct: Correct, when you call the function the second time the value of c is reassigned.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 5

1/1 point (ungraded)

What is the output of the following lines of code?

```
def Print(A):
```

```
    for a in A:
```

```
        print(a+'1')
```

```
Print(['a','b','c'])
```



a1



a1

b1

c1



a

b

c

**Answer**

Correct: Correct, the function concatenates a string.

Submit

You have used 1 of 2 attempts



Correct (1/1 point)

Exception Handling

you will be able to:

Explain Exception Handling, Demonstrate the use of exception handling, and Understand the basics of exception handling.

Have you ever mistakenly entered a number when you were supposed to enter text in an input field? Most of us have either in error or when testing out a program, but do you know why it gave an error message instead of completing and terminating the program?

In order for the error message to appear an event was triggered in the background.

This event was activated because the program tried to perform a computation on the name entry and realized the entry contained numbers and not letters. By encasing this code in an exception handler the program knew how to deal with this type

of error and was able to output the error message to continue along with the program.

This is one of many errors that can happen when asking for user input, so let's see how exception handling works. Let's first explore the try...except statement.

This type of statement will first attempt to execute the code in the "try" block, but if an error occurs it will kick out and begin searching for the exception that matches the error. Once it finds the correct exception to handle the error it will then execute that line of code.

For example, let's say you are writing a program that will open and write a file.

After starting the program an error occurred as the data was not able to be read.

Because of this error the program skipped over the code lines under the "try" statement and went directly to the exception line.

Since this error fell within the IOError guidelines it printed "Unable to open or read the data in the file." to our console.

When writing simple programs we can sometimes get away with only one except statement, but what happens if another error occurs that is not caught by the IOError?

If that happened we would need to add another except statement.

For this except statement you will notice that the type of error to catch is not specified.

While this may seem a logical step so the program will catch all errors and not terminate this is not a best practice.

For example, let's say our small program was just one section of a much larger program that was over a thousand lines of code.

Our task was to debug the program as it kept throwing an error causing a disruption for our users. When investigating the program you found this error kept appearing. Because this error had no details you ended up spending hours trying to pinpoint and fix the error. So far in our program we have defined that an error message should print out if an error occurs, but we don't receive any messages that the program executed properly.

This is where we can now add an else statement to give us that notification.

By adding this else statement it will provide us a notification to the console that "The file was written successfully". Now that we have defined what will happen if our program executes properly, or if an error occurs there is one last statement to add.

For this example since we are opening a file the last thing we need to do is close the file. By adding a finally statement it will tell the program to close the file no matter the end result and print "File is now closed" to our console.

In this video, you learned: How to write a try...except statement,

Why is it important to always define errors when creating exceptions, and

How to add an else and finally statement.

Exception Handling

Objectives

After watching this video, you will be able to:

- Explain Exception Handling
- Demonstrate the use of exception handling
- Understand the basics of exception handling

Exception Handling

Try...Except Statement

Please enter your name: Please only enter letters

try:

except:

Try...Except Statement



Try...Except Statement

```
try:  
    myfile = open("myfile", "r")  
    myfile.write("File for exception  
handling.")  
except IOError:  
    print("Unable to open or read the data  
in the file.")
```

Try...Except Statement

Unable to open or read the data in the file.

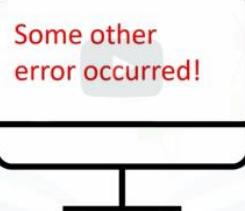
Try...Except Statements

```
try:  
    myfile = open("myfile", "r")  
    myfile.write("My file for exception handling.")  
except IOError:  
    print("Unable to open or read the data in the file.")  
except:  
    print("Some other error occurred!")
```

if another error occurs and not seen by IOError, we add another except statement

Try...Except...Else Statement

Try...Except Statements



```
try:  
    myfile = open("myfile", "r")  
    myfile.write("My file for exception handling.")  
except IOError:  
    print("Unable to open or read the data in the file.")  
else:  
    print("The file was written successfully")
```

Try...Except...Else Statement

The file was written successfully.

Try...Except...Else...Finally Statement

```
try:  
    myfile = open("myfile", "r")  
    myfile.write("My file for exception  
handling.")  
except IOError:  
    print("Unable to open or read the data in  
the file.")  
else:  
    print("The file was written successfully")
```

To Close the file: add "finally" statement

Try...Except...Else...Finally Statement

```
try:  
    myfile = open("myfile", "r")  
    myfile.write("My file for exception  
handling.")  
except IOError:  
    print("Unable to open or read the data in  
the file.")  
else:  
    print("The file was written successfully")  
finally:  
    myfile.close()  
    print("File is now closed.")
```



IBM Developer
SKILLS NETWORK

Exception Handling

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Understand exceptions
- Handle the exceptions

Table of Contents

- What is an Exception?
 - Exception Handling
-

What is an Exception?

In this section you will learn about what an exception is and see examples of them.

Definition

An **exception** is an error that occurs during the execution of code. This error causes the code to raise an exception and if not prepared to handle it will halt the execution of the code.

Examples

Run each piece of code and observe the exception raised

In [1]:

```
1/0
```

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
<ipython-input-1-9e1622b385b6> in <module>  
----> 1 1/0
```

ZeroDivisionError: division by zero

ZeroDivisionError occurs when you try to divide by zero.

In [2]:

```
y = a + 5
```

```
-----  
NameError                                         Traceback (most recent call last)  
<ipython-input-2-6ddcec040107> in <module>  
----> 1 y = a + 5
```

NameError: name 'a' is not defined

NameError in this case means that you tried to use the variable a when it was not defined.

In [3]:

```
a = [1, 2, 3]  
a[10]
```

```
-----  
IndexError                                         Traceback (most recent call last)  
<ipython-input-3-3f911ca4e3d3> in <module>  
      1 a = [1, 2, 3]  
----> 2 a[10]
```

IndexError: list index out of range

There are many more exceptions that are built into Python, here is a list of them <https://docs.python.org/3/library/exceptions.html>

Exception Handling

In this section you will learn how to handle exceptions to perform certain tasks and not halt the execution of your code.

Try Except

A try except will allow you to execute code that might raise an exception and in the case of any exception or a specific one we can handle or catch the exception and execute specific code. This will allow us to continue the execution of our program even if there is an exception.

Python tries to execute the code in the try block. In this case if there is any exception raised by the code in the try block it will be caught and the code block in the except block will be executed. After the code that comes after the try except will be executed.

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except:
    # code to execute if there is an exception

# code that will still execute if there is an exception
```

Try Except Example

In this example we are trying to divide a number given by the user, save the outcome in the variable a, and then we would like to print the result of the operation. When taking user input and dividing a number by it there are a couple of exceptions that can be raised. For example if we divide by zero, try running the following block of code, with b as a number an exception will only be Raised if b, is zero

```
In [5]: a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
    print("Success a=",a)
except:
    print("There was an error")
```

There was an error

Try Except Specific

A specific try except allows you to catch certain exceptions and also execute certain code depending on the exception. This is useful if you do not want to deal with some exceptions and the execution should halt, it can also help you find errors in your code that you might not know about, and it can help you differentiate responses to different exceptions. In this case the code after the try except might not run depending on the error.

do not run, just to illustrate

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except (ZeroDivisionError, NameError):
    # code to execute if there is an exception of the given types

# code that will execute if there is no exception or a one that we are handling
```

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError

# code that will execute if there is no exception or a one that we are handling
```

You can also have an empty except at the end to catch an unexpected exception:

do not run, just to illustrate

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if there is any exception

# code that will execute if there is no exception or a one that we are handling
```

Try Except Specific Example

This is the same example as above, but now we will add differentiated messages depending on the exception letting the user know what is wrong with the input.

```
In [15]: a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
    print("Success a=",a)
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
```

You did not provide a number

Try Except Else and Finally

`else` allows one to check if there was no exception when executing the try block. This is useful when we want to execute something only if there were no errors.

do not run, just to illustrate

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if ther is any exception
else:
    # code to execute if there is no exception

# code that will execute if there is no exception or a one that we are handling
```

`finally` allows us to always execute something even if there is an exception or not. This is usually used to signify the end of the try except.

```
In [ ]: # potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if there is any exception
else:
    # code to execute if there is no exception
finally:
    # code to execute at the end of the try except no matter what

# code that will execute if there is no exception or a one that we are handling
```

Try Except Else and Finally Example

You might have noticed that even if there is an error the value of a is always printed. Lets use the else and print the value of a only if there is no error.

```
In [ ]: a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
else:
    print("success a=",a)
```

Now lets let the user know that we are done processing their answer. Using the finally lets add a print.

```
In [ ]: a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
else:
    print("success a=",a)
finally:
    print("Processing Complete")
```


Classes and Objects in Python

Estimated time needed: **40** minutes

Objectives

After completing this lab you will be able to:

- Work with classes and objects
- Identify and define attributes and methods

Table of Contents

- Introduction to Classes and Objects
 - Creating a class
 - Instances of a Class: Objects and Attributes
 - Methods
- Creating a class
- Creating an instance of a class Circle
- The Rectangle Class

Introduction to Classes and Objects

Creating a Class

The first part of creating a class is giving it a name: In this notebook, we will create two classes, Circle and Rectangle. We need to determine all the data that make up that class, and we call that an attribute. Think about this step as creating a blue print that we will use to create objects. In figure 1 we see two classes, circle and rectangle. Each has their attributes, they are variables. The class circle has the attribute radius and color, while the rectangle has the attribute height and width. Let's use the visual examples of these shapes before we get to the code, as this will help you get accustomed to the vocabulary.

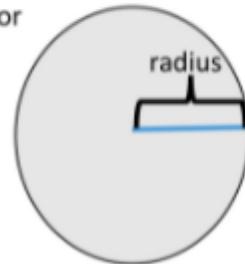
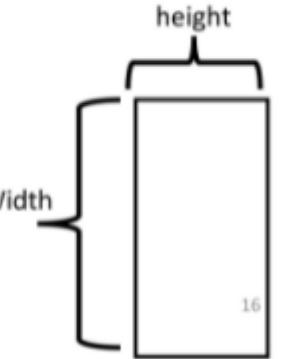
| Class Circle | Class Rectangle |
|--|---|
| Attributes: radius, Color | Attributes: Color, height and Width |
|  <p>Color</p> |  <p>height</p> <p>Width</p> <p>Color</p> <p>16</p> |

Figure 1: Classes circle and rectangle, and each has their own attributes. The class circle has the attribute radius and colour, the rectangle has the attribute height and width.

Instances of a Class: Objects and Attributes

An instance of an object is the realisation of a class, and in Figure 2 we see three instances of the class circle. We give each object a name: red circle, yellow circle and green circle. Each object has different attributes, so let's focus on the attribute of colour for each object.

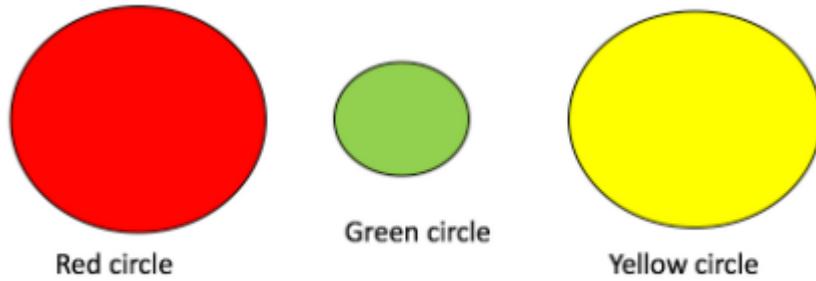


Figure 2: Three instances of the class circle or three objects of type circle.

The colour attribute for the red circle is the colour red, for the green circle object the colour attribute is green, and for the yellow circle the colour attribute is yellow.

Methods

Methods give you a way to change or interact with the object; they are functions that interact with objects. For example, let's say we would like to increase the radius by a specified amount of a circle. We can create a method called **add_radius(r)** that increases the radius by **r**. This is shown in figure 3, where after applying the method to the "orange circle object", the radius of the object increases accordingly. The "dot" notation means to apply the method to the object, which is essentially applying a function to the information in the object.

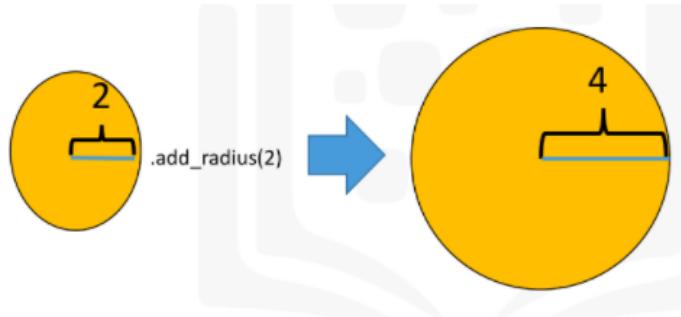


Figure 3: Applying the method "add_radius" to the object orange circle object.

Creating a Class

Now we are going to create a class circle, but first, we are going to import a library to draw the objects:

```
In [1]: # Import the library
import matplotlib.pyplot as plt
%matplotlib inline
```

The first step in creating your own class is to use the class keyword, then the name of the class as shown in Figure 4. In this course the class parent will always be object:

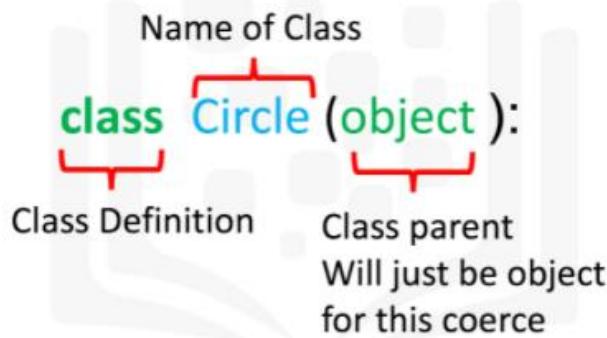


Figure 4: Creating a class Circle.

The next step is a special method called a `constructor __init__`, which is used to initialize the object. The input are data attributes. The term `self` contains all the attributes in the set. For example the `self.color` gives the value of the attribute `color` and `self.radius` will give you the radius of the object. We also have the method `add_radius()` with the parameter `r`, the method adds the value of `r` to the attribute `radius`. To access the radius we use the syntax `self.radius`. The labeled syntax is summarized in Figure 5:

```
class Circle(object):
```

} Define your class

```
    def __init__(self, radius, color):
```

} Data attributes used to
initialize object

```
        self.radius = radius;
```

```
        self.color = color;
```

} Method used to add r
to radius

```
    def add_radius(self, r):
```

```
        self.radius = self.radius + r
```

```
        return (self.radius)
```

Figure 5: Labeled syntax of the object circle.

The actual object is shown below. We include the method `drawCircle` to display the image of a circle. We set the default radius to 3 and the default colour to blue:

In [2]:

```
# Create a class Circle

class Circle(object):

    # Constructor
    def __init__(self, radius=3, color='blue'):
        self.radius = radius
        self.color = color

    # Method
    def add_radius(self, r):
        self.radius = self.radius + r
        return(self.radius)

    # Method
    def drawCircle(self):
        plt.gca().add_patch(plt.Circle((0, 0), radius=self.radius, fc=self.color))
        plt.axis('scaled')
        plt.show()
```

Creating an instance of a class Circle

Let's create the object RedCircle of type Circle to do the following:

```
In [3]: # Create an object RedCircle  
  
RedCircle = Circle(10, 'red')
```

We can use the dir command to get a list of the object's methods. Many of them are default Python methods.

```
In [4]: # Find out the methods can be used on the object RedCircle  
  
dir(RedCircle)
```

```
Out[4]: ['__class__',  
         '__delattr__',  
         '__dict__',  
         '__dir__',  
         '__doc__',  
         '__eq__',  
         '__format__',  
         '__ge__',  
         '__getattribute__',  
         '__gt__',  
         '__hash__',  
         '__init__',  
         '__init_subclass__',  
         '__le__',  
         '__lt__',  
         '__module__',  
         '__ne__',  
         '__new__',  
         '__reduce__',  
         '__reduce_ex__',  
         '__repr__',  
         '__setattr__',  
         '__sizeof__',  
         '__str__',  
         '__subclasshook__',  
         '__weakref__',  
         'add_radius',  
         'color',  
         'drawCircle',  
         'radius']
```

We can look at the data attributes of the object:

```
In [5]: # Print the object attribute radius  
RedCircle.radius
```

```
Out[5]: 10
```

```
In [6]: # Print the object attribute color  
RedCircle.color
```

```
Out[6]: 'red'
```

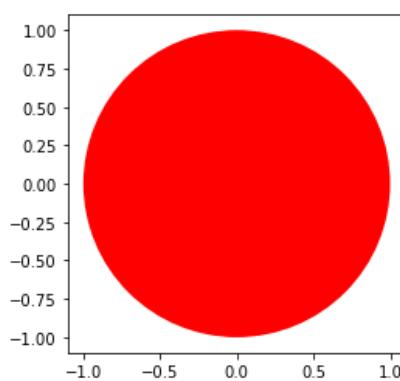
We can change the object's data attributes:

```
In [11]: # Set the object attribute radius  
RedCircle.radius = 1  
RedCircle.radius
```

```
Out[11]: 1
```

We can draw the object by using the method `drawCircle()`:

```
In [12]: # Call the method drawCircle  
RedCircle.drawCircle()
```



We can increase the radius of the circle by applying the method `add_radius()`. Let increases the radius by 2 and then by 5:

```
In [13]: # Use method to change the object attribute radius

print('Radius of object:',RedCircle.radius)
RedCircle.add_radius(2)
print('Radius of object of after applying the method add_radius(2):',RedCircle.radius)
RedCircle.add_radius(5)
print('Radius of object of after applying the method add_radius(5):',RedCircle.radius)
RedCircle.add_radius(6)
print('Radius of object of after applying the method add radius(6):',RedCircle.radius)

Radius of object: 1
Radius of object of after applying the method add_radius(2): 3
Radius of object of after applying the method add_radius(5): 8
Radius of object of after applying the method add radius(6): 14
```

Let's create a blue circle. As the default colour is blue, all we have to do is specify what the radius is:

```
In [14]: # Create a blue circle with a given radius

BlueCircle = Circle(radius=100)
```

As before we can access the attributes of the instance of the class by using the dot notation:

```
In [15]: # Print the object attribute radius

BlueCircle.radius
```

```
Out[15]: 100
```

```
In [16]: # Print the object attribute color

BlueCircle.color
```

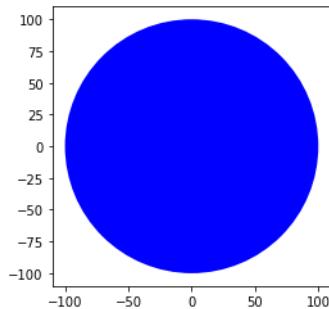
```
Out[16]: 'blue'
```

We can draw the object by using the method `drawCircle()`:

We can draw the object by using the method `drawCircle()` :

```
In [17]: # Call the method drawCircle

BlueCircle.drawCircle()
```



Compare the x and y axis of the figure to the figure for RedCircle; they are different.

The Rectangle Class

Let's create a class rectangle with the attributes of height, width and color. We will only add the method to draw the rectangle object:

```
In [18]: # Create a new Rectangle class for creating a rectangle object

class Rectangle(object):

    # Constructor
    def __init__(self, width=2, height=3, color='r'):
        self.height = height
        self.width = width
        self.color = color

    # Method
    def drawRectangle(self):
        plt.gca().add_patch(plt.Rectangle((0, 0), self.width, self.height ,fc=self.color))
        plt.axis('scaled')
        plt.show()
```

Let's create the object SkinnyBlueRectangle of type Rectangle. Its width will be 2 and height will be 3, and the color will be blue:

```
In [19]: # Create a new object rectangle

SkinnyBlueRectangle = Rectangle(2, 10, 'blue')
```

As before we can access the attributes of the instance of the class by using the dot notation:

```
In [20]: # Print the object attribute height  
  
SkinnyBlueRectangle.height
```

```
Out[20]: 10
```

```
In [21]: # Print the object attribute width  
  
SkinnyBlueRectangle.width
```

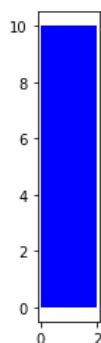
```
Out[21]: 2
```

```
In [22]: # Print the object attribute color  
  
SkinnyBlueRectangle.color
```

```
Out[22]: 'blue'
```

We can draw the object:

```
In [23]: # Use the drawRectangle method to draw the shape  
  
SkinnyBlueRectangle.drawRectangle()
```



Let's create the object FatYellowRectangle of type Rectangle :

```
In [24]: # Create a new object rectangle  
  
FatYellowRectangle = Rectangle(20, 5, 'yellow')
```

We can access the attributes of the instance of the class by using the dot notation:

```
In [25]: # Print the object attribute height  
  
FatYellowRectangle.height
```

```
Out[25]: 5
```

```
In [26]: # Print the object attribute width  
  
FatYellowRectangle.width
```

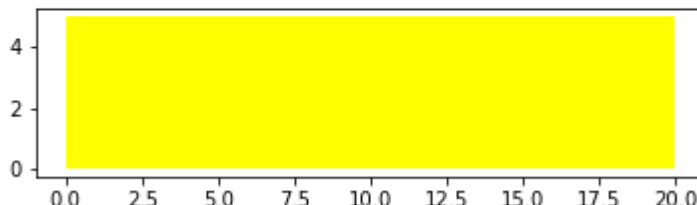
```
Out[26]: 20
```

```
In [27]: # Print the object attribute color  
  
FatYellowRectangle.color
```

```
Out[27]: 'yellow'
```

We can draw the object:

```
In [28]: # Use the drawRectangle method to draw the shape  
  
FatYellowRectangle.drawRectangle()
```



Exercises

Text Analysis

You have been recruited by your friend, a linguistics enthusiast, to create a utility tool that can perform analysis on a given piece of text. Complete the class 'analysedText' with the following methods -

- Constructor - Takes argument 'text', makes it lower case and removes all punctuation. Assume only the following punctuation is used - period (.), exclamation mark (!), comma (,) and question mark (?). Store the argument in "fmtText"
- freqAll - returns a dictionary of all unique words in the text along with the number of their occurrences.
- freqOf - returns the frequency of the word passed in argument.

The skeleton code has been given to you. Docstrings can be ignored for the purpose of the exercise.

Hint: Some useful functions are replace(), lower(), split(), count()

```
In [66]: class analysedText(object):

    def __init__(self, text):
        reArrText = text.lower()
        reArrText = reArrText.replace('.','').replace('!','').replace(',','').replace('?','')
        self.fmtText = reArrText

    def freqAll(self):
        wordList = self.fmtText.split(' ')
        freqMap = {}
        for word in set(wordList): # use set to remove duplicates in list
            freqMap[word] = wordList.count(word)

        return freqMap

    def freqOf(self,word):
        freqDict = self.freqAll()

        if word in freqDict:
            return freqDict[word]
        else:
            return 0
```

Execute the block below to check your progress.

```
In [67]:
import sys

sampleMap = {'eirmod': 1, 'sed': 1, 'amet': 2, 'diam': 5, 'consetetur': 1, 'labore': 1, 'tempor': 1, 'dolor': 1, 'magna': 2, 'et': 3, 'nonumy': 1, 'ipsu

def testMsg(passed):
    if passed:
        return 'Test Passed'
    else :
        return 'Test Failed'

print("Constructor: ")
try:
    samplePassage = analysedText("Lorem ipsum dolor! diam amet, consetetur Lorem magna. sed diam nonumy eirmod tempor. diam et labore? et diam magna. e
    print(testMsg(samplePassage.fmtText == "lorem ipsum dolor diam amet consetetur lorem magna sed diam nonumy eirmod tempor diam et labore et diam mag
except:
    print("Error detected. Recheck your function " )
print("freqAll: ")
try:
    wordMap = samplePassage.freqAll()
    print(testMsg(wordMap==sampleMap))
except:
    print("Error detected. Recheck your function " )
print("freqOf: ")
try:
    passed = True
    for word in sampleMap:
        if samplePassage.freqOf(word) != sampleMap[word]:
            passed = False
            break
    print(testMsg(passed))
except:
    print("Error detected. Recheck your function " )

Constructor:
Test Passed
freqAll:
Test Passed
freqOf:
Test Passed
```

Click here for the solution ````python class analysedText(object): def __init__(self, text): # remove punctuation formattedText = text.replace('.','').replace('!','').replace('?','').replace(',') # make text lowercase formattedText = formattedText.lower() self.fmtText = formattedText def freqAll(self): # split text into words wordList = self.fmtText.split(' ') # Create dictionary freqMap = {} for word in set(wordList): # use set to remove duplicates in list freqMap[word] = wordList.count(word) return freqMap def freqOf(self,word): # get frequency map freqDict = self.freqAll() if word in freqDict: return freqDict[word] else: return 0 ````

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

Exception Handling

Objectives

After watching this video, you will be able to:

- Explain Exception Handling
- Demonstrate the use of exception handling
- Understand the basics of exception handling

Exception Handling

Try...Except Statement

Please enter your name: Please only enter letters

try:

except:

Try...Except Statement



Try...Except Statement

```
try:  
    myfile = open("myfile", "r")  
    myfile.write("File for exception  
handling.")  
except IOError:  
    print("Unable to open or read the data  
in the file.")
```

Try...Except Statement

Unable to open or read the data in the file.

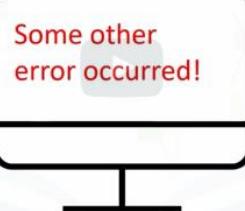
Try...Except Statements

```
try:  
    myfile = open("myfile", "r")  
    myfile.write("My file for exception handling.")  
except IOError:  
    print("Unable to open or read the data in the file.")  
except:  
    print("Some other error occurred!")
```

if another error occurs and not seen by IOError, we add another except statement

Try...Except...Else Statement

Try...Except Statements



```
try:  
    myfile = open("myfile", "r")  
    myfile.write("My file for exception handling.")  
except IOError:  
    print("Unable to open or read the data in the file.")  
else:  
    print("The file was written successfully")
```

Try...Except...Else Statement

The file was written successfully.

Try...Except...Else...Finally Statement

```
try:  
    myfile = open("myfile", "r")  
    myfile.write("My file for exception  
handling.")  
except IOError:  
    print("Unable to open or read the data in  
the file.")  
else:  
    print("The file was written successfully")
```

To Close the file: add "finally" statement

Try...Except...Else...Finally Statement

```
try:  
    myfile = open("myfile", "r")  
    myfile.write("My file for exception  
handling.")  
except IOError:  
    print("Unable to open or read the data in  
the file.")  
else:  
    print("The file was written successfully")  
finally:  
    myfile.close()  
    print("File is now closed.")
```

Practice Quiz: Exception Handling

Bookmark

Question 1

1/1 point (ungraded)

Why do we use exception handlers?

Terminate a program

Catch errors within a program

Write a file

Read a file



Submit You have used 1 of 2 attempts

Question 2

1/1 point (ungraded)

What is the purpose of a try...except statement?

Only executes if one condition is true

Crash a program when errors occur

Executes the code block only if a certain condition exists

Catch and handle exceptions when an error occurs



Submit You have used 1 of 2 attempts

Objects and Classes

In this module, we're going to talk about objects and classes. Python has many different kinds of data types: integers, floats, strings, lists, dictionaries, booleans.

In Python, each is an object. Every object has the following: a type, internal representation, a set of functions called methods to interact with the data.

An object is an instance of a particular type. For example, we have two types, type one and type two. We can have several objects of type one as shown in yellow.

Each object is an instance of type one. We also have several objects of type two shown in green. Each object is an instance of type two.

Let's do several less abstract examples. Every time we create an integer, we are creating an instance of type integer, or we are creating an integer object.

In this case, we are creating five instances of type integer or five integer objects.

Similarly, every time we create a list, we are creating an instance of type list, or we are creating a list object. In this case, we are creating five instances of type list or five list objects.

We could find out the type of an object by using the type command. In this case, we have an object of type list, we have an object of type integer,

we have an object of type string. Finally, we have an object of type dictionary.

A class or type's methods are functions that every instance of that class or type provides.

It's how you interact with the object. We have been using methods all this time, for example, on lists. Sorting is an example of a method that interacts with the data in the object.

Consider the list ratings, the data is a series of numbers contained within the list.

The method sort will change the data within the object. We call the method by adding a period at the end of the object's name, and the method's name we would like to call with parentheses.

We have the rating's list represented in orange. The data contained in the list is a sequence of numbers.

We call the sort method, this changes the data contained in the object.

You can say it changes the state of the object. We can call the reverse method on the list, changing the list again. We call the method, reversing the order of the sequence within the object. In many cases, you don't have to know the inner workings of the class and its methods, you just have to know how to use them.

Next, we will cover how to construct your own classes.

You can create your own type or class in Python. In this section, you'll create a class.

The class has data attributes. The class has methods. We then create instances or instances of that class or objects.

The class data attributes define the class. Let's create two classes. The first class will be a circle, the second will be a rectangle. Let's think about what constitutes a circle.

Examining this image, all we need is a radius to define a circle, and let's add color to make it easier to distinguish between different instances of the class later. Therefore, our class data attributes are radius and color. Similarly, examining the image in order to define a rectangle,

we need the height and width. We will also add color to distinguish between instances later.

Therefore, the data attributes are color, height, and width. To create the class circle, you will need to include the class definition. This tells Python you're creating your own class, the name of the class. For this course in parentheses, you will always place the term object, this is the parent of the class. For the class rectangle, we changed the name of the class, but the rest is kept the same. Classes are outlines we have to set the attributes to create objects.

We can create an object that is an instance of type circle. The color data attribute is red, and the data attribute radius is four. We could also create a second object that is an instance of type circle. In this case, the color data attribute is green, and the data attribute radius is two.

We can also create an object that is an instance of type rectangle. The color data attribute is blue, and the data attribute of height and width is two. The second object is also an instance of type rectangle. In this case, the color data attribute is yellow, and the height is one, and the width is three. We now have different objects of class circle or type circle.

We also have different objects of class rectangle or type rectangle. Let us continue building the circle class in Python. We define our class. We then initialize each instance of the class with data attributes, radius, and color using the class constructor.

The function `init` is a constructor. It's a special function that tells Python you are making a new class. There are other special functions in Python to make more complex classes.

The radius and color parameters are used to initialize the radius and color data attributes of the class instance. The `self` parameter refers to the newly created instance of the class.

The parameters, radius, and color can be used in the constructors body to access the values passed to the class constructor when the class is constructed.

We could set the value of the radius and color data attributes to the values passed to the constructor method. Similarly, we can define the class rectangle in Python.

The name of the class is different. This time, the class data attributes are color, height, and width.

After we've created the class, in order to create an object of class circle, we introduce a variable. This will be the name of the object. We create the object by using the object constructor.

The object constructor consists of the name of the class as well as the parameters. These are the data attributes. When we create a circle object, we call the code like a function.

The arguments passed to the circle constructor are used to initialize the data attributes of the newly created circle instance. It is helpful to think of self as a box that contains all the data attributes of the object.

Typing the object's name followed by a dot and the data attribute name gives us the data attribute value, for example, radius. In this case, the radius is 10.

We can do the same for color. We can see the relationship between the self parameter and the object. In Python, we can also set or change the data attribute directly.

Typing the object's name followed by a dot and the data attribute name, and set it equal to the corresponding value. We can verify that the color data attribute has changed.

Usually, in order to change the data in an object, we define methods in the class.

Let's discuss methods. We have seen how data attributes consist of the data defining the objects. Methods are functions that interact and change the data attributes, changing or using the data attributes of the object.

Let's say we would like to change the size of a circle. This involves changing the radius attribute.

We add a method, add radius to the class circle. The method has a function that requires the self as well as other parameters. In this case, we are going to add a value to the radius,

We denote that value as r. We are going to add r to the data attribute radius. Let's see how this part of the code works when we create an object and call the add_radius method.

As before, we create an object with the object constructor. We pass two arguments to the constructor. The radius is set to two and the color is set to red. In the constructor's body,

the data attributes are set. We can use the box analogy to see the current state of the object.

We call the method by adding a dot followed by the method, name, and parentheses. In this case, the argument of the function is the amount we would like to add.

We do not need to worry about the self parameter when calling the method. Just like with the constructor, Python will take care of that for us. In many cases, there may not be

any parameters other than self specified in the method's definition. So we don't pass any arguments when calling the function. Internally, the method is called with a value of eight, and the proper self object. The method assigns a new value to self radius.

This changes the object, in particular, the radius data attribute.

When we call the add_radius method, this changes the object by changing the value of the radius data attribute. We can add default values to the parameters of a class as constructor.

In the labs, we also create the method called drawCircle. See the lab for the implementation of drawCircle. In the labs, we can create a new object of type circle using the constructor.

The color will be red and the radius will be three. We can access the data attribute radius.

We can access the attribute color. Finally, we can use the method drawCircle to draw the circle. Similarly, we can create a new object of type circle. We can access the data attribute of radius.

We can access the data attribute color. We can use the method drawCircle to draw the circle.

In summary, we have created an object of class circle called RedCircle with a radius attribute of three,

and a color attribute of red. We also created an object of class circle called BlueCircle, with a radius attribute of 10 and a color attribute of blue. In the lab, we have a similar class for rectangle. We can create a new object of type rectangle using the constructor.

We can access a data attribute of height. We can also access the data attribute of width.

We could do the same for the data attribute of color. We can use the method drawRectangle to draw the rectangle. So we have a class, an object that is a realization or instantiation of that class.

For example, we can create two objects of class Circle, or two objects of class Rectangle.

The dir function is useful for obtaining the list of data attributes and methods associated with a class. The object you're interested in is passed as an argument. The return value is a list of the objects data attributes.

The attribute surrounded by underscores are for internal use, and you shouldn't have to worry about them. The regular looking attributes are the ones you should concern yourself with. These are the objects, methods, and data attributes.

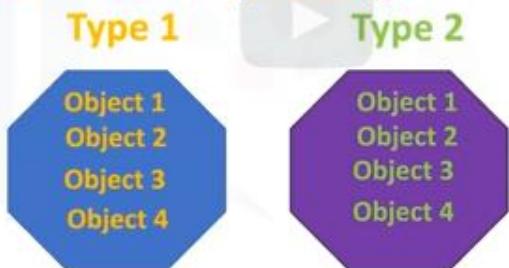
There is a lot more you can do with objects in Python.

Python

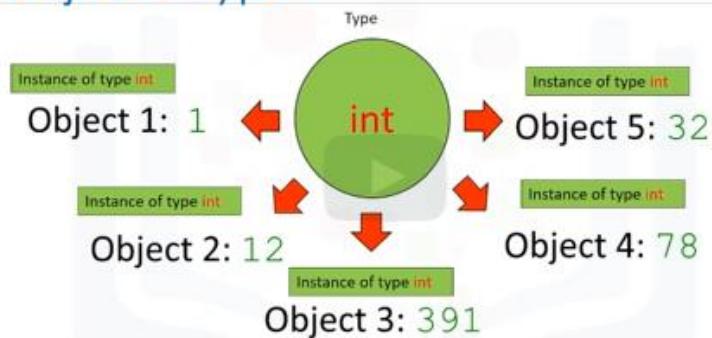
Objects and Classes

Built-in Types in Python

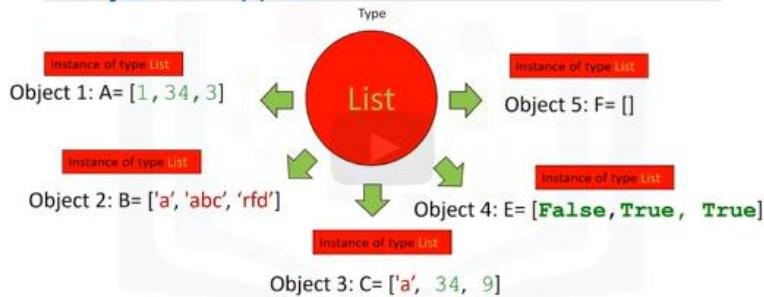
- every **object** has:
 - a **type**
 - an internal data representation (a blueprint)
 - a set of procedures for interacting with the object (**methods**)
- an **object** is an **instance** of a particular **type**



Objects: Type

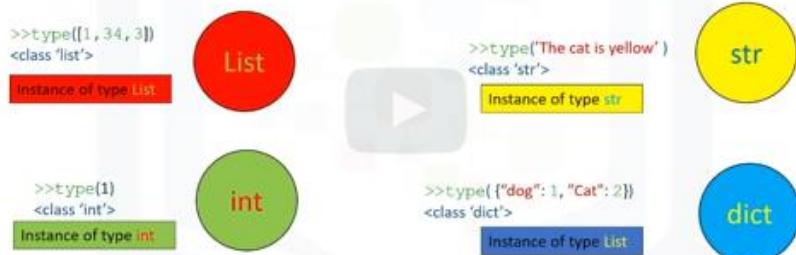


Objects: Type



Objects: Type

- You can find the type of a object by using the command `type()`



Methods

- A class or type's methods are functions that every instance of that class or type provides
- It's how you interact with the data in a object
- Sorting is an example of a method that interacts with the data in the object

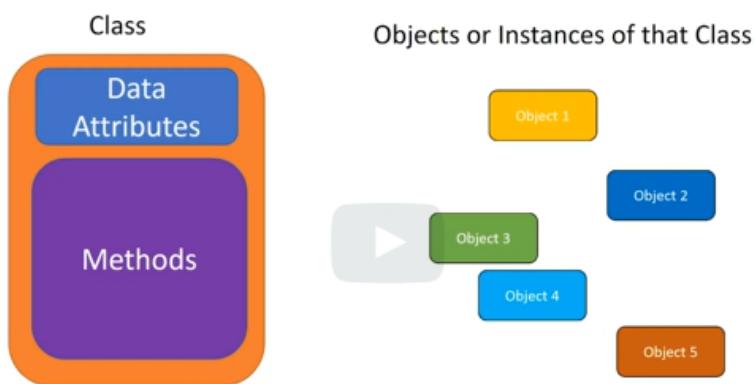
Ratings=[**10, 9, 6, 5, 10, 8, 9, 6, 2**]

Ratings.sort()

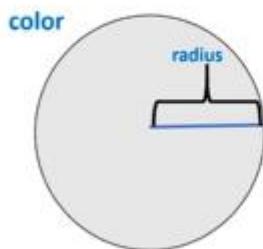




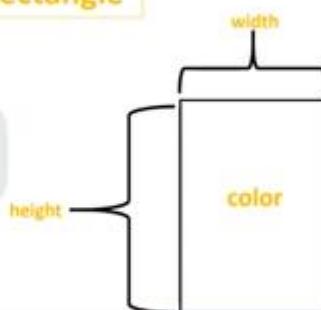
Creating Your Own Types: Defining Classes



Class Circle



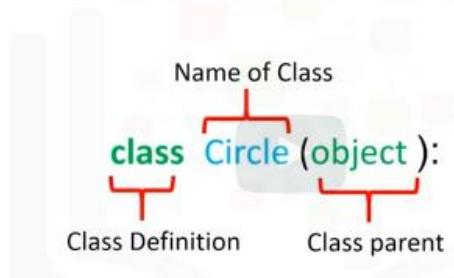
Class Rectangle



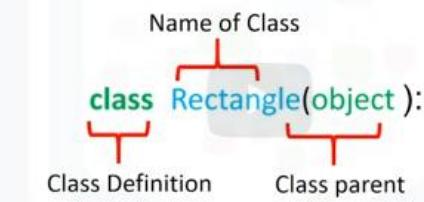
Data Attributes: radius, color

Data Attributes: color, height and width

Create a class: Circle



Create a class: Circle



Defining a Class

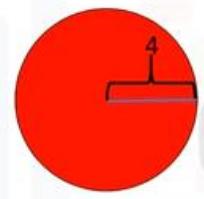
`class Circle(object):`



`class Rectangle(object):`



Attributes and Objects



Object 1: Instance of type Circle

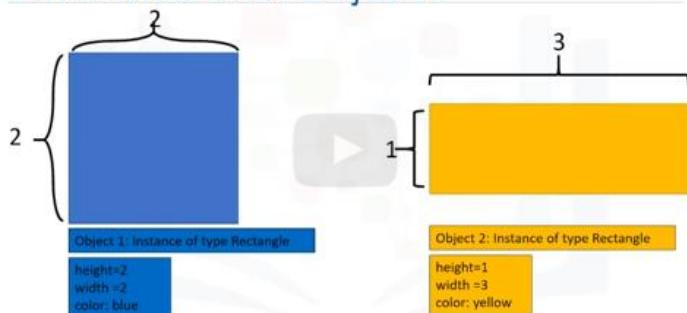
Data Attributes:
radius=4
color:red



Object 2: Instance of type Circle

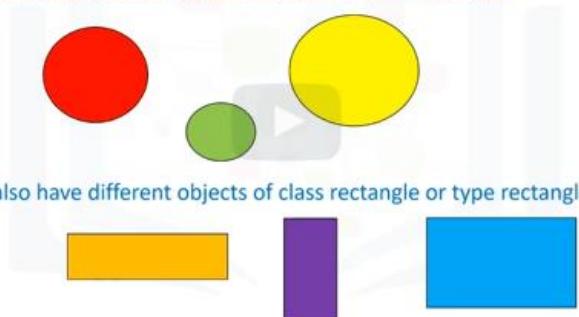
Data Attributes:
radius=2
color:green

Attributes and Objects



Instances of a Class: objects

- We now have different objects of class circle or type circle



- We also have different objects of class rectangle or type rectangle

Create a class: Circle

```
class Circle(object):  
    def __init__(self, radius, color):  
        self.radius = radius;  
        self.color = color;
```

Define your class

Data attributes used to Initialize each instance of the class

Create a class: Circle

special method or constructor used to initialize data attributes

```
def __init__(self, radius, color):
```

parameters

```
    self.radius = radius;  
    self.color = color;
```

Create a class: Rectangle

```
class Rectangle(object):
```

Define your class

```
def __init__(self, color, height, width):
```

```
    self.height = height;  
    self.width = width  
    self.color = color;
```

Initialize the object's
Data attributes

Create an Instance of a Class: Circle

How to create an object of class circle:

Name of Object

RedCircle =Circle (10, "red")

Create an Instance of a Class: Circle

How to create an object of class circle:

Name of Class

RedCircle =Circle (10, "red")

Attributes

Create an Instance of a Class: Circle

C1=Circle (10,'red')

```
class Circle(object):
```

```
    def __init__(self, 10, 'red'):  
        self.radius = 10;  
        self.color = 'red';
```



self.radius = 10;
self.color = 'red';

Create an Instance of a Class: Circle

How to create an object of class circle:

RedCircle =Circle (10, "red")

Object Constructor

Create an Instance of a Class: Circle

C1=Circle (10,'red')

```
class Circle(object):
```

```
    def __init__(self, radius, color):  
        self.radius = radius;  
        self.color = color;
```

Create an Instance of a Class: Circle

```
class Circle(object):
```

```
    def __init__(self, radius, color):  
        self.radius = radius;  
        self.color = color;
```

self
self.radius = 10
self.color ='red'

Create an Instance of a Class: Circle

C1=Circle (10, "red")

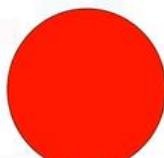
C1.radius

10

C1.color

"red"

self
self .radius
self. color



self.radius = 10;
self.color = 'red';

Create an Instance of a Class: Circle

C1=Circle (10, "red")

C1.radius

10

C1.color

"red"

C1
C1.radius = 10
C1.color = 'red'



self.radius = 10;
self.color = 'red';

Create an Instance of a Class: Circle

C1=Circle (10, "red")

C1.color="blue"

C1.color

"blue"

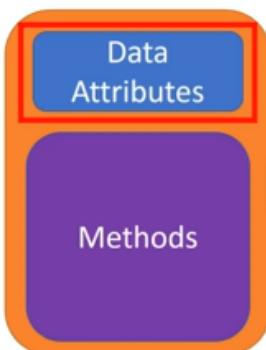


self.radius = 10;
self.color = 'red';

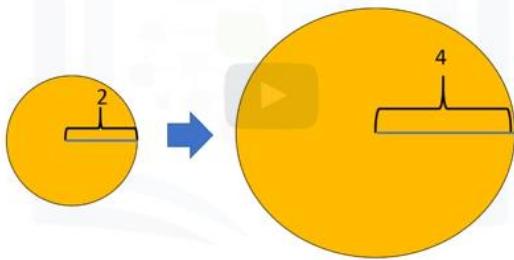
Methods

Class

Objects or Instances of that Class



Methods



Create a class: Circle

```
class Circle (object):  
    def __init__(self, radius , color):  
        self.radius = radius;  
        self.color = color;  
  
    def add_radius(self,r):  
        self.radius= self.radius +r
```

Method used to add r to radius

Create an instance of a class: Circle

C1=Circle (2, 'red')

```
def __init__(self, 2, 'red'):-----  
    self .radius = 2;.....  
    self. color = 'red';
```

Create an instance of a class: Circle

C1=Circle (2, 'red')

self .radius = 2
self. color ='red'

Create an instance of a class: Circle

C1=Circle (2, 'red')

self .radius = 2
self. color ='red'

```
def add_radius(self,r):  
    self.radius= self.radius +r  
    return (self.radius)
```

Create an instance of a class: Circle

C1=Circle (2, 'red')

self .radius = 2
self. color ='red'

C1.add_radius(8)

```
def add_radius(self,8):  
    self.radius= 2 +8  
    return (10)
```

Create an instance of a class: Circle

C1=Circle (2, 'red')

C1.add_radius(8)

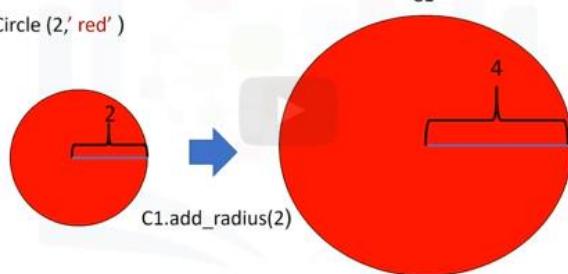
self .radius = 2
self. color ='red'

```
def add_radius(self,8):  
    self.radius= 2 +8  
    return (10)
```

self .radius = 10
self. color ='red'

Methods

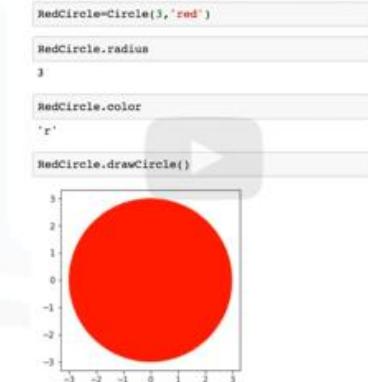
C1=Circle (2, 'red')



Creating an instance of the class: Circle

Create a class: Circle

```
class Circle(object):  
    def __init__(self, radius=3, color='red'): } We can add  
        self.radius = radius;  
        self.color = color;  
  
    def add_radius(self,r):  
        self.radius= self.radius +r  
  
    def drawCircle(self): } New method
```

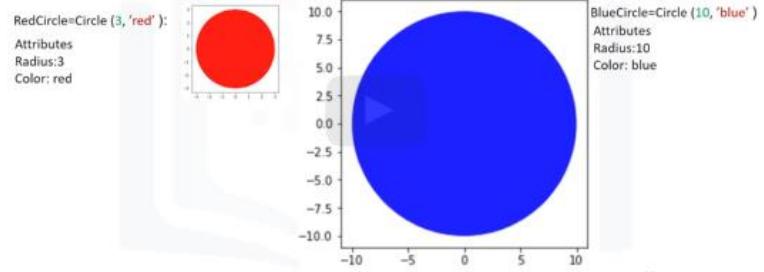


Instance is like the details example: color = red radius = 2

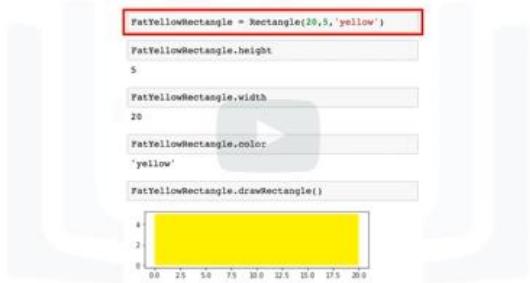
Creating a new instance of the class: Circle



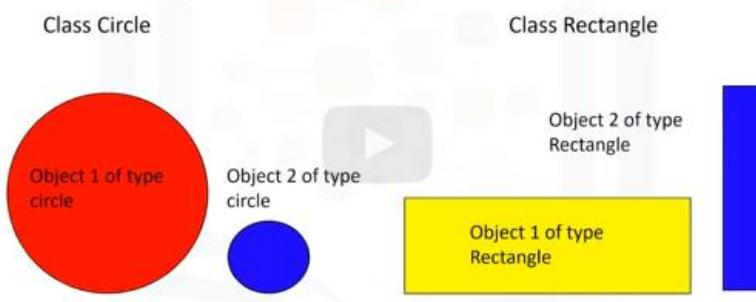
Two instances of the class: Circle



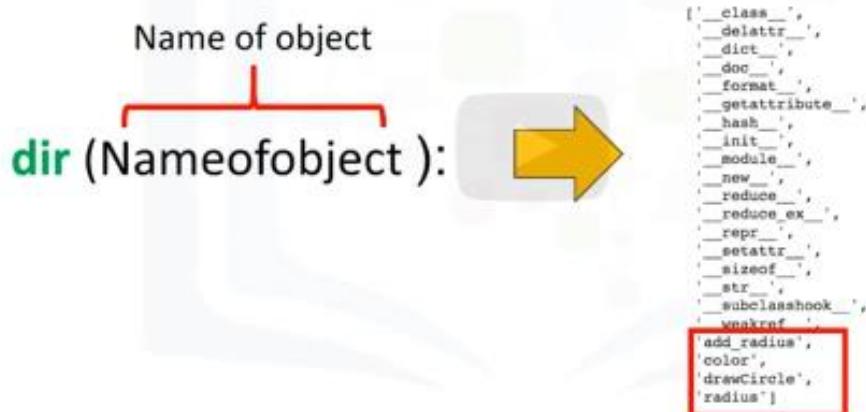
Creating an Instance of the Class: Rectangle



Review



Create a class: Circle



Practice Quiz: Objects and Classes

 [Bookmark this page](#)

Question 1

1/1 point (ungraded)

What is the type of the following?

["a"]

list

str



Answer

Correct: Correct, the "a" is surrounded by brackets so it is a list.

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 2

1/1 point (ungraded)

What does a method do to an object?

Returns a new values.

Changes or interacts with the object.



Answer

Correct: Correct.

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 3

1/1 point (ungraded)

We create the object:

Circle(3,'blue')

What is the color attribute set to?

'blue'

2



Answer

Correct: Correct.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 4

1/1 point (ungraded)

What is the radius attribute after the following code block is run?

RedCircle=Circle(10,'red')

RedCircle.radius=1

'red'

1

10



Answer

Correct: Correct, the line of code RedCircle.radius=1 will change the radius.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 5

1/1 point (ungraded)

>>What is the radius attribute after the following code block is run?

```
BlueCircle=Circle(10,'blue')
```

```
BlueCircle.add_radius(20)
```

30

20

10



Answer

Correct: Correct.

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Graded Quiz: Python Programming Fundamentals

Bookmarked

Graded Quiz due Jan 24, 2022 14:15 +08

Question 2

1/1 point (graded)

Question 1

1/1 point (graded)

What value of **x** will produce the output?

Hello

Mike

```
1 x=
2
3 ↘ if(x=="A"):
4
5   | print('Hello')
6
7 ↘ else:
8
9   | print('Hi')
10
11  print('Mike')
```

```
1 A=[1,2,3]
2
3 for a in A:
4
5   | print(2*a)
```

Y

- 11
- 22
- 33

x=A

x="A"

x=1



- 2
- 4
- 6



You have used 1 of 2 attempts

You have used 1 of 1 attempt

Question 3

1/1 point (graded)

Consider the function step, when will the function return a value of 1?

```
1 ✓ def step(x):
2 ✓     if x>0:
3         |   y=1
4 ✓     else:
5         |   y=0
6     return y
```

if x is equal to or less than zero

if x is larger than 0

if x is less than zero



Answer

Correct: correct, the value of y is 1 only if x is larger than 0

Question 4

1/1 point (graded)

Why is the "finally" statement used?

Only execute the remaining code if an error occurs

Execute the remaining code no matter the end result

Only execute the remaining code if one condition is false



Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 5

1/1 point (graded)

Consider the class Points, what are the data attributes?

```
1  class Point(object):
2
3      def __init__(self,x,y):
4
5          self.x=x
6          self.y=y
7
8      def print_point(self):
9
10         print('x=',self.x,'y=',self.y)
11
```

__init__

self.x self.y

print_point



Answer

Correct: correct

You have used 1 of 2 attempts

Correct (1/1 point)

<https://github.com/brendensong/IBM-Data-Science-Professional-Certificate/wiki/4.1.Python-Basics>

Module 4 - Working with Data in Python

-  [Module Introduction and Learning Objectives](#) 1 min
-  [Video: Reading Files with Open \(3:43\)](#) 4 min
-  [Hands-On Lab: Reading Files with Open](#) 1 min + 1 activity
-  [Video: Writing Files with Open \(2:54\)](#) 3 min
-  [Hands-On Lab: Writing Files with Open](#) 1 min + 1 activity
-  [Practice Quiz: Reading & Writing Files with Open](#) 1 activity
-  [Video: Loading Data with Pandas](#) 4 min
-  [Video: Pandas: Working with and Saving Data \(2:06\)](#) 3 min
-  [Hands-on Lab: Pandas with IBM Watson Studio](#) 1 min
-  [Practice Quiz: Pandas](#) 1 activity
-  [Video: One Dimensional NumPy \(11:23\)](#) 12 min
-  [Hands-On Lab: One Dimensional Numpy](#) 1 min + 1 activity
-  [Video: Two Dimensional NumPy \(7:13\)](#) 8 min
-  [Hands-On Lab: Two Dimensional Numpy](#) 1 min + 1 activity
-  [Practice Quiz: Numpy in Python](#) 1 activity
-  [Graded Quiz: Working with Data in Python \(5 Questions\)](#) 1 activity

Graded Quiz due Jan 27, 2022, 3:21 AM GMT+8

Module Introduction and Learning Objectives

[Bookmark this page](#)

This module explains the basics of working with data in Python and begins the path with learning how to read and write files.

Continue the module and uncover the best Python libraries that will aid in data manipulation and mathematical operations.

Learning Objectives

In this lesson you will learn about:

- Demonstrate an open function to create and identify a file object.
- Use pandas for library and data analysis by using commands.
- Demonstrate how to create a text file by using the open function.
- Demonstrate how to use NumPy to create multi-dimensional arrays.

Reading Files with Open

In this section, we will use Python's built-in open function to create a file object, and obtain the data from a "txt" file. We will use Python's open function to get a file object.

We can apply a method to that object to read data from the file. We can open the file, Example1.txt, as follows. We use the open function. The first argument is the file path.

This is made up of the file name, and the file directory. The second parameter is the mode.

Common values used include 'r' for reading, 'w' for writing, and 'a' for appending.

We will use 'r' for reading. Finally, we have the file object. We can now use the file object to obtain information about the file. We can use the data attribute name to get the name of the file.

The result is a string that contains the name of the file. We can see what mode the object is in using the data attribute mode, and 'r' is shown representing read. You should always close the file object using the method close. This may get tedious sometimes, so let's use the "with" statement. Using a "with" statement to open a file is better practice because it automatically closes the file.

The code will run everything in the indent block, then closes the file. This code reads the file, Example1.txt.

We can use the file object, "File1." The code will perform all operations in the indent block then close the file at the end of the indent. The method "read" stores the values of the file in the variable "file_stuff" as a string. You can print the file content. You can check if the file content is closed, but you cannot read from it outside the indent. But you can print the file content outside the indent as well.

We can print the file content. We will see the following. When we examine the raw string, we will see the ". ". This is so Python knows to start a new line. We can output every line as an element in a list using the method "readlines." The first line corresponds to the first element in the list.

The second line corresponds to the second element in the list, and so on. We can use the method "readline" to read the first line of the file. If we run this command, it will store the first line in the variable "file_stuff" then print the first line. We can use the method "readline" twice.

The first time it's called, it will save the first line in the variable "file_stuff," and then print the first line. The second time it's called, it will save the second line in the variable "file_stuff," and then print the second line. We can use a loop to print out each line individually as follows. Let's represent every character in a string as a grid.

We can specify the number of characters we would like to read from the string as an argument to the method "readlines." When we use a four as an argument in the method "readlines," we print out the first four characters in the file. Each time we call the method, we will progress through the text. If we call a method with the arguments 16, the first 16 characters are printed out, and then the new line.

If we call the method a second time, the next five characters are printed out. Finally, if we call the method the last time with the argument nine, the last nine characters are printed out.

Check out the labs for more examples of methods and other file types.

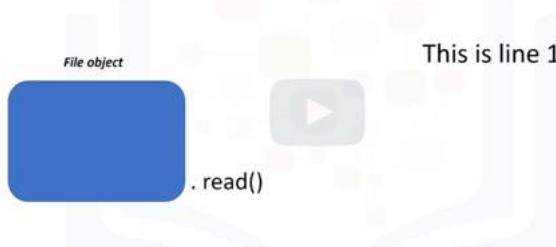
Python

Reading Files with Open

File Object



File Object



```
File1 = open("/resources/data/Example2.txt","w")
```

Open function

```
File1 = open("/resources/data/Example2.txt","w")
```

File Path

File name

```
File1 = open("/resources/data/Example2.txt","w")
```

Directory

```
File1 = open("/resources/data/Example2.txt","w")
```

Mode

File object

```
File1 = open("/resources/data/Example2.txt","w")
```

```
File1  
'/resources/data/Example1.txt'  
'r'
```

```
File1.name  
'/resources/data/Example1.txt'  
File1.mode  
'r'  
File1.close()
```

```
with open("Example1.txt","r") as File1:
```

```
    file_stuff=File1.read()
```

```
    print(file_stuff)
```

```
    print(File1.closed)
```

```
    print(file_stuff)
```

Using a "with" statement to open a file is better practice because it automatically closes the file.

```
print(file_stuff)
```

```
This is line 1
```

```
This is line 2
```

```
This is line 3
```

```
file_stuff:
```

```
This is line 1 \n This is line 2 \n This is line 3
```

*Python starts a new line with \n

```
with open("Example1.txt","r") as File1:  
    file_stuff[0]  
    file_stuff=File1.readlines()  
    print(file_stuff)  
        This is line 1  
            file_stuff[1]  
                This is line 2  
                    file_stuff[2]  
                        This is line 3  
file_stuff: ['This is line 1\n', 'This is line 2\n', 'This is line 3']
```

```
with open("Example1.txt","r") as File1:  
    file_stuff=File1.readline()  
    print(file_stuff)  
    file_stuff=File1.readline()  
    print(file_stuff)  
        This is line 1  
            This is line 2
```

```
with open("Example1.txt","r") as File1:
```

```
for line in File1:  
    print(line)
```

```
This is line 1  
This is line 2  
This is line 3
```

```
This is line 1  
This is line 2  
This is line 3
```

We can use a loop to print out each line individually as follows.

```
with open("Example1.txt","r") as File1:
```

```
file_stuff=File1.readlines(4)  
print(file_stuff)
```

```
This is line 1  
This is line 2  
This is line 3
```

```
with open("Example1.txt","r") as File1:  
  
    file_stuff=File1.readlines(4)  
    print(file_stuff)
```

This

This is line 1

This is line 2

This is line 3

```
with open("Example1.txt","r") as File1:
```

```
    file_stuff=File1.readlines(16)  
    print(file_stuff)  
    file_stuff=File1.readlines(5)  
    print(file_stuff)  
    file_stuff=File1.readlines(9)  
    print(file_stuff)
```

This is line 1

This

is line 2

This is line 1

This is line 2

This is line 3

Reading Files Python

Estimated time needed: 40 minutes

Objectives

After completing this lab you will be able to:

Read text files using Python libraries

Table of Contents

- Download Data
- Reading Text Files
- A Better Way to Open a File

Download Data

```
In [1]: import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDriverSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%204/data/example
filename = 'Example1.txt'
urllib.request.urlretrieve(url, filename)

Out[1]: ('Example1.txt', <http.client.HTTPMessage at 0x2055450dc40>)

In [2]: # Download Example file

!wget -O /resources/data/Example1.txt https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDriverSkillsNetwork-PY0101EN-SkillsNetwo
'wget'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는
배치 파일이 아닙니다.
```

Reading Text Files

One way to read or write a file in Python is to use the built-in open function. The open function provides a **File object** that contains the methods and attributes you need in order to read, save, and manipulate the file. In this notebook, we will only cover .txt files. The first parameter you need is the file path and the file name. An example is shown as follow:



The mode argument is optional and the default value is **r**. In this notebook we only cover two modes:

- **r** Read mode for reading files
- **w** Write mode for writing files

For the next example, we will use the text file **Example1.txt**. The file is shown as follow:

We read the file:

```
In [3]: # Read the Example1.txt
example1 = "Example1.txt"
file1 = open(example1, "r")
```

We can view the attributes of the file.

The name of the file:

```
In [4]: # Print the path of file
file1.name
```

Out[4]: 'Example1.txt'

The mode the file object is in:

```
In [5]: # Print the mode of file, either 'r' or 'w'
file1.mode
```

Out[5]: 'r'

We can read the file and assign it to a variable :

```
In [6]: # Read the file
FileContent = file1.read()
FileContent
```

Out[6]: 'This is line 1 \nThis is line 2\nThis is line 3'

The **\n** means that there is a new line.

We can print the file:

```
In [7]: # Print the file with '\n' as a new line  
print(FileContent)
```

```
This is line 1  
This is line 2  
This is line 3
```

The file is of type string:

```
In [8]: # Type of file content  
type(FileContent)
```

```
Out[8]: str
```

It is very important that the file is closed in the end. This frees up resources and ensures consistency across different python versions.

```
In [9]: # Close file after finish  
file1.close()
```

A Better Way to Open a File

Using the `with` statement is better practice, it automatically closes the file even if the code encounters an exception. The code will run everything in the indent block then close the file object.

```
In [13]: # Open file using with  
  
with open(example1, "r") as file1:  
    FileContent = file1.read()  
  
print(FileContent)
```

```
This is line 1  
This is line 2  
This is line 3
```

The file object is closed, you can verify it by running the following cell:

```
In [14]: # Verify if the file is closed  
  
file1.closed
```

```
Out[14]: True
```

We can see the info in the file:

```
In [15]: # See the content of file  
  
print(FileContent)
```

```
This is line 1  
This is line 2  
This is line 3
```

The syntax is a little confusing as the file object is after the as statement. We also don't explicitly close the file. Therefore we summarize the steps in a figure:



We don't have to read the entire file, for example, we can read the first 4 characters by entering three as a parameter to the method `.read()`:

```
In [16]: # Read first four characters

with open(example1, "r") as file1:
    print(file1.read(4))
```

This

Once the method .read(4) is called the first 4 characters are called. If we call the method again, the next 4 characters are called. The output for the following cell will demonstrate the process for different inputs to the method read():

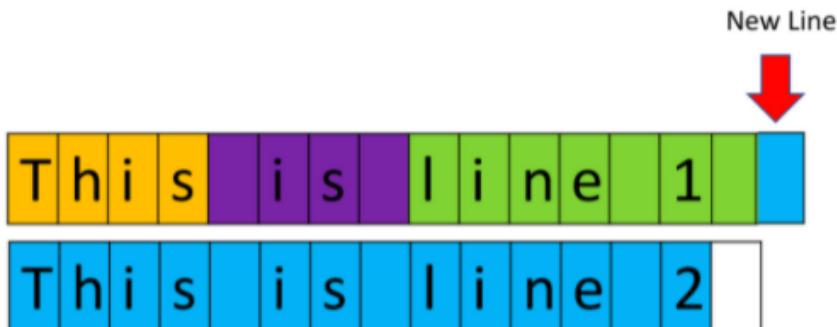
```
In [17]: # Read certain amount of characters

with open(example1, "r") as file1:
    print(file1.read(4))
    print(file1.read(4))
    print(file1.read(7))
    print(file1.read(15))

This
is
line 1

This is line 2
```

The process is illustrated in the below figure, and each color represents the part of the file read after the method read() is called:



- 1)file1.read(4)
- 2) file1.read(4)
- 3)file1.read(7)
- 4)file1.read(15)

Here is an example using the same file, but instead we read 16, 5, and then 9 characters at a time:

Here is an example using the same file, but instead we read 16, 5, and then 9 characters at a time:

```
In [18]: # Read certain amount of characters

with open(example1, "r") as file1:
    print(file1.read(16))
    print(file1.read(5))
    print(file1.read(9))
```

This is line 1

This
is line 2

We can also read one line of the file at a time using the method `readline()`:

```
In [21]: # Read one line

with open(example1, "r") as file1:
    print("first line: " + file1.readline())
    print("Second line: " + file1.readline())
    print("Third line: " + file1.readline())

first line: This is line 1
Second line: This is line 2
Third line: This is line 3
```

We can also pass an argument to `readline()` to specify the number of characters we want to read. However, unlike `read()`, `readline()` can only read one line at most.

```
In [22]: with open(example1, "r") as file1:
    print(file1.readline(20)) # does not read past the end of line
    print(file1.read(20)) # Returns the next 20 chars
```

This is line 1

This is line 2
This

We can use a loop to iterate through each line:

```
In [23]: # Iterate through the lines

with open(example1, "r") as file1:
    i = 0;
    for line in file1:
        print("Iteration", str(i), ": ", line)
        i = i + 1
```

```
In [23]: # Iterate through the lines

with open(example1,"r") as file1:
    i = 0;
    for line in file1:
        print("Iteration", str(i), ":", line)
        i = i + 1
```

```
Iteration 0 : This is line 1
Iteration 1 : This is line 2
Iteration 2 : This is line 3
```

```
In [24]: # Read all lines and save as a list

with open(example1, "r") as file1:
    FileasList = file1.readlines()
```

Each element of the list corresponds to a line of text:

```
In [25]: # Print the first line

FileasList[0]
```

```
Out[25]: 'This is line 1 \n'
```

Print the second line

```
FileasList[1]
```

```
In [27]: FileasList[1]
```

```
Out[27]: 'This is line 2\n'
```

```
In [26]: # Print the third line

FileasList[2]
```

```
Out[26]: 'This is line 3'
```

Writing Files with Open

We can also write to files using the open function. We will use Python's open function to get a file object to create a text file. We can apply method write to write data to that file.

As a result, text will be written to the file. We can create the file Example2.txt as follows.

We use the open function. The first argument is the file path. This is made up of the file name. If you have that file in your directory, it will be overwritten, and the file directory.

We set the mode parameter to W for writing. Finally, we have the file object.

As before we use the with statement. The code will run everything in the indent block then close the file. We create the file object, File1. We use the open function.

This creates a file Example2.txt in your directory. We use the method write, to write data into the file. The argument is the text we would like input into the file.

If we use the write method successively, each time it is called, it will write to the file.

The first time it is called, we will write, "This is line A " to represent a new line.

The second time we call the method, it will write, "this is line B " then it will close the file.

We can write each element in a list to a file. As before, we use a with command and the open function to create a file. The list, Lines, has three elements consisting of text.

We use a for loop to read each element of the first lines and pass it to the variable line.

The first iteration of the loop writes the first element of the list to the file Example2.

The second iteration writes the second element of the list and so on. At the end of the loop, the file will be closed. We can set the mode to appended using a lowercase a.

This will not create a new file but just use the existing file. If we call the method write, it will just write to the existing file, then add "This is line C" then close the file.

We can copy one file to a new file as follows. First, we read the file Example1 and interact with it via the file object, read file. Then we create a new file Example3 and

use the file object write file to interact with it. The for loop takes a line from the file object, read file, and stores it in the file Example3 using the file object, write file. The first iteration copies the first line.

The second iteration copies the second line, till the end of the file is reached.

Then both files are closed.

Python

Writing Files with Open

File Object



```
with open("/resources/data/Example2.txt", "w") as File1: with open("/resources/data/Example2.txt", "w") as File1:
```

```
File1.write ("This is line A\n")  
File1.write ("This is line B\n")
```

This is line A



Example2.txt

```
File1.write ("This is line A\n")  
File1.write ("This is line B\n")
```

This is line A
This is line B

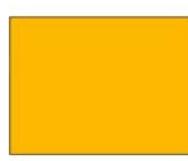
Example2.txt

```
Lines=["This is line A\n","This is line B\n","This is line C\n"]
```

```
with open("/resources/data/Example2.txt", "w") as File1:
```

```
for line in Lines:
```

```
    File1.write(line)
```



Example2.txt

```
Lines=["This is line A\n","This is line B\n","This is line C\n"]
```

```
with open("/resources/data/Example2.txt", "w") as File1:
```

```
for line in Lines:
```



```
    File1.write(line)
```

This is line A
This is line B
This is line C

Example2.txt

```
with open("/resources/data/Example2.txt", "a") as File1:
```

```
    File1.write ("This is line C" )
```

```
with open("Example1.txt", "r") as readfile:
```

```
    with open("Example3.txt", "w") as writefile:
```

```
        for line in readfile:
```

```
            writefile.write(line)
```

This is line A
This is line B
This is line C

Example2.txt

This is line A
This is line B
This is line C

Example1.txt

```
with open("Example1.txt", "r") as readfile:
```

```
    with open("Example3.txt", "w") as writefile:
```

```
        for line in readfile:
```

```
            writefile.write(line)
```

This is line A
This is line B
This is line C

Example1.txt

This is line A
This is line B
This is line C

Example3.txt



Write and Save Files in Python

Objectives

After completing this lab you will be able to:

Write to files using Python libraries

Table of Contents

- Writing Files
- Appending Files
- Additional File modes
- Copy a File

Writing Files

We can open a file object using the method write() to save the text file to a list. To write the mode, argument must be set to write w. Let's write a file Example2.txt with the line: "This is line A"

```
In [9]: # Write line to file
exmp2 = '/resources/data/Example2.txt'
with open(exmp2, 'w') as writefile:
    writefile.write("This is line A")

-----
OSError                                     Traceback (most recent call last)
<ipython-input-9-adc52450dbd9> in <module>
      1 # Write line to file
      2 exmp2 = 'http://localhost:8888/edit/IBM%20Data%20Science%20Professional%20Certificate/4.%20Python%20for%20Data%20Science%2C%20AI%20%26%20Development/Week%204/Example2.txt'
----> 3 with open(exmp2, 'w') as writefile:
      4     writefile.write("This is line A")

OSError: [Errno 22] Invalid argument: 'http://localhost:8888/edit/IBM%20Data%20Science%20Professional%20Certificate/4.%20Python%20for%20Data%20Science%2C%20AI%20%26%20Development/Week%204/Example2.txt'
```

We can read the file to see if it worked:

```
In [2]: # Read file

with open(exmp2, 'r') as testwritefile:
    print(testwritefile.read())

-----
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-2-3ac77dad032> in <module>
      1 # Read file
      2
----> 3 with open(exmp2, 'r') as testwritefile:
      4     print(testwritefile.read())

FileNotFoundError: [Errno 2] No such file or directory: '/resources/data/Example2.txt'
```

We can write multiple lines:

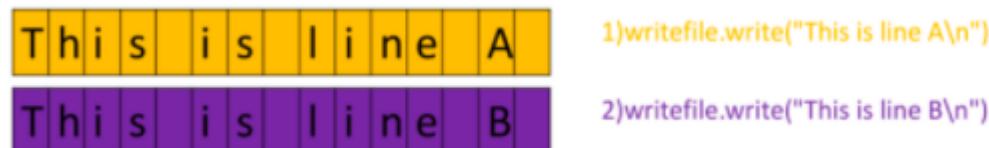
```
In [3]: # Write lines to file

with open(exmp2, 'w') as writefile:
    writefile.write("This is line A\n")
    writefile.write("This is line B\n")

-----
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-3-4081497a182f> in <module>
      1 # Write lines to file
      2
----> 3 with open(exmp2, 'w') as writefile:
      4     writefile.write("This is line A\n")
      5     writefile.write("This is line B\n")

FileNotFoundError: [Errno 2] No such file or directory: '/resources/data/Example2.txt'
```

The method `.write()` works similar to the method `.readline()`, except instead of reading a new line it writes a new line. The process is illustrated in the figure , the different colour coding of the grid represents a new line added to the file after each method call.



You can check the file to see if your results are correct

```
In [11]: # Check whether write to file

with open(exmp2, 'r') as testwritefile:
    print(testwritefile.read())

-----
OSErron-input-11-1915d8d9632a> Traceback (most recent call last)
<ipython-input-11-1915d8d9632a> in <module>
      1 # Check whether write to file
      2
----> 3 with open(exmp2, 'r') as testwritefile:
      4     print(testwritefile.read())

OSErron: [Errno 22] Invalid argument: 'http://localhost:8888/edit/IBM%20Data%20Science%20Professional%20Certificate/4.%20Python%20for%20Data%20Science%2C%20AI%20&%26%20Development/Week%204/Example2.txt'
```

We write a list to a .txt file as follows:

```
In [10]: # Sample List of text

Lines = ["This is line A\n", "This is line B\n", "This is line C\n"]
Lines

Out[10]: ['This is line A\n', 'This is line B\n', 'This is line C\n']
```

```
In [ ]: # Write the strings in the list to text file

with open('Example2.txt', 'w') as writefile:
    for line in Lines:
        print(line)
        writefile.write(line)
```

We can verify the file is written by reading it and printing out the values:

```
In [ ]: # Verify if writing to file is successfully executed

with open('Example2.txt', 'r') as testwritefile:
    print(testwritefile.read())
```

However, note that setting the mode to w overwrites all the existing data in the file.

```
In [ ]: with open('Example2.txt', 'w') as writefile:
    writefile.write("Overwrite\n")
with open('Example2.txt', 'r') as testwritefile:
    print(testwritefile.read())
```

Appending Files

We can write to files without losing any of the existing data as follows by setting the mode argument to append **a**. you can append a new line as follows:

```
In [ ]: # Write a new line to text file

with open('Example2.txt', 'a') as testwritefile:
    testwritefile.write("This is line C\n")
    testwritefile.write("This is line D\n")
    testwritefile.write("This is line E\n")
```

You can verify the file has changed by running the following cell:

```
In [ ]: # Verify if the new line is in the text file

with open('Example2.txt', 'r') as testwritefile:
    print(testwritefile.read())
```

Additional modes

It's fairly inefficient to open the file in **a** or **w** and then reopening it in **r** to read any lines. Luckily we can access the file in the following modes:

r+ : Reading and writing. Cannot truncate the file.

w+ : Writing and reading. Truncates the file.

a+ : Appending and Reading. Creates a new file, if none exists. You dont have to dwell on the specifics of each mode for this lab.

Let's try out the **a+** mode:

```
In [15]: with open('Example2.txt', 'a+') as testwritefile:
    testwritefile.write("This is line E\n")
    print(testwritefile.read())
```

There were no errors but **read()** also did not output anything. This is because of our location in the file.

Most of the file methods we've looked at work in a certain location in the file. **.write()** writes at a certain location in the file. **.read()** reads at a certain location in the file and so on. You can think of this as moving your pointer around in the notepad to make changes at specific location.

Opening the file in **w** is akin to opening the .txt file, moving your cursor to the beginning of the text file, writing new text and deleting everything that follows. Whereas opening the file in **a** is similiar to opening the .txt file, moving your cursor to the very end and then adding the new pieces of text.

It is often very useful to know where the 'cursor' is in a file and be able to control it. The following methods allow us to do precisely this -

- .tell() - returns the current position in bytes
- .seek(offset,from) - changes the position by 'offset' bytes with respect to 'from'. From can take the value of 0,1,2 corresponding to beginning, relative to current position and end

Now lets revisit **a+**

```
In [14]: with open('Example2.txt', 'at') as testwritefile:  
    print("Initial Location: {}".format(testwritefile.tell()))  
  
    data = testwritefile.read()  
    if (not data): #empty strings return false in python  
        print('Read nothing')  
    else:  
        print(testwritefile.read())  
  
    testwritefile.seek(0,0) # move 0 bytes from beginning.  
  
    print("\nNew Location : {}".format(testwritefile.tell()))  
    data = testwritefile.read()  
    if (not data):  
        print('Read nothing')  
    else:  
        print(data)  
  
    print("Location after read: {}".format(testwritefile.tell())) )  
  
Initial Location: 34  
Read nothing  
  
New Location : 0  
Line 1  
Line 2  
Line 3  
finished  
  
Location after read: 34
```

Finally, a note on the difference between **w+** and **r+**. Both of these modes allow access to read and write methods, However opening a file in **w+** overwrites it and deletes all existing data.

To work with a file on existing data, use **r+** and **a+**. While using **r+**, it can be useful to add a **.truncate()** method at the end of your data. This will reduce the file to your data and delete everything that follows.

In the following code block, Run the code as it is first and then run it with the **.truncate()**.

```
In [13]: with open('Example2.txt', 'r+') as testwritefile:  
    data = testwritefile.readlines()  
    testwritefile.seek(0,0) #write at beginning of file  
  
    testwritefile.write("Line 1" + "\n")  
    testwritefile.write("Line 2" + "\n")  
    testwritefile.write("Line 3" + "\n")  
    testwritefile.write("finished\n")  
    #Uncomment the line below  
    #testwritefile.truncate()  
    testwritefile.seek(0,0)  
    print(testwritefile.read())
```

```
Line 1  
Line 2  
Line 3  
finished
```

Copy a File

Let's copy the file **Example2.txt** to the file **Example3.txt**:

Copy a File

Let's copy the file **Example2.txt** to the file **Example3.txt**:

```
In [16]: # Copy file to another  
  
with open('Example2.txt','r') as readfile:  
    with open('Example3.txt','w') as writefile:  
        for line in readfile:  
            writefile.write(line)
```

We can read the file to see if everything works:

```
In [17]: # Verify if the copy is successfully executed  
  
with open('Example3.txt','r') as testwritefile:  
    print(testwritefile.read())
```



```
Line 1  
Line 2  
Line 3  
finished  
This is line E
```

After reading files, we can also write data into files and save them in different file formats like **.txt**, **.csv**, **.xls (for excel files) etc.** You will come across these in further examples

Now go to the directory to ensure the **.txt** file exists and contains the summary data that we wrote.

Exercise

Your local university's Raptors fan club maintains a register of its active members on a **.txt** document. Every month they update the file by removing the members who are not active. You have been tasked with automating this with your python skills.

Given the file **currentMem**, Remove each member with a 'no' in their inactive column. Keep track of each of the removed members and append them to the **exMem** file. Make sure the format of the original files is preserved. (*Hint: Do this by reading/writing whole lines and ensuring the header remains*)

Run the code block below prior to starting the exercise. The skeleton code has been provided for you, Edit only the **cleanFiles** function.

```
In [6]: #Run this prior to starting the exercise
from random import randint as rnd

memReg = 'members.txt'
exReg = 'inactive.txt'
fee = ('yes','no')

def genFiles(current,old):
    with open(current,'w+') as writeFile:
        writeFile.write('Membership No Date Joined Active \n')
        data = "{:|^13} {:<11} {:<6}\n"
        
        for rowno in range(20):
            date = str(rnd(2015,2020))+'-' + str(rnd(1,12))+'-'+str(rnd(1,25))
            writeFile.write(data.format(rnd(10000,99999),date,fee[rnd(0,1)]))

    with open(old,'w+') as writeFile:
        writeFile.write('Membership No Date Joined Active \n')
        data = "{:|^13} {:<11} {:<6}\n"
        for rowno in range(3):
            date = str(rnd(2015,2020))+'-' + str(rnd(1,12))+'-'+str(rnd(1,25))
            writeFile.write(data.format(rnd(10000,99999),date,fee[1]))


genFiles(memReg,exReg)
```

Start your solution below:

```
In [8]: def cleanFiles(currentMem,exMem):

    with open(currentMem,'r+') as writeFile:
        with open(exMem, 'a+') as appendFile:

            #get the data
            writeFile.seek(0)
            members = writeFile.readlines()

            #remove header
            header = members[0]
            members.pop(0)

            inactive = []

            for member in members:
                if 'no' in member:
                    inactive.append(member)

            #go to the beginning of the write file
            writeFile.seek(0)
            writeFile.write(header)

            for member in members:
                if (member in inactive):
                    appendFile.write(member)
                else:
                    writeFile.write(member)

            writeFile.truncate()

# Code to help you see the files
# Leave as is
memReg = 'members.txt'
exReg = 'inactive.txt'
cleanFiles(memReg,exReg)

headers = "Membership No Date Joined Active \n"
with open(memReg,'r') as readFile:
    print("Active Members: \n\n")
    print(readFile.read())

with open(exReg,'r') as readFile:
    print("Inactive Members: \n\n")
    print(readFile.read())
```

Active Members:

| Membership No | Date Joined | Active |
|---------------|-------------|--------|
| 43840 | 2016-12-17 | yes |
| 69345 | 2020-10-12 | yes |
| 53080 | 2018-3-24 | yes |
| 90493 | 2016-6-22 | yes |
| 18436 | 2017-4-17 | yes |
| 86574 | 2017-1-19 | yes |
| 57030 | 2019-10-16 | yes |
| 64078 | 2019-8-8 | yes |
| 91529 | 2017-3-21 | yes |

Inactive Members:

| Membership No | Date Joined | Active |
|---------------|-------------|--------|
| 61222 | 2020-10-24 | no |
| 56510 | 2017-10-10 | no |
| 12171 | 2015-5-23 | no |
| 66903 | 2016-11-24 | no |
| 92490 | 2018-11-5 | no |
| 92029 | 2015-11-23 | no |
| 30320 | 2019-7-14 | no |
| 71354 | 2015-6-9 | no |
| 24592 | 2018-6-3 | no |
| 49597 | 2017-10-2 | no |
| 33579 | 2015-4-12 | no |
| 55403 | 2015-5-5 | no |
| 52200 | 2019-6-9 | no |
| 55205 | 2017-5-7 | no |

Run the following to verify your code:

```
In [9]:
def testMsg(passed):
    if passed:
        return 'Test Passed'
    else :
        return 'Test Failed'

testWrite = "testWrite.txt"
testAppend = "testAppend.txt"
passed = True

genFiles(testWrite,testAppend)

with open(testWrite,'r') as file:
    ogWrite = file.readlines()

with open(testAppend,'r') as file:
    ogAppend = file.readlines()

try:
    cleanFiles(testWrite,testAppend)
except:
    print('Error')

with open(testWrite,'r') as file:
    clWrite = file.readlines()

with open(testAppend,'r') as file:
    clAppend = file.readlines()

# checking if total no of rows is same, including headers

if (len(ogWrite) + len(ogAppend) != len(clWrite) + len(clAppend)):
    print("The number of rows do not add up. Make sure your final files have the same header and format.")
    passed = False

for line in clWrite:
    if 'no' in line:
        passed = False
        print("Inactive members in file")
        break
    else:
        if line not in ogWrite:
            print("Data in file does not match original file")
            passed = False
print ("{}".format(testMsg(passed)))
```

Test Passed

Click here for the solution ````python def cleanFiles(currentMem,exMem): with open(currentMem,'r+') as writeFile: with open(exMem,'a+') as appendFile: #get the data writeFile.seek(0) members = writeFile.readlines() #remove header header = members[0] members.pop(0) inactive = [member for member in members if ('no' in member)] """ The above is the same as for member in active: if 'no' in member: inactive.append(member) """ #go to the beginning of the write file writeFile.seek(0) writeFile.write(header) for member in members: if (member in inactive): appendFile.write(member) else: writeFile.write(member) writeFile.truncate() memReg = 'members.txt' exReg = 'inactive.txt' cleanFiles(memReg,exReg) # code to help you see the files headers = "Membership No Date Joined Active \n" with open(memReg,'r') as readFile: print("Active Members: \n\n") print(readFile.read()) with open(exReg,'r') as readFile: print("Inactive Members: \n\n") print(readFile.read()) ````

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow this article to learn how to share your work.

Practice Quiz: Reading & Writing Files with Open

Bookmark

Question 1

1/1 point (ungraded)

What are the most common modes used when opening a file?

(a)ppend, (c)lose, (w)rite

(s)ave, (r)ead, (w)rite

(a)ppend, (r)ecline, (w)rite

(a)ppend, (r)ead, (w)rite



Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 2

1/1 point (ungraded)

What is the data attribute that will return the title of the file?

File1.open()

File1.close()

File1.mode

File1.name



Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 3

1/1 point (ungraded)

What is the command that tells Python to begin a new line?

 \q \b \e \n**Submit**

You have used 1 of 2 attempts



Question 4

1/1 point (ungraded)

What attribute is used to input data into a file?

 File1.close() File1.read() File1.open() File1.write()**Submit**

You have used 1 of 2 attempts



Loading Data with Pandas

Dependencies or libraries are pre-written code to help solve problems. In this video, we will introduce Pandas, a popular library for data analysis.

We can import the library or a dependency like Pandas using the following command.

We start with the import command followed by the name of the library.

We now have access to a large number of pre-built classes and functions.

This assumes the library is installed. In our lab environment,

all the necessary libraries are installed. Let's say we would like to load a CSV file using the Pandas built-in function, read csv. A CSV is a typical file type used to store data.

We simply typed the word Pandas, then a dot, and the name of the function with all the inputs. Typing Pandas all the time may get tedious. We can use the as statement to shorten the name of the library.

In this case, we use the standard abbreviation, pd. Now we type pd, and a dot, followed by the name of the function we would like to use. In this case, read_csv.

We are not limited to the abbreviation pd. In this case, we use the term banana.

We will stick with pd for the rest of this video. Let's examine this code more in-depth.

One way Pandas allows you to work with data is with the data frame. Let's go over the process to go from a CSV file to a data frame. This variable stores the path of the CSV.

It is used as an argument to the read_csv function. The result is stored to the variable df.

This is short for data frame. Now that we have the data in a data frame, we can work with it.

We can use the method head to examine the first five rows of a data frame. The process for loading an Excel file is similar. We use the path of the Excel file.

The function reads Excel. The result is a data frame. A data frame is comprised of rows and columns. We can create a data frame out of a dictionary. The keys correspond to the column labels.

The values or lists corresponding to the rows. We then cast the dictionary to a data frame using the function data frame. We can see the direct correspondence between the table.

The keys correspond to the table headers. The values are lists corresponding to the rows.

We can create a new data frame consisting of one column. We just put the data frame name, in this case, df, and the name of the column header enclosed in double brackets.

The result is a new data frame comprised of the original column. You can do the same thing for multiple columns. We just put the data frame name, in this case, df,

and the name of the multiple column headers enclosed in double brackets.

The result is a new data frame comprised of the specified columns.

Python

Loading Data with Pandas

Importing Pandas

```
import pandas  
csv_path='file1.csv'  
df=pandas.read_csv(csv_path)
```

pandas

| |
|------------|
| read_csv() |
| Series() |
| DataFrame |
| values |
| : |
| : |
| : |
| |

Importing Pandas

```
import pandas  
csv_path='file1.csv'  
df=pandas.read_csv(csv_path)
```

pandas

| |
|------------|
| read_csv() |
| Series() |
| DataFrame |
| values |
| : |
| : |
| : |
| |

Importing

```
import pandas as pd  
csv_path='file1.csv'  
df= pd.read_csv(csv_path)
```

Importing

```
import pandas as pd  
csv_path='file1.csv'  
df= pd.read_csv(csv_path)
```

Importing

```
import pandas as banana  
csv_path='file1.csv'  
df=banana.read_csv(csv_path)
```

Dataframes

```
csv_path='file1.csv'
```

```
df= pd.read_csv(csv_path)  
df.head()
```



Dataframes

```
csv_path='file1.csv'
```

```
df= pd.read_csv(csv_path)  
df.head()
```



Dataframes

```
csv_path='file1.csv'
```

```
df= pd.read_csv(csv_path)
```

```
df.head()
```

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |

for Excel file

Dataframes

```
xlsx_path='file1.xlsx'
```

```
df= pd.read_excel(xlsx_path)  
df.head()
```



Dataframes

```
xlsx_path='file1.xlsx'
```

```
df= pd.read_excel(xlsx_path)  
df.head()
```



Dataframes

The diagram illustrates the structure of a DataFrame. It features a grid of 8 rows and 12 columns. A vertical bracket on the left side is labeled 'ROWS' and spans all 8 rows. A horizontal bracket at the top is labeled 'columns' and spans all 12 columns. The columns are labeled: Artist, Album, Released, Length, Genre, Music Recording Sales (millions), Claimed Sales (millions), Released.1, Soundtrack, and Rating.

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 | Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 | Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 | Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

Create Dataframe out of a dictionary

Dataframes

```
songs = {'Album': ['Thriller', 'Back in Black', 'The Dark Side of the Moon',\ 
'The Bodyguard', 'Bat Out of Hell'],\ 
'Released': [1982, 1980, 1973, 1992, 1977],\ 
'Length':['00:42:19', '00:42:11', '00:42:49', '00:57:44', '00:46:33']}
```

```
songs_frame = pd.DataFrame(songs)
```

Dataframes

```
songs = {'Album' : ['Thriller', 'Back in Black', 'The Dark Side of the Moon',\ 
'The Bodyguard', 'Bat Out of Hell'],\ 
'Released' : [1982, 1980, 1973, 1992, 1977],\ 
'Length': ['00:42:19', '00:42:11', '00:42:49', '00:57:44', '00:46:33']}
```

| | Album | Length | Released |
|---|---------------------------|----------|----------|
| 0 | Thriller | 00:42:19 | 1982 |
| 1 | Back in Black | 00:42:11 | 1980 |
| 2 | The Dark Side of the Moon | 00:42:49 | 1973 |
| 3 | The Bodyguard | 00:57:44 | 1992 |
| 4 | Bat Out of Hell | 00:46:33 | 1977 |

`x=df[['Length']]`

| Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|-------------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

X

| | Length |
|---|---------|
| 0 | 0:42:19 |
| 1 | 0:42:11 |
| 2 | 0:42:49 |
| 3 | 0:57:44 |
| 4 | 0:46:33 |
| 5 | 0:43:08 |
| 6 | 1:15:54 |
| 7 | 0:40:01 |



For multiple columns

`y=df[['Artist', 'Length', 'Genre']]`

| Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |



| Artist | Length | Genre |
|-------------------|---------|-----------------------------|
| 0 Michael Jackson | 0:42:19 | pop, rock, R&B |
| 1 AC/DC | 0:42:11 | hard rock |
| 2 Pink Floyd | 0:42:49 | progressive rock |
| 3 Whitney Houston | 0:57:44 | R&B, soul, pop |
| 4 Meat Loaf | 0:46:33 | hard rock, progressive rock |
| 5 Eagles | 0:43:08 | rock, soft rock, folk rock |
| 6 Bee Gees | 1:15:54 | disco |
| 7 Fleetwood Mac | 0:40:01 | soft rock |

Pandas: Working with and Saving Data

When we have a data frame we can work with the data and save the results in other formats. Consider the stack of 13 blocks of different colors. We can see there are three unique colors.

Let's say you would like to find out how many unique elements are in a column of a data frame. This may be much more difficult because instead of 13 elements, you may have millions. Pandas has the method `unique` to determine the unique elements in a column of a data frame.

Lets say we would like to determine the unique year of the albums in the data set.

We enter the name of the data frame, then enter the name of the column released within brackets. Then we apply the method `unique`.

The result is all of the unique elements in the column released. Let's say we would like to create a new database consisting of songs from the 1980s and after.

We can look at the column released for songs made after 1979, then select the corresponding columns.

We can accomplish this within one line of code in Pandas. But let's break up the steps.

We can use the inequality operators for the entire data frame in Pandas. The result is a series of Boolean values.

For our case, we simply specify the column released and the inequality for the albums after 1979.

The result is a series of Boolean values. The result is true when the condition is true and false otherwise.

We can select the specified columns in one line. We simply use the data frames names and square brackets we placed the previously mentioned inequality and assign it to the variable `df1`.

We now have a new data frame, where each album was released after 1979.

We can save the new data frame using the method `to_csv`. The argument is the name of the csv file.

Make sure you include `a.csv` extension. There are other functions to save the data frame in other formats.

Pandas

Working with and Saving Data

List Unique Values



`df['Released'].unique()`

| | Released |
|---|----------|
| 0 | 1982 |
| 1 | 1980 |
| 2 | 1973 |
| 3 | 1992 |
| 4 | 1977 |
| 5 | 1976 |
| 6 | 1977 |
| 7 | 1977 |

| |
|------|
| 1982 |
| 1980 |
| 1973 |
| 1992 |
| 1977 |
| 1976 |

Dataframe of Album released after 1979

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 | Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 | Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 | Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

Breaking up the steps of finding albums after 1979:

`df['Released']>=1980`

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 | Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 | Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 | Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

df['Released']>=1980

| Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|-------------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |



| | |
|---|-------|
| 0 | True |
| 2 | True |
| 3 | False |
| 4 | True |
| 5 | False |
| 6 | False |
| 7 | False |
| 8 | False |

df1=df[df['Released']>=1980]

| Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|-------------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

We now have a new dataframe:

```
df1=df[df['Released']>=1980]
```

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------|----------|----------|----------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 00:42:19 | pop, rock, R&B | 46.0 | 65 | 1982-11-30 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 00:42:11 | hard rock | 26.1 | 50 | 1980-07-25 | NaN | 9.5 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 00:57:44 | R&B, soul, pop | 27.4 | 44 | 1992-11-17 | Y | 8.5 |

df1

To Save the new dataframe:

Save as CSV

```
df1.to_csv('new_songs.csv')
```



IBM Developer
SKILLS NETWORK

Introduction to Pandas in Python

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Use Pandas to access and view data

Table of Contents

- About the Dataset
- Introduction of [Pandas](#)
- Viewing Data and Accessing Data
- Quiz on DataFrame

About the Dataset

The table has one row for each album and several columns

- **artist**: Name of the artist
- **album**: Name of the album
- **released_year**: Year the album was released
- **length_min_sec**: Length of the album (hours,minutes,seconds)
- **genre**: Genre of the album
- **music_recording_sales_millions**: Music recording sales (millions in USD)
on [\[SONG://DATABASE\]](#)
- **claimed_sales_millions**: Album's claimed sales (millions in USD)
on [\[SONG://DATABASE\]](#)
- **date_released**: Date on which the album was released
- **soundtrack**: Indicates if the album is the movie soundtrack (Y) or (N)
- **rating_of_friends**: Indicates the rating from your friends from 1 to 10

You can see the dataset here:

```
<table font-size:xx-small>
```

| Artist | Album | Released | Length | Genre | Music recording sales (millions) | Claimed sales (millions) | Released |
|-----------------|----------|----------|----------|----------------|----------------------------------|--------------------------|-----------|
| Michael Jackson | Thriller | 1982 | 00:42:19 | Pop, rock, R&B | 46 | 65 | 30-Nov-82 |

```
10.0 AC/DC Back in Black 1980 00:42:11 Hard rock 26.1 50 25-Jul-80 8.5 Pink Floyd The Dark Side of  
the Moon 1973 00:42:49 Progressive rock 24.2 45 01-Mar-73 9.5 Whitney Houston The Bodyguard  
1992 00:57:44 Soundtrack/R&B, soul, pop 26.1 50 25-Jul-80 Y 7.0 Meat Loaf Bat Out of Hell 1977  
00:46:33 Hard rock, progressive rock 20.6 43 21-Oct-77 7.0 Eagles Their Greatest Hits (1971-1975)  
1976 00:43:08 Rock, soft rock, folk rock 32.2 42 17-Feb-76 9.5 Bee Gees Saturday Night Fever 1977  
1:15:54 Disco 20.6 40 15-Nov-77 Y 9.0 Fleetwood Mac Rumours 1977 00:40:01 Soft rock 27.9 40  
04-Feb-77 9.5 </table></font>
```

Introduction of Pandas

```
In [1]: # Dependency needed to install file  
!pip install xlrd
```

```
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/dhcrypto.py:16:  
CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes instead  
  
from cryptography.utils import int_from_bytes  
  
/opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages/secretstorage/util.py:25: CryptographyDeprecationWarning:  
int_from_bytes is deprecated, use int.from_bytes instead  
  
from cryptography.utils import int_from_bytes  
  
Requirement already satisfied: xlrd in /opt/conda/envs/Python-3.7-main/lib/python3.7/site-packages (1.2.0)
```

```
In [2]: # Import required library  
import pandas as pd
```

After the import command, we now have access to a large number of pre-built classes and functions. This assumes the library is installed; in our lab environment all the necessary libraries are installed. One way pandas allows you to work with data is a dataframe. Let's go through the process to go from a comma separated values (.csv) file to a dataframe. This variable csv_path stores the path of the .csv, that is used as an argument to the read_csv function. The result is stored in the object df, this is a common short form used for a variable referring to a Pandas dataframe.

```
In [4]: # Read data from CSV file  
  
csv_path = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%204/data/To  
df = pd.read_csv(csv_path)
```

We can use the method head() to examine the first five rows of a dataframe:

```
In [5]: # Print first five rows of the dataframe
df.head()
```

Out[5]:

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |

We use the path of the excel file and the function `read_excel`. The result is a data frame as before:

```
In [6]: # Read data from Excel File and print the first five rows
xlsx_path = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/PY0101EN/Chapter%204/Datasets/TopSellingAlbums.xlsx'
df = pd.read_excel(xlsx_path)
df.head()
```

Out[6]:

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------|----------|----------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 00:42:19 | pop, rock, R&B | 46.0 | 65 | 1982-11-30 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 00:42:11 | hard rock | 26.1 | 50 | 1980-07-25 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 00:42:49 | progressive rock | 24.2 | 45 | 1973-03-01 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 00:57:44 | R&B, soul, pop | 27.4 | 44 | 1992-11-17 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 00:46:33 | hard rock, progressive rock | 20.6 | 43 | 1977-10-21 | NaN | 8.0 |

We can access the column **Length** and assign it a new dataframe **x**:

```
In [7]: # Access to the column Length
x = df[['Length']]
x
```

```
Out[7]:    Length
0 00:42:19
1 00:42:11
2 00:42:49
3 00:57:44
4 00:46:33
5 00:43:08
6 01:15:54
7 00:40:01
```

The process is shown in the figure:

x=df[['Length']]

The diagram illustrates the extraction of the 'Length' column from a DataFrame. On the left, a large DataFrame table contains 8 rows of music album data. The 'Length' column is highlighted with a red rectangular box. An arrow points from this box to a smaller table on the right, which shows only the 'Length' column for each row, labeled 'x' at the top.

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | Pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | Hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | Progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | Hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 | Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | Rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 | Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | Disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 | Fleetwood Mac | Rumours | 1977 | 0:40:01 | Soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

X

| | Length |
|---|---------|
| 0 | 0:42:19 |
| 1 | 0:42:11 |
| 2 | 0:42:49 |
| 3 | 0:57:44 |
| 4 | 0:46:33 |
| 5 | 0:43:08 |
| 6 | 1:15:54 |
| 7 | 0:40:01 |

Viewing Data and Accessing Data

You can also get a column as a series. You can think of a Pandas series as a 1-D dataframe. Just use one bracket:

```
In [8]: # Get the column as a series
```

```
x = df['Length']  
x
```

```
Out[8]: 0    00:42:19  
1    00:42:11  
2    00:42:49  
3    00:57:44  
4    00:46:33  
5    00:43:08  
6    01:15:54  
7    00:40:01  
Name: Length, dtype: object
```

You can also get a column as a dataframe. For example, we can assign the column Artist:

```
In [15]: # Get the column as a dataframe
```

```
x = type(df[['Artist']])  
x  
#y = type(df['Artist'])  
#y  
#=> pandas.core.series.Series
```

```
Out[15]: pandas.core.frame.DataFrame
```

You can do the same thing for multiple columns; we just put the dataframe name, in this case, df, and the name of the multiple column headers enclosed in double brackets. The result is a new dataframe comprised of the specified columns:

```
In [16]: # Access to multiple columns
```

```
y = df[['Artist', 'Length', 'Genre']]  
y
```

| | Artist | Length | Genre |
|---|-----------------|----------|-----------------------------|
| 0 | Michael Jackson | 00:42:19 | pop, rock, R&B |
| 1 | AC/DC | 00:42:11 | hard rock |
| 2 | Pink Floyd | 00:42:49 | progressive rock |
| 3 | Whitney Houston | 00:57:44 | R&B, soul, pop |
| 4 | Meat Loaf | 00:46:33 | hard rock, progressive rock |
| 5 | Eagles | 00:43:08 | rock, soft rock, folk rock |
| 6 | Bee Gees | 01:15:54 | disco |
| 7 | Fleetwood Mac | 00:40:01 | soft rock |

```
y=df[ ['Artist' , 'Length', 'Genre' ] ]
```

y

| Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| Fleetwood Mac | Tusk | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 8.5 |



| | Artist | Length | Genre |
|---|-----------------|---------|-----------------------------|
| 0 | Michael Jackson | 0:42:19 | pop, rock, R&B |
| 1 | AC/DC | 0:42:11 | hard rock |
| 2 | Pink Floyd | 0:42:49 | progressive rock |
| 3 | Whitney Houston | 0:57:44 | R&B, soul, pop |
| 4 | Meat Loaf | 0:46:33 | hard rock, progressive rock |
| 5 | Eagles | 0:43:08 | rock, soft rock, folk rock |
| 6 | Bee Gees | 1:15:54 | disco |
| 7 | Fleetwood Mac | 0:40:01 | soft rock |

One way to access unique elements is the `iloc` method, where you can access the 1st row and the 1st column as follows:

```
In [17]: # Access the value on the first row and the first column  
df.iloc[0, 0]
```

```
Out[17]: 'Michael Jackson'
```

You can access the 2nd row and the 1st column as follows:

```
In [18]: # Access the value on the second row and the first column  
df.iloc[1,0]
```

```
Out[18]: 'AC/DC'
```

You can access the 1st row and the 3rd column as follows:

```
In [19]: # Access the value on the first row and the third column  
df.iloc[0,2]
```

```
Out[19]: 1982
```

```
In [20]: # Access the value on the second row and the third column  
df.iloc[1,2]
```

```
Out[20]: 1980
```

This is shown in the following image

df.iloc[0,0]: 'Michael Jackson'

df.iloc[0,2]: 1982

df.iloc[1,0]: 'AC/DC'

df.iloc[1,2]: 1980

| | Artist | Album | Released | Length | Genre | Music recording sales (millions) | Claimed sales (millions) | Released | Soundtrack | Rating (friends) |
|---|-----------------|---------------------------------|----------|----------|-----------------------------|----------------------------------|--------------------------|-----------|------------|------------------|
| 0 | Michael Jackson | Thriller | 1982 | 00:42:19 | Pop, rock, R&B | 46 | 65 | 30-Nov-82 | | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 00:42:11 | Hard rock | 26.1 | 50 | 25-Jul-80 | | 8.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 00:42:49 | Progressive rock | 24.2 | 45 | 01-Mar-73 | | 9.5 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 00:57:44 | Soundtrack/R&B, soul, pop | 26.1 | 50 | 25-Jul-80 | Y | 7.0 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 00:46:33 | Hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | | 7.0 |
| 5 | Eagles | Their Greatest Hits (1971-1975) | 1976 | 00:43:08 | Rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | | 9.5 |
| 6 | Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | Disco | 20.6 | 40 | 15-Nov-77 | Y | 9.0 |
| 7 | Fleetwood Mac | Rumours | 1977 | 00:40:01 | Soft rock | 27.9 | 40 | 04-Feb-77 | | 9.5 |

In []:

In [34]:

```
# Access the column using the name  
df.loc[1, 'Artist']
```

Out[34]:

```
'AC/DC'
```

In [35]:

```
# Access the column using the name  
df.loc[0, 'Released']
```

Out[35]:

```
1982
```

In [36]:

```
# Access the column using the name  
df.loc[1, 'Released']
```

Out[36]:

```
1980
```

df.loc[0, 'Artist']:'Michael Jackson'

df.loc[1, 'Artist']:'AC/DC'

df.loc[0, 'Released']:1982

df.loc[1, 'Released']:1980

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 | Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 | Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 | Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

You can perform slicing using both the index and the name of the column:

```
In [37]: # Slicing the dataframe
df.iloc[0:2, 0:3]
```

```
Out[37]:
```

| | Artist | Album | Released |
|---|-----------------|---------------|----------|
| 0 | Michael Jackson | Thriller | 1982 |
| 1 | AC/DC | Back in Black | 1980 |

z=df.iloc[0:2, 0:3]

| Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|-------------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |



Z

| Artist | Album | Released |
|-------------------|---------------|----------|
| 0 Michael Jackson | Thriller | 1982 |
| 1 AC/DC | Back in Black | 1980 |

```
In [38]: # Slicing the dataframe using name
df.loc[0:2, 'Artist':'Released']
```

Out[38]:

| | Artist | Album | Released |
|---|-----------------|---------------------------|----------|
| 0 | Michael Jackson | Thriller | 1982 |
| 1 | AC/DC | Back in Black | 1980 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 |

| Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.f | Soundtrack | Rating |
|-------------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 85 | 00-Nov-82 | NaN | 10.0 |
| 1 AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:40 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.9 |
| 4 Meat Loaf | Bat Out of Hell | 1977 | 0:48:30 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:00 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.8 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 8.5 |



Quiz on DataFrame

Use a variable q to store the column Rating as a dataframe

```
In [40]: # Write your code below and press Shift+Enter to execute
q = df[['Rating']]
q
```

Out[40]:

| | Rating |
|---|--------|
| 0 | 10.0 |
| 1 | 9.5 |
| 2 | 9.0 |
| 3 | 8.5 |
| 4 | 8.0 |
| 5 | 7.5 |
| 6 | 7.0 |
| 7 | 6.5 |

Click here for the solution `python q = df[['Rating']] q`
Assign the variable `q` to the dataframe that is made up of the column **Released** and **Artist**:

```
In [41]: # Write your code below and press Shift+Enter to execute
q = df[['Released', 'Artist']]
q
```

```
Out[41]:   Released      Artist
0    1982  Michael Jackson
1    1980        AC/DC
2    1973     Pink Floyd
3    1992  Whitney Houston
4    1977     Meat Loaf
5    1976        Eagles
6    1977     Bee Gees
7    1977  Fleetwood Mac
```

Click here for the solution `python q = df[['Released', 'Artist']] q`

Access the 2nd row and the 3rd column of `df`:

```
In [43]: # Write your code below and press Shift+Enter to execute
df.iloc[1,2]
```

```
Out[43]: 1980
```

Click here for the solution `python df.iloc[1, 2]`

Use the following list to convert the dataframe index `df` to characters and assign it to `df_new`; find the element corresponding to the row index `a` and column `'Artist'`. Then select the rows `a` through `d` for the column `'Artist'`

```
In [44]: new_index=['a','b','c','d','e','f','g','h']
df_new = df
df_new.index = new_index
df_new.loc['a', 'Artist']
df_new.loc['a':'d', 'Artist']
```

```
Out[44]: a    Michael Jackson
b        AC/DC
c     Pink Floyd
d  Whitney Houston
Name: Artist, dtype: object
```

Click here for the solution `python df_new=df df_new.index=new_index df_new.loc['a', 'Artist'] df_new.loc['a':'d', 'Artist']`

Practice Quiz: Pandas

Bookmarked

Question 1

1/1 point (ungraded)

What python object do you cast to a dataframe?

Dictionary

Set



Answer

Correct: Correct.

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 2

1/1 point (ungraded)

How would you access the first-row and first column in the dataframe df?

df.ix[1,0]

df.ix[0,1]

df.ix[0,0]



Answer

Correct: Correct

Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 3

1/1 point (ungraded)

What is the proper way to load a CSV file using pandas?

pandas.import_csv('data.csv')

pandas.load_csv('data.csv')

pandas.read_csv('data.csv')

pandas.from_csv('data.csv')



Answer

Correct: Correct

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 4

1/1 point (ungraded)

Use this dataframe to answer the question.

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.t | Soundtrack | Rating |
|---|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 | Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 | Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 | Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

How would you select the Genre disco?

df.loc['Bee Gees', 'Genre']

df.loc[6, 5]

df.iloc[6, 4]

df.iloc[6, 'genre']



Answer

Correct: Corecct.

Submit

You have used 2 of 2 attempts

Question 5

0/1 point (ungraded)

Use this dataframe to answer the question.

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:46:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 | Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 | Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 | Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

Which will NOT evaluate to 20.6, select all that apply?

df.iloc[4,5]

df.iloc[6,5]

df.loc[4,'Music Recording Sales']

df.iloc[6, 'Music Recording Sales (millions)']

Submit

You have used 2 of 2 attempts

✖ Incorrect (0/1 point)

Question 6

0/1 point (ungraded)

Use this dataframe to answer the question.

| | Artist | Album | Released | Length | Genre | Music Recording Sales (millions) | Claimed Sales (millions) | Released.1 | Soundtrack | Rating |
|---|-----------------|---------------------------------|----------|---------|-----------------------------|----------------------------------|--------------------------|------------|------------|--------|
| 0 | Michael Jackson | Thriller | 1982 | 0:42:19 | pop, rock, R&B | 46.0 | 65 | 30-Nov-82 | NaN | 10.0 |
| 1 | AC/DC | Back in Black | 1980 | 0:42:11 | hard rock | 26.1 | 50 | 25-Jul-80 | NaN | 9.5 |
| 2 | Pink Floyd | The Dark Side of the Moon | 1973 | 0:42:49 | progressive rock | 24.2 | 45 | 01-Mar-73 | NaN | 9.0 |
| 3 | Whitney Houston | The Bodyguard | 1992 | 0:57:44 | R&B, soul, pop | 27.4 | 44 | 17-Nov-92 | Y | 8.5 |
| 4 | Meat Loaf | Bat Out of Hell | 1977 | 0:48:33 | hard rock, progressive rock | 20.6 | 43 | 21-Oct-77 | NaN | 8.0 |
| 5 | Eagles | Their Greatest Hits (1971-1975) | 1976 | 0:43:08 | rock, soft rock, folk rock | 32.2 | 42 | 17-Feb-76 | NaN | 7.5 |
| 6 | Bee Gees | Saturday Night Fever | 1977 | 1:15:54 | disco | 20.6 | 40 | 15-Nov-77 | Y | 7.0 |
| 7 | Fleetwood Mac | Rumours | 1977 | 0:40:01 | soft rock | 27.9 | 40 | 04-Feb-77 | NaN | 6.5 |

How do we select Albums The Dark Side of the Moon to Their Greatest Hits (1971-1975)? Select:

df.iloc[2:5, 'Album']

df.loc[2:5, 'Album']

df.iloc[2:6, 1]

df.loc[2:5, 1]

✗

Submit

You have used 2 of 2 attempts

One Dimensional NumPy

In this video we will be covering numpy in 1D, in particular ND arrays. Numpy is a library for scientific computing. It has many useful functions.

There are many other advantages like speed and memory. Numpy is also the basis for pandas.

So check out our pandas video. In this video we will be covering the basics and array creation, indexing and slicing, basic operations, universal functions. Let's go over how to create a numpy array.

A Python list is a container that allows you to store and access data. Each element is associated with an index.

We can access each element using a square bracket as follows. A numpy array or ND array is similar to a list.

It's usually fixed in size and each element is of the same type, in this case integers. We can cast a list to a numpy array by first importing numpy. We then cast the list as follows; we can access the data via an index.

As with the list, we can access each element with an integer and a square bracket. The value of a is stored as follows.

If we check the type of the array we get, numpy.ndarray. As numpy arrays contain data of the same type, we can use the attribute dtype to obtain the data type of the array's elements. In this case a 64-bit integer.

Let's review some basic array attributes using the array a. The attribute size is the number of elements in the array.

As there are five elements the result is five. The next two attributes will make more sense when we get to higher dimensions, but let's review them.

The attribute ndim represents the number of array dimensions or the rank of the array, in this case one.

The attribute shape is a tuple of integers indicating the size of the array in each dimension. We can create a numpy array with real numbers.

When we check the type of the array, we get numpy.ndarray. If we examine the attribute D type, we see float64 as the elements are not integers.

There were many other attributes, check out numpy.org. Let's review some indexing and slicing methods.

We can change the first element of the array to 100 as follows. The array's first value is now 100.

We can change the fifth element of the array as follows. The fifth element is now zero. Like lists and tuples we can slice a NumPy array. The elements of the array correspond to the following index. We can select the elements from one to three and assign it to a new numpy array d as follows.

The elements in d correspond to the index. Like lists, we do not count the element corresponding to the last index. We can assign the corresponding indices to new values as follows. The array c now has new values.

See the labs or numpy.org for more examples of what you can do with numpy. Numpy makes it easier to do many operations that are commonly performed in data science. The same operations are usually computationally faster and require less memory in numpy compared to regular Python. Let's review some of these operations on one-dimensional arrays. We will look at many of the operations in the context of Euclidian vectors to make things more interesting. Vector addition is a widely used operation in data science.

Consider the vector u with two elements, the elements are distinguished by the different colors. Similarly, consider the vector v with two components. In vector addition, we create a new vector in this case z.

The first component of z is the addition of the first component of vectors u and v. Similarly, the second component is the sum of the second components of u and v. This new vector z is now a linear combination of the vector u and

v. Representing vector addition with line segment or arrows is helpful. The first vector is represented in red.

The vector will point in the direction of the two components. The first component of the vector is one.

As a result the arrow is offset one unit from the origin in the horizontal direction. The second component is zero, we represent this component in the vertical direction. As this component is zero, the vector does not point in the vertical direction.

We represent the second vector in blue. The first component is zero, therefore the arrow does not point to the horizontal direction. The second component is one. As a result the vector points in the vertical direction one unit.

When we add the vector u and v , we get the new vector z . We add the first component, this corresponds to the horizontal direction. We also add the second component. It's helpful to use the tip to tail method when adding vectors, placing the tail of the vector v on the tip of vector u . The new vector z is constructed by connecting

the base of the first vector u with the tail of the second v . The following three lines of code we'll add the two lists and place the result in the list z . We can also perform vector addition with one line of NumPy code.

It would require multiple lines to perform vector subtraction on two lists as shown on the right side of the screen.

In addition, the numpy code will run much faster. This is important if you have lots of data. We can also perform vector subtraction by changing the addition sign to a subtraction sign. It would require multiple lines

perform vector subtraction on two lists as shown on the right side of the screen. Vector multiplication with a scalar is another commonly performed operation. Consider the vector y , each component is specified by a different color.

We simply multiply the vector by a scalar value in this case two. Each component of the vector is multiplied by two, in this case each component is doubled. We can use the line segment or arrows to visualize what's going on.

The original vector y is in purple. After multiplying it by a scalar value of two, the vector is stretched out by two units as shown in red. The new vector is twice as long in each direction. Vector multiplication with a scalar only requires one line of code using numpy. It would require multiple lines to perform the same task as

shown with Python lists as shown on the right side of the screen. In addition, the operation would also be much slower.

Hadamard product is another widely used operation in data science. Consider the following two vectors, u and v . The Hadamard product of u and v is a new vector z . The first component of z is the product of the first element of u and v . Similarly, the second component is the product of the second element of u and v . The resultant vector consists of the entry wise product of u and v . We can also perform hadamard product with one line of code in numpy. It would require multiple lines to perform hadamard product on two lists as shown on the right side of the screen. The dot product is another widely used operation in data science.

Consider the vector u and v , the **dot product** is a single number given by the following term and

represents how similar two vectors are. We multiply the first component from v and u , we then multiply the second component and add the result together. The result is a number that represents how similar the two vectors are.

We can also perform dot product using the numpy function `dot` and assign it with the variable `result` as follows.

Consider the array u , the array contains the following elements. If we add a scalar value to the array,

numpy will add that value to each element. This property is known as **broadcasting**. A universal function is a function that operates on ND arrays. We can apply a universal function to a numpy array.

Consider the arrays a , we can calculate the mean or average value of all the elements in a using the method `mean`.

This corresponds to the average of all the elements. In this case the result is zero. There are many other functions. For example, consider the numpy arrays b. We can find the maximum value using the method max.

We see the largest value is five, therefore the method max returns a five. We can use numpy to create functions that map numpy arrays to new numpy arrays. Let's implement some code on the left side of the screen

and use the right side of the screen to demonstrate what's going on. We can access the value of pie in numpy as follows. We can create the following numpy array in radians. This array corresponds to the following vector.

We can apply the function sin to the array x and assign the values to the array y. This applies the sin function to each element in the array, this corresponds to applying the sine function to each component of the vector.

The result is a new array y, where each value corresponds to a sine function being applied to each element in the array x.

A useful function for plotting mathematical functions is line space. Line space returns evenly spaced numbers over specified interval. We specify the starting point of the sequence, the ending point of the sequence.

The parameter num indicates the number of samples to generate, in this case five. The space between samples is one.

If we change the parameter num to nine, we get nine evenly spaced numbers over the integral from negative two to two. The result is the difference between subsequent samples is 0.5 as opposed to one as before. We can use the function line space to generate 100 evenly spaced samples from the interval zero to two pie. We can use the numpy function sin to map the array x to a new array y. We can import the library pyplot as plt to help us plot the function.

As we are using a Jupiter notebook, we use the command matplotlib inline to display the plot. The following command plots a graph. The first input corresponds to the values for the horizontal or x-axis. The second input corresponds to the values for the vertical or y-axis. There's a lot more you can do with numpy.

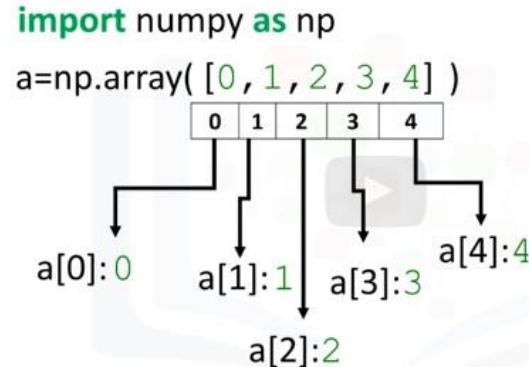
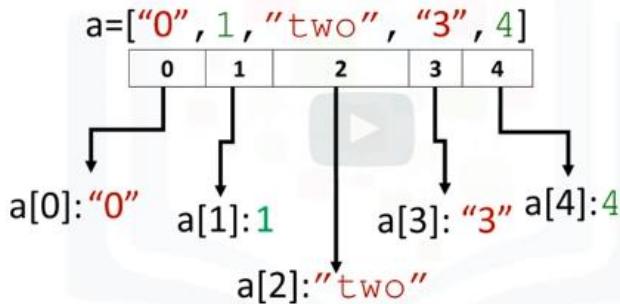
Check out the labs and numpy.org for more.



Objectives

- The Basics and Array Creation
- Indexing and Slicing
- Basic Operations
- Universal Functions

The Basics & Array Creation



To know the type of the array: ex. a.dtype: dtype('int64')

```
a:array([0, 1, 2, 3, 4])
```

```
type(a): numpy.ndarray
```

```
a.dtype: dtype('int64')
```

```
a=np.array([0, 1, 2, 3, 4])
```

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

```
a.size :5
```

```
a.ndim: 1
```

```
a.shape: (5,)
```

```
b=np.array([3.1, 11.02, 6.2, 213.2, 5.2])
```

```
type(b): numpy.ndarray
```

```
b.dtype: dtype('float64')
```

Indexing and Slicing

```
c=np.array([20,1,2,3, 4])  
c:array([20,1,2,3, 4])
```

```
c[0]=100  
c:array([100,1,2,3,4])  
c[4]=0  
c:array([100,1,2,3,0])
```

c:array([100, 1, 2, 3, 0])

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|

d=c[1:4]

d:array([1, 2, 3])

c:array([100, 1, 2, 3, 0])

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|

c[3:5]=300,400

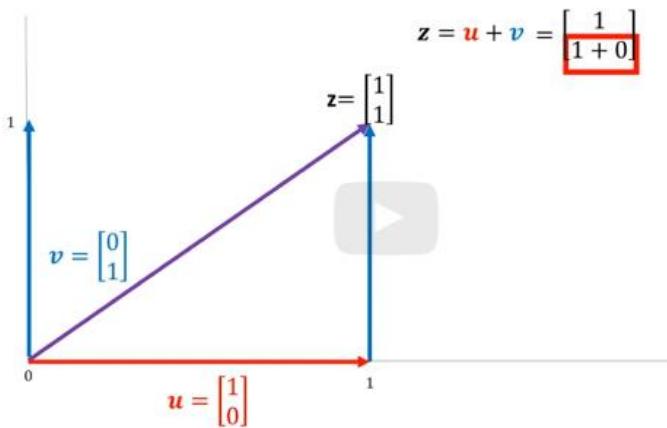
c:array([100, 1, 2, 300, 400])

Basic Operations

Vector Addition and Subtraction

$$\mathbf{u} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\mathbf{z} = \mathbf{u} + \mathbf{v} = \begin{bmatrix} 1+0 \\ 0+1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



$u=[1, 0]$

$v=[0, 1]$

$z=[]$

for n, m **in** zip(u,v):

 z.append(n+m)

Vector Addition

$u=np.array([1, 0])$

$v=np.array([0, 1])$

$z=u+v$

$z:array([1, 1])$

$u=[1, 0]$

$v=[0, 1]$

$z=[]$

for n, m **in** zip(u,v):

 z.append(n+m)

Vector Subtraction

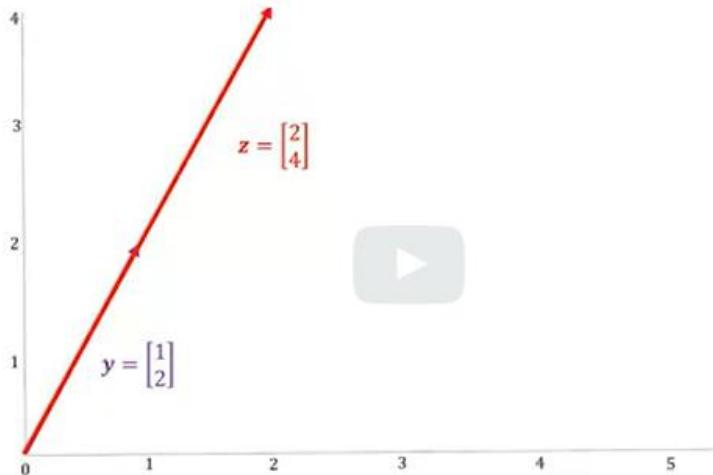
```
u=np.array([1,0])  
v=np.array([0,1])  
  
z=u-v  
z=array([1,-1])
```

```
u=[1, 0]  
v=[0, 1]  
z=[ ]  
  
for n, m in zip(u,v):  
    z.append(n-m)
```

Array multiplication with a Scalar

$$y = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$z = 2y = \begin{bmatrix} 2(1) \\ 2(2) \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$



```
y=np.array([1,2])
```

```
z=2*y  
z=array([2,4])
```

```
y=[1, 2]  
z=[ ]
```

```
for n in y:
```

```
    z.append(2*n)
```

Hadamard Product

Product of two numpy arrays

$$\mathbf{u} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$\mathbf{z} = \mathbf{u} \circ \mathbf{v} = \begin{bmatrix} 1*3 \\ 2*2 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

u=np.array([1,2])
v=np.array([3,2])

z=u*v

z:array([3, 4])

u=[1, 2]

v=[3, 2]

z=[]

for n, m in zip(u,v):

z.append(n*m)

Dot Product

$$\mathbf{u} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \mathbf{v} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$\mathbf{u}^T \mathbf{v} = \boxed{1} \times \boxed{3} + \boxed{2} \times \boxed{1} = 5$$

u=np.array([1,2])

v=np.array([3,1])

result =np.dot(u,v)

result :5

Adding Constant to an numpy Array

```
u=np.array([1,2,3,-1])
```

```
z=u+1
```

```
z:array([2,3,4,0])
```

1, 2, 3, -1



1+1, 2+1, 3+1, -1+1



Universal Functions

```
a=np.array([1,-1,1,-1])
```

```
mean_a=a.mean()
```

```
mean_a:0.0
```

$$\frac{1}{4} (1 - 1 + 1 - 1)$$

$$=0$$

Universal Functions

```
b=np.array([1, -2,3,4,5])
```

```
max_b=b.max()
```

```
max_b:5
```

Universal Functions

```
np.pi  
x=np.array([ 0 , np.pi/2, np.pi ] )  
y=np.sin(x)  
y:array([ 0,1, 1.2e-16])
```

π
 $x = [0, \frac{\pi}{2}, \pi]$
 $y = [\sin(0), \sin(\frac{\pi}{2}), \sin(\pi)]$
 $y = [0, 1, 0]$

Linspace

```
np.linspace(-2,2,num=5)
```

| | | | | |
|----|----|---|---|---|
| -2 | -1 | 0 | 1 | 2 |
|----|----|---|---|---|

```
np.linspace(-2,2,num=5)
```

| | | | | |
|----|----|---|---|---|
| -2 | -1 | 0 | 1 | 2 |
| 1 | 2 | 3 | 4 | 5 |

```
np.linspace(-2,2,num=5)
```

| | | | | |
|----|----|---|---|---|
| -2 | -1 | 0 | 1 | 2 |
| 1 | 2 | 3 | 4 | 5 |

```
np.linspace(-2,2,num=9)
```

| | | | | | | | | |
|----|------|----|------|---|-----|---|-----|---|
| -2 | -1.5 | -1 | -0.5 | 0 | 0.5 | 1 | 1.5 | 2 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

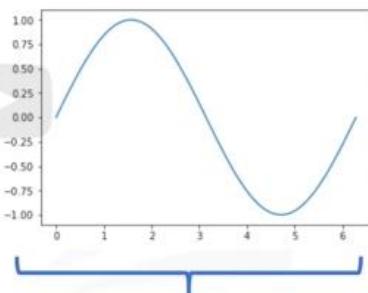
```
np.linspace(-2,2,num=9)
```

| | | | | | | | | |
|----|------|----|------|---|-----|---|-----|---|
| -2 | -1.5 | -1 | -0.5 | 0 | 0.5 | 1 | 1.5 | 2 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Plotting Mathematical Functions

```
x=np.linspace( 0 , 2*np.pi,100)  
y=np.sin(x)
```

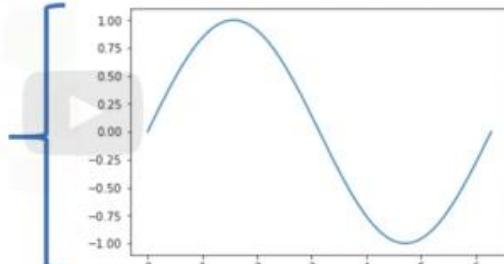
```
import matplotlib.pyplot as plt  
%matplotlib inline  
plt.plot(x,y)
```



Plotting Mathematical Functions

```
x=np.linspace( 0 , 2*np.pi,100)  
y=np.sin(x)
```

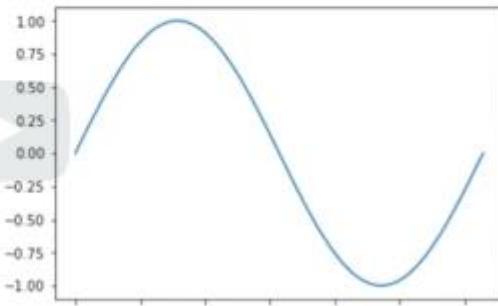
```
import matplotlib.pyplot as plt  
%matplotlib inline  
plt.plot(x,y)
```



Plotting Mathematical Functions

```
x=np.linspace( 0 , 2*np.pi,100)  
y=np.sin(x)
```

```
import matplotlib.pyplot as plt  
%matplotlib inline  
plt.plot(x,y)
```



1D Numpy in Python

Estimated time needed: 30 minutes

Objectives

After completing this lab you will be able to:

Import and use numpy library

Perform operations with numpy

Table of Contents

- Preparation
- What is Numpy?
 - Type
 - Assign Value
 - Slicing
 - Assign Value with List
 - Other Attributes
- Numpy Array Operations
 - Array Addition
 - Array Multiplication
 - Product of Two Numpy Arrays
 - Dot Product
 - Adding Constant to a Numpy Array
- Mathematical Functions
- Linspace

Preparation

```
In [1]: # Import the Libraries
```

```
import time
import sys
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
```

```
In [2]: # Plotting functions
```

```
def Plotvec1(u, z, v):

    ax = plt.axes()
    ax.arrow(0, 0, *u, head_width=0.05, color='r', head_length=0.1)
    plt.text(*(u + 0.1), 'u')

    ax.arrow(0, 0, *v, head_width=0.05, color='b', head_length=0.1)
    plt.text(*(v + 0.1), 'v')
    ax.arrow(0, 0, *z, head_width=0.05, head_length=0.1)
    plt.text(*(z + 0.1), 'z')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)

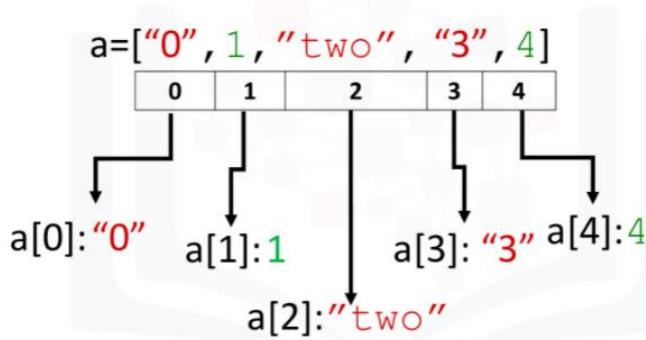
def Plotvec2(a,b):
    ax = plt.axes()
    ax.arrow(0, 0, *a, head_width=0.05, color ='r', head_length=0.1)
    plt.text(*(a + 0.1), 'a')
    ax.arrow(0, 0, *b, head_width=0.05, color ='b', head_length=0.1)
    plt.text(*(b + 0.1), 'b')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)
```

Create a Python List as follows:

```
In [3]: # Create a python list
```

```
a = ["0", 1, "two", "3", 4]
```

We can access the data via an index:



We can access each element using a square bracket as follows:

```
In [4]: # Print each element

print("a[0]:", a[0])
print("a[1]:", a[1])
print("a[2]:", a[2])
print("a[3]:", a[3])
print("a[4]:", a[4])
```

```
a[0]: 0
a[1]: 1
a[2]: two
a[3]: 3
a[4]: 4
```

What is Numpy?

A **numpy array** is similar to a list. It's usually fixed in size and each element is of the same type. We can cast a list to a numpy array by first importing numpy:

```
In [5]: # import numpy library

import numpy as np
```

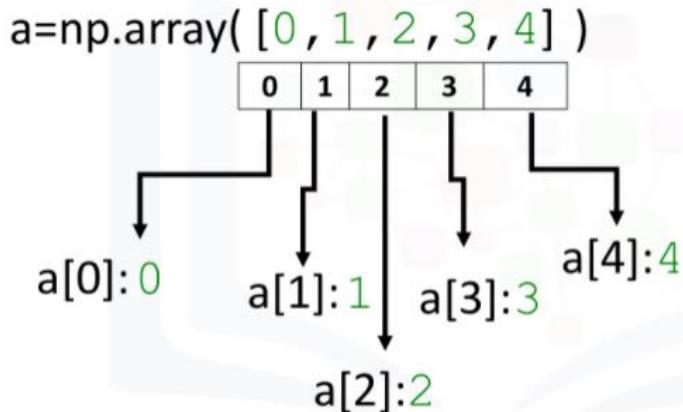
We then cast the list as follows:

```
In [6]: # Create a numpy array

a = np.array([0, 1, 2, 3, 4])
a
```

```
Out[6]: array([0, 1, 2, 3, 4])
```

Each element is of the same type, in this case integers:



As with lists, we can access each element via a square bracket:

```
In [7]: # Print each element

print("a[0]:" , a[0])
print("a[1]:" , a[1])
print("a[2]:" , a[2])
print("a[3]:" , a[3])
print("a[4]:" , a[4])
```

```
a[0]: 0
a[1]: 1
a[2]: 2
a[3]: 3
a[4]: 4
```

Type

If we check the type of the array we get `numpy.ndarray`:

```
In [8]: # Check the type of the array

type(a)
```

```
Out[8]: numpy.ndarray
```

As numpy arrays contain data of the same type, we can use the attribute "dtype" to obtain the Data-type of the array's elements. In this case a 64-bit integer:

```
In [9]: # Check the type of the values stored in numpy array  
a.dtype  
  
Out[9]: dtype('int64')
```

We can create a numpy array with real numbers:

```
In [10]: # Create a numpy array  
b = np.array([3.1, 11.02, 6.2, 213.2, 5.2])
```

When we check the type of the array we get **numpy.ndarray**:

```
In [11]: # Check the type of array  
type(b)  
  
Out[11]: numpy.ndarray
```

If we examine the attribute `dtype` we see float 64, as the elements are not integers:

```
In [12]: # Check the value type  
b.dtype  
  
Out[12]: dtype('float64')
```

Assign value

We can change the value of the array, consider the array c:

```
In [26]: # Create numpy array  
  
c = np.array([20, 1, 2, 3, 4])  
c
```

```
Out[26]: array([20, 1, 2, 3, 4])
```

We can change the first element of the array to 100 as follows:

```
In [27]: # Assign the first element to 100  
  
c[0] = 100  
c
```

```
Out[27]: array([100, 1, 2, 3, 4])
```

We can change the 5th element of the array to 0 as follows:

```
In [28]: # Assign the 5th element to 0  
  
c[4] = 0  
c
```

```
Out[28]: array([100, 1, 2, 3, 0])
```

Slicing

Like lists, we can slice the numpy array, and we can select the elements from 1 to 3 and assign it to a new numpy array d as follows:

```
In [29]: # Slicing the numpy array  
  
d = c[1:4]  
d
```

```
Out[29]: array([1, 2, 3])
```

We can assign the corresponding indexes to new values as follows:

```
In [30]: # Set the fourth element and fifth element to 300 and 400  
  
c[3:5] = 300, 400  
c
```

```
Out[30]: array([100, 1, 2, 300, 400])
```

Assign Value with List

Similarly, we can use a list to select a specific index. The list 'select' contains several values:

```
In [31]: # Create the index list  
  
select = [0, 2, 4]
```

We can use the list as an argument in the brackets. The output is the elements corresponding to the particular index:

```
In [32]: # Use List to select elements  
  
d = c[select]  
d  
  
Out[32]: array([100, 2, 400])
```

We can assign the specified elements to a new value. For example, we can assign the values to 100 000 as follows:

```
In [33]: # Assign the specified elements to new value  
  
c[select] = 100000  
c  
  
Out[33]: array([100000, 1, 100000, 300, 100000])
```

Other Attributes

Let's review some basic array attributes using the array `a`:

Other Attributes

Let's review some basic array attributes using the array `a`:

```
In [34]: # Create a numpy array  
  
a = np.array([0, 1, 2, 3, 4])  
a  
  
Out[34]: array([0, 1, 2, 3, 4])
```

The attribute `size` is the number of elements in the array:

```
In [35]: # Get the size of numpy array  
  
a.size  
  
Out[35]: 5
```

The next two attributes will make more sense when we get to higher dimensions but let's review them. The attribute `ndim` represents the number of array dimensions or the rank of the array, in this case, one:

```
In [36]: # Get the number of dimensions of numpy array  
a.ndim
```

```
Out[36]: 1
```

The attribute `shape` is a tuple of integers indicating the size of the array in each dimension:

```
In [37]: # Get the shape/size of numpy array  
a.shape
```

```
Out[37]: (5,)
```

```
In [38]: # Create a numpy array  
  
a = np.array([1, -1, 1, -1])
```

```
In [39]: # Get the mean of numpy array  
  
mean = a.mean()  
mean
```

```
Out[39]: 0.0
```

```
In [40]: # Get the standard deviation of numpy array  
  
standard_deviation=a.std()  
standard_deviation
```

```
Out[40]: 1.0
```

```
In [41]: # Create a numpy array  
  
b = np.array([-1, 2, 3, 4, 5])  
b
```

```
Out[41]: array([-1, 2, 3, 4, 5])
```

```
In [42]: # Get the biggest value in the numpy array  
  
max_b = b.max()  
max_b
```

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

```
In [43]: # Get the smallest value in the numpy array  
  
min_b = b.min()  
min_b
```

```
Out[43]: -1
```

Numpy Array Operations

Array Addition

Consider the numpy array u:

```
In [44]: u = np.array([1, 0])  
u
```

```
Out[44]: array([1, 0])
```

Consider the numpy array `v`:

```
In [45]: v = np.array([0, 1])  
v
```

```
Out[45]: array([0, 1])
```

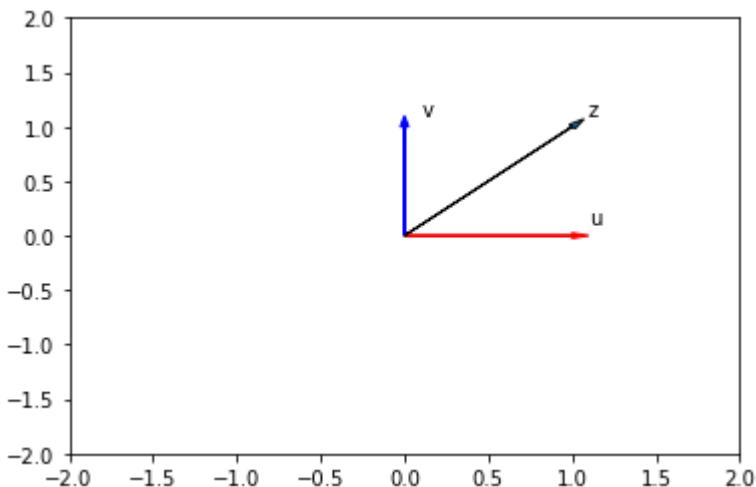
We can add the two arrays and assign it to `z`:

```
In [46]: # Numpy Array Addition  
  
z = u + v  
z
```

```
Out[46]: array([1, 1])
```

The operation is equivalent to vector addition:

```
In [47]: # Plot numpy arrays  
  
Plotvec1(u, z, v)
```



Array Multiplication

Consider the vector numpy array `y`:

```
In [48]: # Create a numpy array  
  
y = np.array([1, 2])  
y
```

```
Out[48]: array([1, 2])
```

We can multiply every element in the array by 2:

```
In [49]: # Numpy Array Multiplication  
  
z = 2 * y  
z
```

```
Out[49]: array([2, 4])
```

This is equivalent to multiplying a vector by a scalar:

Product of Two Numpy Arrays

Consider the following array u:

```
In [50]: # Create a numpy array  
  
u = np.array([1, 2])  
u
```

```
Out[50]: array([1, 2])
```

Consider the following array v :

```
In [51]: # Create a numpy array  
  
v = np.array([3, 2])  
v
```

```
Out[51]: array([3, 2])
```

The product of the two numpy arrays u and v is given by:

```
In [54]: # Calculate the production of two numpy arrays  
  
z = u * v  
z
```

```
Out[54]: array([3, 4])
```

Dot Product

The dot product of the two numpy arrays u and v is given by:

```
In [56]: # Calculate the dot product  
np.dot(u, v)
```

```
Out[56]: 7
```

Adding Constant to a Numpy Array

Consider the following array:

```
In [57]: # Create a constant to numpy array  
  
u = np.array([1, 2, 3, -1])  
u
```

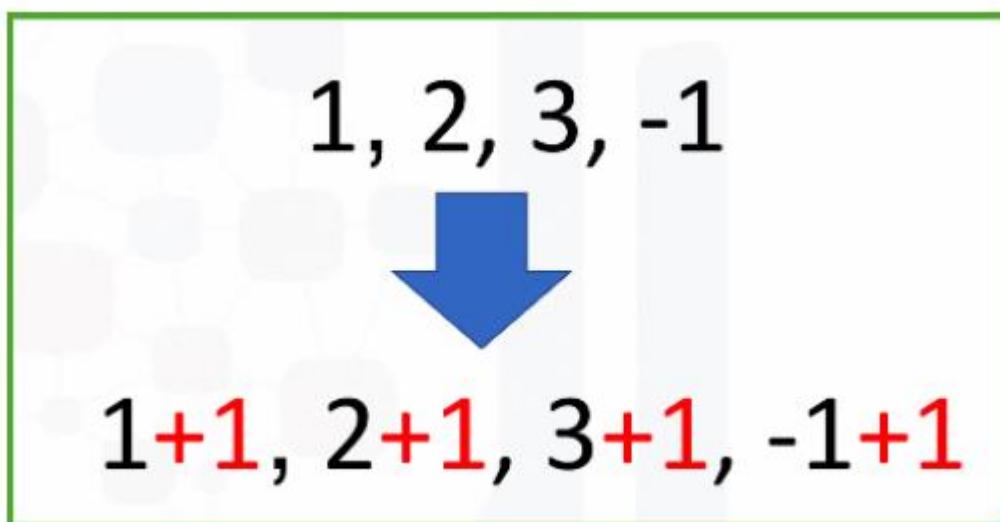
```
Out[57]: array([ 1,  2,  3, -1])
```

Adding the constant 1 to each element in the array:

```
In [58]: # Add the constant to array  
  
u + 1
```

```
Out[58]: array([2, 3, 4, 0])
```

The process is summarised in the following animation:



Mathematical Functions

We can access the value of `pi` in numpy as follows :

```
In [59]: # The value of pi  
np.pi
```

```
Out[59]: 3.141592653589793
```

We can create the following numpy array in Radians:

```
In [60]: # Create the numpy array in radians  
x = np.array([0, np.pi/2 , np.pi])
```

We can apply the function `sin` to the array `x` and assign the values to the array `y` ; this applies the sine function to each element in the array:

```
In [61]: # Calculate the sin of each elements  
y = np.sin(x)  
y
```

```
Out[61]: array([0.000000e+00, 1.000000e+00, 1.2246468e-16])
```

Linspace

A useful function for plotting mathematical functions is `linspace`. Linspace returns evenly spaced numbers over a specified interval. We specify the starting point of the sequence and the ending point of the sequence. The parameter "num" indicates the Number of samples to generate, in this case 5:

```
In [62]: # Makeup a numpy array within [-2, 2] and 5 elements  
np.linspace(-2, 2, num=5)
```

```
Out[62]: array([-2., -1., 0., 1., 2.])
```

If we change the parameter `num` to 9, we get 9 evenly spaced numbers over the interval from -2 to 2:

```
In [63]: # Makeup a numpy array within [-2, 2] and 9 elements  
np.linspace(-2, 2, num=9)
```

```
Out[63]: array([-2. , -1.5, -1. , -0.5, 0. , 0.5, 1. , 1.5, 2. ])
```

We can use the function `linspace` to generate 100 evenly spaced samples from the interval 0 to 2π :

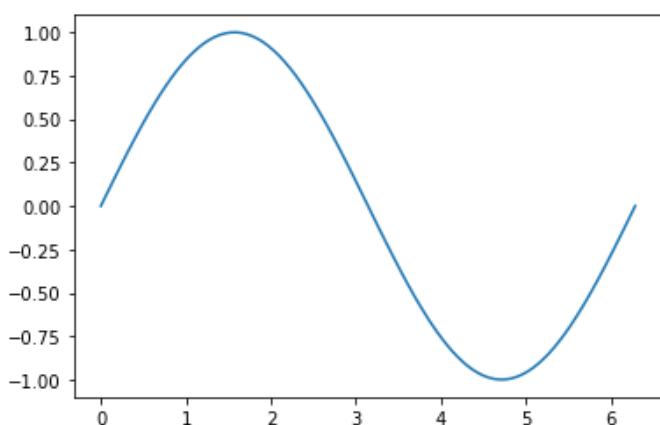
```
In [64]: # Makeup a numpy array within [0, 2π] and 100 elements  
x = np.linspace(0, 2*np.pi, num=100)
```

We can apply the sine function to each element in the array `x` and assign it to the array `y`:

```
In [65]: # Calculate the sine of x list  
y = np.sin(x)
```

```
In [66]: # Plot the result  
plt.plot(x, y)
```

```
Out[66]: [<matplotlib.lines.Line2D at 0x7ffb71d89cc0>]
```



Quiz on 1D Numpy Array

Implement the following vector subtraction in numpy: $u-v$

```
In [70]: # Write your code below and press Shift+Enter to execute  
  
u = np.array([1, 0])  
v = np.array([0, 1])  
  
z = u - v  
z
```

Out[70]: array([1, -1])

[Click here for the solution](#) `python u - v`

Multiply the numpy array z with -2:

```
In [72]: # Write your code below and press Shift+Enter to execute  
  
z = np.array([2, 4])  
  
z * -2
```

Out[72]: array([-4, -8])

[Click here for the solution](#) `python -2 * z`

Consider the list $[1, 2, 3, 4, 5]$ and $[1, 0, 1, 0, 1]$, and cast both lists to a numpy array then multiply them together:

```
In [75]: # Write your code below and press Shift+Enter to execute  
a = np.array([1, 2, 3, 4, 5])  
b = np.array([1, 0, 1, 0, 1])  
a * b
```

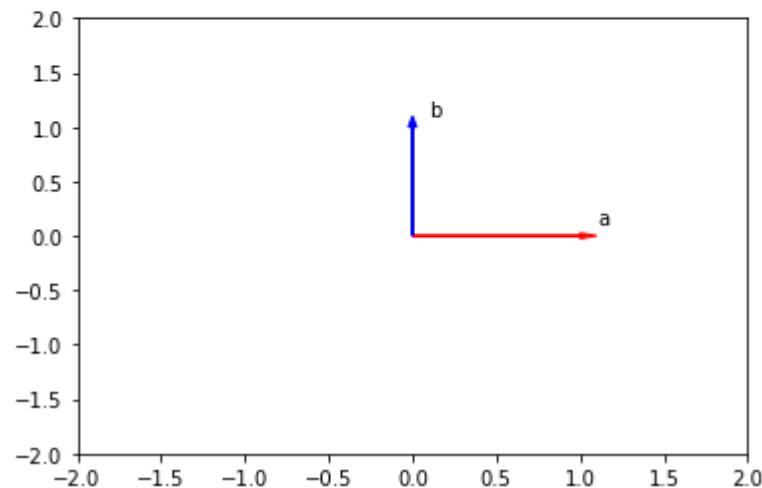
Out[75]: array([1, 0, 3, 0, 5])

[Click here for the solution](#) `python a = np.array([1, 2, 3, 4, 5]) b = np.array([1, 0, 1, 0, 1]) a * b`

Convert the list $[-1, 1]$ and $[1, 1]$ to numpy arrays a and b . Then, plot the arrays as vectors using the function Plotvec2 and find the dot product:

```
In [81]: # Write your code below and press Shift+Enter to execute
a = np.array([1, 0])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product:", np.dot(a, b))
```

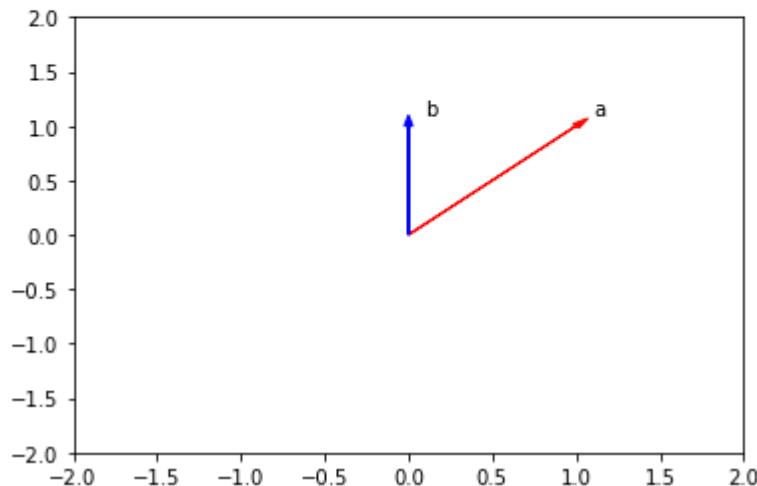
The dot product: 0



Convert the list $[1, 1]$ and $[0, 1]$ to numpy arrays a and b. Then plot the arrays as vectors using the function Plotvec2 and find the dot product:

```
In [82]: # Write your code below and press Shift+Enter to execute
a = np.array([1, 1])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product:", np.dot(a, b))
```

The dot product: 1



Click here for the solution ````python a = np.array([1, 1]) b = np.array([0, 1]) Plotvec2(a, b) print("The dot product is", np.dot(a, b)) print("The dot product is", np.dot(a, b))````

Why are the results of the dot product for $[-1, 1]$ and $[1, 1]$ and the dot product for $[1, 0]$ and $[0, 1]$ zero, but not zero for the dot product for $[1, 1]$ and $[0, 1]$?

Hint: Study the corresponding figures, pay attention to the direction the arrows are pointing to.

In []: `# Write your code below and press Shift+Enter to execute`

Click here for the solution ````python The vectors used for question 4 and 5 are perpendicular. As a result, the dot product is zero.````

Additional

Quiz on 1D Numpy Array

Implement the following vector subtraction in numpy: u-v

```
[51]: # Write your code below and press Shift+Enter to execute  
  
u = np.array([1, 0])  
v = np.array([0, 1])  
  
z = u - v  
z
```

```
[51]: array([ 1, -1])
```

► [Click here for the solution](#)

Multiply the numpy array z with -2:

```
[52]: # Write your code below and press Shift+Enter to execute  
  
z = np.array([2, 4])  
  
z * -2
```

```
[52]: array([-4, -8])
```

► [Click here for the solution](#)

Consider the list [1, 2, 3, 4, 5] and [1, 0, 1, 0, 1]. Cast both lists to a numpy array then multiply them together:

```
[53]: # Write your code below and press Shift+Enter to execute  
a = np.array([1, 2, 3, 4, 5])  
b = np.array([1, 0, 1, 0, 1])  
a * b
```

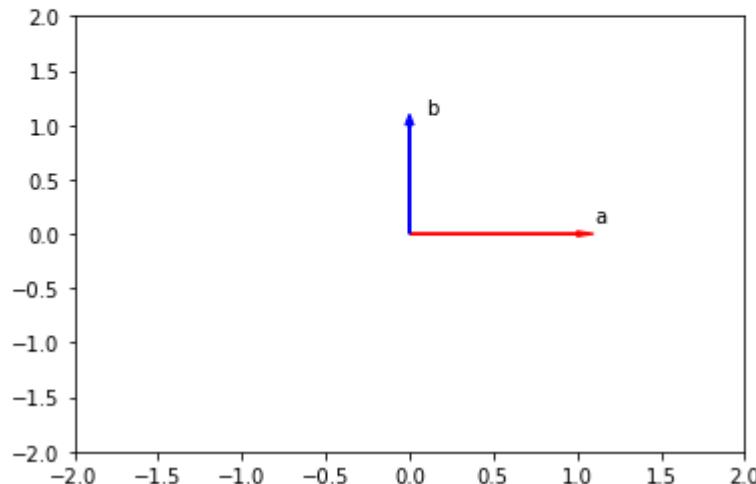
```
[53]: array([1, 0, 3, 0, 5])
```

► [Click here for the solution](#)

Convert the list $[-1, 1]$ and $[1, 1]$ to numpy arrays a and b. Then, plot the arrays as vectors using the function Plotvec2 and find their dot product:

```
[54]: # Write your code below and press Shift+Enter to execute
a = np.array([1, 0])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product:", np.dot(a, b))
```

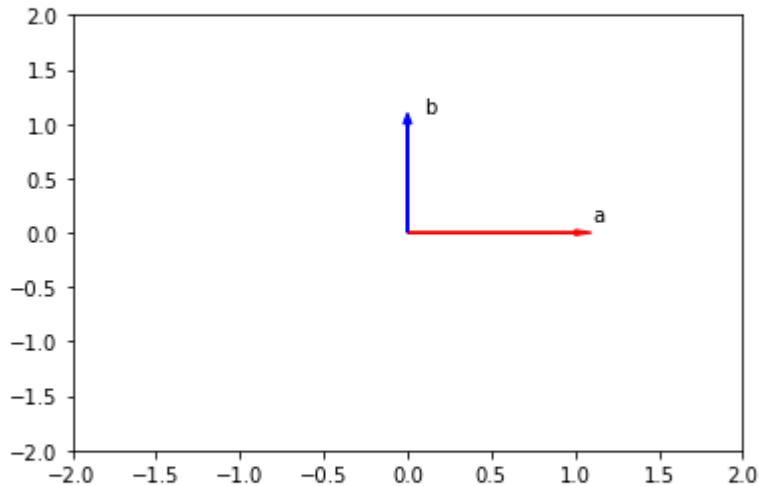
The dot product: 0



Convert the list $[1, 0]$ and $[0, 1]$ to numpy arrays a and b. Then, plot the arrays as vectors using the function Plotvec2 and find their dot product:

```
[55]: # Write your code below and press Shift+Enter to execute
a = np.array([1, 0])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a, b))
```

The dot product is 0



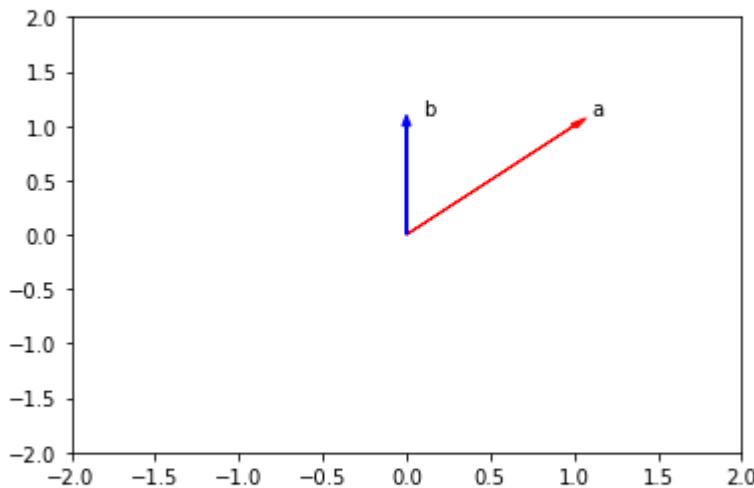
▼ Click here for the solution

```
a = np.array([1, 0])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a, b))
```

Convert the list [1, 1] and [0, 1] to numpy arrays a and b. Then plot the arrays as vectors using the function Plotvec2 and find their dot product:

```
[56]: # Write your code below and press Shift+Enter to execute
a = np.array([1, 1])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a, b))
```

The dot product is 1



▼ Click here for the solution

```
a = np.array([1, 1])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a, b))
```

Why are the results of the dot product for $[-1, 1]$ and $[1, 1]$ and the dot product for $[1, 0]$ and $[0, 1]$ zero, but not zero for the dot product for $[1, 1]$ and $[0, 1]$?

Hint: Study the corresponding figures, pay attention to the direction the arrows are pointing to.

```
[ ]: # Write your code below and press Shift+Enter to execute
```

▼ Click here for the solution

The vectors used for question 4 and 5 are perpendicular. As a result, the dot product is zero.

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python.

Two Dimensional NumPy

We can create numpy arrays with more than one dimension. This section will focus only on 2D arrays but you can use numpy to build arrays of much higher dimensions.

In this video, we will cover the basics and array creation in 2D, indexing and slicing in 2D, and basic operations in 2D.

Consider the list `a`, the list contains three nested lists each of equal size. Each list is color-coded for simplicity. We can cast the list to a numpy array as follows. It is helpful to visualize the numpy array as a rectangular array each nested lists corresponds to a different row of the matrix.

We can use the attribute `ndim` to obtain the number of axes or dimensions referred to as the rank. The term rank does not refer to the number of linearly independent columns like a matrix.

It's useful to think of `ndim` as the number of nested lists. The first list represents the first dimension. This list contains another set of lists. This represents the second dimension or axis.

The number of lists the list contains does not have to do with the dimension but the shape of the list. As with a 1D array, the attribute `shape` returns a tuple. It's helpful to use the rectangular representation as well. The first element in the tuple corresponds to the number of nested lists contained in the original list or the number of rows in the rectangular representation, in this case three.

The second element corresponds to the size of each of the nested list or the number of columns in the rectangular array zero. The convention is to label this axis zero and this axis one as follows.

We can also use the attribute `size` to get the size of the array. We see there are three rows and three columns.

Multiplying the number of columns and rows together, we get the total number of elements, in this case nine. Check out the labs for arrays of different shapes and other attributes.

We can use rectangular brackets to access the different elements of the array. The following image demonstrates the relationship between the indexing conventions for the lists like representation. The index in the first bracket corresponds to the different nested lists each a different color. The second bracket corresponds to the index of a particular element within the nested list.

Using the rectangular representation, the first index corresponds to the row index.

The second index corresponds to the column index. We could also use a single bracket to access the elements as follows. Consider the following syntax. This index corresponds to the second row, and this index the third column, the value is 23. Consider this example, this index corresponds to the first row and the second index corresponds to the first column, and a value of 11.

We can also **use slicing in numpy arrays**. The first index corresponds to the first row.

The second index accesses the first two columns. Consider this example, the first index corresponds to the first two rows.

The second index accesses the last column. We can also add arrays, the process is identical to matrix addition. Consider the matrix X, each element is colored differently.

Consider the matrix Y. Similarly, each element is colored differently. We can add the matrices.

This corresponds to adding the elements in the same position, i.e adding elements contained in the same color boxes together. The result is a new matrix that has the same size as matrix Y or X.

Each element in this new matrix is the sum of the corresponding elements in X and Y.

To add two arrays in numpy, we define the array in this case X. Then we define the second array Y, we add the arrays.

The result is identical to matrix addition. Multiplying a numpy array by a scalar is identical to multiplying a matrix by a scalar. Consider the matrix Y. If we multiply the matrix by this scalar two, we simply multiply every element in the matrix by two.

The result is a new matrix of the same size where each element is multiplied by two.

Consider the array Y. We first define the array, we multiply the array by a scalar as follows and assign it to the variable Z. The result is a new array where each element is multiplied by two.

Multiplication of two arrays corresponds to an element-wise product, or Hadamard product.

Consider array X and array Y. Hadamard product corresponds to multiplying each of the elements in the same position i.e multiplying elements contained in the same color boxes together.

The result is a new matrix that is the same size as matrix Y or X. Each element in this new matrix is the product of the corresponding elements in X and Y.

Consider the array X and Y. We can find the product of two arrays X and Y in one line, and assign it to the variable Z as follows. The result is identical to Hadamard product.

We can also perform matrix multiplication with Numpy arrays. Matrix multiplication is a little more complex but let's provide a basic overview. Consider the matrix A where each row is a different color.

Also, consider the matrix B where each column is a different color. In linear algebra, before we multiply matrix A by matrix B, we must make sure that the number of columns in matrix A in this case three is equal to the number of rows in matrix B, in this case three.

From matrix multiplication, to obtain the i th row and j th column of the new matrix, we take the dot product of the i th row of A with the j th columns of B.

For the first column, first row we take the dot product of the first row of A with the first column of B as follows.

The result is zero. For the first row and the second column of the new matrix, we take the dot product of the first row of the matrix A, but this time we use the second column of matrix B, the result is two.

For the second row and the first column of the new matrix, we take the dot product of the second row of the matrix A. With the first column of matrix B, the result is zero.

Finally, for the second row and the second column of the new matrix, we take the dot product of the second row of the matrix A with the second column of matrix B,

the result is two.

In numpy, we can define the numpy arrays A and B. We can perform matrix multiplication and assign it to array C. The result is the array C. It corresponds to the matrix multiplication of array A and B.

There is a lot more you can do with it in numpy.

Python

Two Dimensional Numpy

Table of Contents

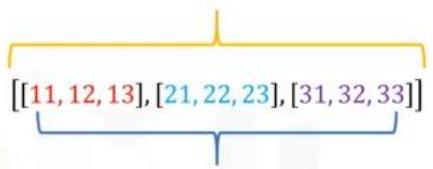
- The Basics and Array Creation in 2D
- Indexing and Slicing in 2D
- Basic Operations in 2D

$a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$

A = np.array(a)

A: $\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$

A.ndim:2



A.ndim:2

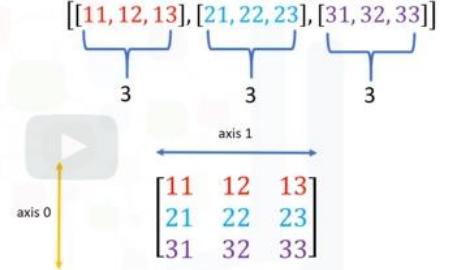
A.shape: (3,3)

3
[[11, 12, 13], [21, 22, 23], [31, 32, 33]]

A.ndim:2

A.shape: (3,3)

3
 $\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$



A.ndim:2

A.shape: (3,3)

A.size : 9

[[11, 12, 13], [21, 22, 23], [31, 32, 33]]

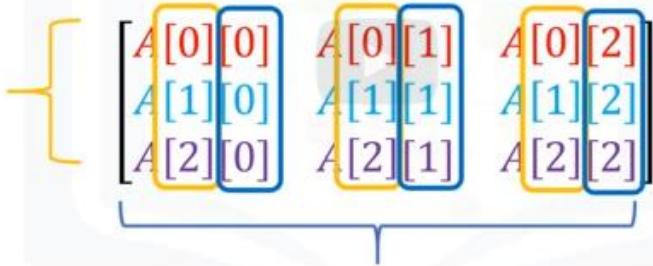
3x3=9

3
 $\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$
3

Yellow corresponds to the row index

Blue corresponds to the column index

$A: [[A[0][0], A[0][1], A[0][2]], [A[1][0], A[1][1], A[1][2]], [A[2][0], A[2][1], A[2][2]]]$



$$A = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$$

$A[1][2]: 23$

| | | | 0 | 1 | 2 |
|--|--|---|----|----|----|
| | | 0 | 11 | 12 | 13 |
| | | 1 | 21 | 22 | 23 |
| | | 2 | 31 | 32 | 33 |

Example 4.2

*Yellow "1" is the second row Yellow "2" is the 3rd column and the value is 23

$$A = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$$

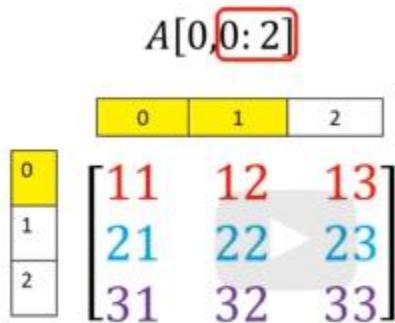
$A[0][0] : 11$

| | | | 0 | 1 | 2 |
|--|--|---|----|----|----|
| | | 0 | 11 | 12 | 13 |
| | | 1 | 21 | 22 | 23 |
| | | 2 | 31 | 32 | 33 |

The index above corresponds to the first row and the second index corresponds to the first column, and a value of 11

Slicing

The first index below corresponds to the first row. The second index accesses the first two columns.



The first index below corresponds to the first two rows. The second index accesses the last column.

$$A = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$$

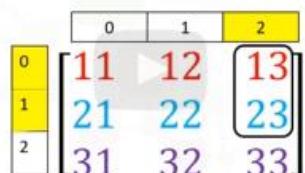


Example 4.2

The first index below corresponds to the first two rows. The second index accesses the last column.

$$A = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$$

$$A[0:2,2]:\text{array}([13, 23])$$



Example 4.2

Add Arrays

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X + Y = \begin{bmatrix} 1+2 & 0+1 \\ 0+1 & 1+2 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

Add 2 Arrays in Numpy

```
X=np.array([[1,0],[0,1]])
Y=np.array([[2,1],[1,2]])
Z=X+Y;
Z=array([[3,1],
          [1,3]])
```

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad Z = X + Y \quad Z = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

Multiplying Arrays in a Scalar

Multiplying a numpy array by a scalar is identical to multiplying a matrix by a scalar

$$Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$2Y = \begin{bmatrix} 2x2 & 2x1 \\ 2x1 & 2x2 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

Multiply the array by a scalar as follows and assign it to the variable Z. The result is a new array where each element is multiplied by two.

```
Y=np.array([[2,1],[1,2]])
```

```
Z=2*Y;
```

```
Z=array([[4,2],  
[2,4]])
```

$$Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$Z = 2Y = \begin{bmatrix} (2)2 & (2)1 \\ (2)1 & (2)2 \end{bmatrix}$$

$$Z = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

Hadamard product corresponds to multiplying each of the elements in the same position i.e multiplying elements contained in the same color boxes together. The result is a new matrix that is the same size as matrix Y or X. Each element in this new matrix is the product of the corresponding elements in X and Y.

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X \circ Y = \begin{bmatrix} (1)2 & (0)1 \\ (0)1 & (1)2 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

```
X=np.array([[1,0],[0,1]])
```

```
Y=np.array([[2,1][1,2]])
```

```
Z=X*Y;
```

```
Z=array([[2,0],  
[0,2]])
```

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$Z = X \circ Y = \begin{bmatrix} (1)2 & (0)1 \\ (0)1 & (1)2 \end{bmatrix}$$

$$Z = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Matrix Multiplication with Numpy Arrays

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix}$$

$A B = []$

$0 \times 1 + 1 \times 1$

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix}$$

$0 \times 1 + 1 \times 1 + 1 \times 1 = 2$

$$A B = \begin{bmatrix} 0 & 2 \end{bmatrix} \quad A B = \begin{bmatrix} 0 & 2 \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix}$$

$1 \times 1 + 0 \times 1 + (1) \times 1 = 2$

$$A B = \begin{bmatrix} 0 & 2 \\ 0 & 2 \end{bmatrix}$$

```

A=np.array([[0,1,1],[1,0,1]])
B=np.array([[1,1],[1,1],[-1,1]])
C=np.dot(A,B);
C=array([[0,2],
[0,2]])

```



2D Numpy in Python

Estimated time needed: 20 minutes

Objectives

After completing this lab you will be able to:

Operate comfortably with numpy

Perform complex operations with numpy

Table of Contents

- Create a 2D Numpy Array
- Accessing different elements of a Numpy Array
- Basic Operations

Create a 2D Numpy Array

In [1]:

```
# Import the Libraries  
  
import numpy as np  
import matplotlib.pyplot as plt
```

Consider the list `a`, the list contains three nested lists **each of equal size**.

In [2]:

```
# Create a List  
  
a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]  
a
```

Out[2]:

```
[[11, 12, 13], [21, 22, 23], [31, 32, 33]]
```

We can cast the list to a Numpy Array as follow

In [3]:

```
# Convert List to Numpy Array  
# Every element is the same type  
  
A = np.array(a)  
A
```

Out[3]:

```
array([[11, 12, 13],  
       [21, 22, 23],  
       [31, 32, 33]])
```

We can use the attribute `ndim` to obtain the number of axes or dimensions referred to as the rank.

```
In [4]: # Show the numpy array dimensions  
A.ndim
```

```
Out[4]: 2
```

Attribute `shape` returns a tuple corresponding to the size or number of each dimension.

```
In [5]: # Show the numpy array shape  
A.shape
```

```
Out[5]: (3, 3)
```

The total number of elements in the array is given by the attribute `size`.

```
In [6]: # Show the numpy array size  
A.size
```

```
Out[6]: 9
```

Accessing different elements of a Numpy Array

We can use rectangular brackets to access the different elements of the array. The correspondence between the rectangular brackets and the list and the rectangular representation is shown in the following figure for a 3x3 array:

```
A: [[A[0,0], A[0,1], A[0,2]], [A[1,0], A[1,1], A[1,2]], [A[2,0], A[2,1], A[2,2]]]
```

$$\begin{bmatrix} A[0,0] & A[0,1] & A[0,2] \\ A[1,0] & A[1,1] & A[1,2] \\ A[2,0] & A[2,1] & A[2,2] \end{bmatrix}$$

We can access the 2nd-row 3rd column as shown in the following figure:

| | 0 | 1 | 2 |
|---|----|----|----|
| 0 | 11 | 12 | 13 |
| 1 | 21 | 22 | 23 |
| 2 | 31 | 32 | 33 |

We simply use the square brackets and the indices corresponding to the element we would like:

```
In [7]: # Access the element on the second row and third column  
A[1, 2]
```

```
Out[7]: 23
```

We can also use the following notation to obtain the elements:

```
In [8]: # Access the element on the second row and third column  
A[1][2]
```

```
Out[8]: 23
```

Consider the elements shown in the following figure

| | 0 | 1 | 2 |
|---|----|----|----|
| 0 | 11 | 12 | 13 |
| 1 | 21 | 22 | 23 |
| 2 | 31 | 32 | 33 |

We can access the element as follows

```
In [9]: # Access the element on the first row and first column  
A[0][0]
```

Out[9]: 11

We can also use slicing in numpy arrays. Consider the following figure. We would like to obtain the first two columns in the first row

| | 0 | 1 | 2 |
|---|----|----|----|
| 0 | 11 | 12 | 13 |
| 1 | 21 | 22 | 23 |
| 2 | 31 | 32 | 33 |

This can be done with the following syntax

```
In [10]: # Access the element on the first row and first and second columns
A[0][0:2]
```

```
Out[10]: array([11, 12])
```

Similarly, we can obtain the first two rows of the 3rd column as follows:

```
In [11]: # Access the element on the first and second rows and third column
A[0:2, 2]
```

```
Out[11]: array([13, 23])
```

Corresponding to the following figure:

| | 0 | 1 | 2 |
|---|----|----|----|
| 0 | 11 | 12 | 13 |
| 1 | 21 | 22 | 23 |
| 2 | 31 | 32 | 33 |

Basic Operations

We can also add arrays. The process is identical to matrix addition. Matrix addition of X and Y is shown in the following figure:

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X + Y = \begin{bmatrix} 1+2 & 0+1 \\ 0+1 & 1+2 \end{bmatrix} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

The numpy array is given by `X` and `Y`

```
In [12]: # Create a numpy array X  
  
X = np.array([[1, 0], [0, 1]])  
X
```

```
Out[12]: array([[1, 0],  
                 [0, 1]])  
  
In [13]: # Create a numpy array Y  
  
Y = np.array([[2, 1], [1, 2]])  
Y
```

```
Out[13]: array([[2, 1],  
                 [1, 2]])
```

We can add the numpy arrays as follows.

```
In [14]: # Add X and Y  
  
Z = X + Y  
Z
```

```
Out[14]: array([[3, 1],  
                 [1, 3]])
```

Multiplying a numpy array by a scalar is identical to multiplying a matrix by a scalar. If we multiply the matrix `Y` by the scalar 2, we simply multiply every element in the matrix by 2 as shown in the figure

$$Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$2Y = \begin{bmatrix} 2 \times 2 & 2 \times 1 \\ 2 \times 1 & 2 \times 2 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

We can perform the same operation in numpy as follows

```
In [15]: # Create a numpy array Y  
  
Y = np.array([[2, 1], [1, 2]])  
Y
```

```
Out[15]: array([[2, 1],  
                 [1, 2]])
```

```
In [16]: # Multiply Y with 2  
  
Z = 2 * Y  
Z
```

```
Out[16]: array([[4, 2],  
                 [2, 4]])
```

Multiplication of two arrays corresponds to an element-wise product or Hadamard product. Consider matrix X and Y. The Hadamard product corresponds to multiplying each of the elements in the same position, i.e. multiplying elements contained in the same color boxes together. The result is a new matrix that is the same size as matrix Y or X, as shown in the following figure.

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$X \circ Y = \begin{bmatrix} (1)2 & (0)1 \\ (0)1 & (1)2 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

We can perform element-wise product of the array X and Y as follows:

```
In [17]: # Create a numpy array Y
Y = np.array([[2, 1], [1, 2]])
Y
```

```
Out[17]: array([[2, 1],
 [1, 2]])
```

```
In [18]: # Create a numpy array X
X = np.array([[1, 0], [0, 1]])
X
```

```
Out[18]: array([[1, 0],
 [0, 1]])
```

```
In [19]: # Multiply X with Y
Z = X * Y
Z
```

```
Out[19]: array([[2, 0],
 [0, 2]])
```

We can also perform matrix multiplication with the numpy arrays A and B as follows:

First, we define matrix A and B:

```
In [20]: # Create a matrix A
A = np.array([[0, 1, 1], [1, 0, 1]])
A
```

```
Out[20]: array([[0, 1, 1],
 [1, 0, 1]])
```

```
In [21]: # Create a matrix B
B = np.array([[1, 1], [1, 1], [-1, 1]])
B
```

```
Out[21]: array([[ 1,  1],
 [ 1,  1],
 [-1,  1]])
```

We use the numpy function dot to multiply the arrays together.

```
In [22]: # calculate the dot product
Z = np.dot(A,B)
Z
```

```
Out[22]: array([[0, 2],
 [0, 2]])
```

```
In [23]: # Calculate the sine of Z
np.sin(Z)
```

```
Out[23]: array([[0.          , 0.90929743],
 [0.          , 0.90929743]])
```

We use the numpy attribute T to calculate the transposed matrix

```
In [24]: # Create a matrix C
C = np.array([[1,1],[2,2],[3,3]])
C
```

```
Out[24]: array([[1, 1],
 [2, 2],
 [3, 3]])
```

```
In [25]: # Get the transposed of C
C.T
```

```
Out[25]: array([[1, 2, 3],
 [1, 2, 3]])
```

Quiz on 2D Numpy Array

Consider the following list a, convert it to Numpy Array.

```
In [27]: # Write your code below and press Shift+Enter to execute  
a = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]  
A = np.array(a)  
A
```



```
Out[27]: array([[ 1,  2,  3,  4],  
                 [ 5,  6,  7,  8],  
                 [ 9, 10, 11, 12]])
```

Click here for the solution `python A = np.array(a) A`

Calculate the numpy array size.

```
In [28]: # Write your code below and press Shift+Enter to execute  
A.size
```



```
Out[28]: 12
```

Click here for the solution `python A.size`

Access the element on the first row and first and second columns.

```
In [ ]: # Write your code below and press Shift+Enter to execute  
A[0, 0:2]  
#A[0][0:2]
```

Click here for the solution `python A[0][0:2]`

Perform matrix multiplication with the numpy arrays `A` and `B`.

```
In [ ]: # Write your code below and press Shift+Enter to execute  
B = np.array([[0, 1], [1, 0], [1, 1], [-1, 0]])
```

Click here for the solution `python X = np.dot(A,B) X`

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](#) to learn how to share your work.

Practice Quiz: Numpy in Python

Bookmarked

Question 1

1/1 point (ungraded)

What is the Python library used for scientific computing and is a basis for Pandas?

- datetime
- Requests
- Tkinter
- Numpy



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 2

1/1 point (ungraded)

What attribute is used to retrieve the number of elements in an array?

- a.dtype
- a.ndim
- a.shape
- a.size



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 3

1/1 point (ungraded)

How would you change the first element to "10" in this array? `c=array([100,1,2,3,0])`

c[4]=10

c[2]=10

c[1]=10

c[0]=10



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 4

1/1 point (ungraded)

What attribute is used to return the number of dimensions in an array?

a.dtype

a.size

a.shape

a.ndim



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Graded Quiz: Working with Data in Python

[Bookmark this page](#)

Graded Quiz due Jan 27, 2022 03:21 +08

Question 1

1/1 point (graded)

Consider the following text file: **Example1.txt**:

This is line 1

This is line 2

This is line 3

What is the output of the following lines of code?

```
1 with open("Example1.txt","r") as File1:  
2  
3     file_stuff=File1.readline()  
4  
5     print(file_stuff)  
6
```



This is line 1

This is line 2

This is line 3



This is line 1



This is line 1

This is line 2



Answer

Correct: Correct.

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 2

1/1 point (graded)

Consider the file object: **File1**. How would you read the first line of text?



File1.readline ()

File1.read ()



File1.readline ()



File1.read ()



Answer

Correct: Correct.

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 3

1/1 point (graded)

What is the result of applying the following method **df.head()**, to the dataframe **df** ?

prints the first column of the dataframe

prints the first 5 rows of the dataframe

prints the first 100 row of the dataframe



Answer

Correct: correct

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 4

1/1 point (graded)

Consider the dataframe `df`. How would you access the element in the 1st row 3rd column? ?

df.iloc[2,0]

df.iloc[0,2]

df.iloc[1,3]



Answer

Correct: correct

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 5

1/1 point (graded)

What is the result of the following lines of code?

```
1 a=np.array([0,1,0,1,0])
2 b=np.array([1,0,1,0,1])
3 a*b
```

0

array([0, 0, 0, 0, 0])

array([1, 1, 1, 1, 1])



Answer

Correct: correct

Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Module 5 - APIs and Data Collection / Module Introduction and Learning Objectives

S



Module Introduction and Learning Objectives

[Bookmark this page](#)

This module delves into the unique ways to collect data by the use of APIs and webscraping. It further explores data collection by explaining how to read and collect data when dealing with different file formats.

Learning Objectives

- Define the difference between APIs and REST APIs
- Learn how APIs receive and send information
- Demonstrate how to communicate with Watson text to speech and language translator through the use of APIs
- Learn the basics of webscraping
- Work with different file formats

Simple APIs

In this video we will discuss Application Program Interfaces, or APIs for short.

Specifically, we will discuss: What an API is, API Libraries, and REST APIs, including: Request and Response, and an Example with PyCoinGecko. An API lets two pieces

of software talk to each other. For example you have your program, you have some data, and you have other software components.

You use the API to communicate with the API via inputs and outputs. Just like a function, you don't have to know how the API works, just its inputs and outputs. Pandas is actually a set of software components, much of which are not even written in Python.

You have some data. You have a set of software components. We use the pandas API to process the data by communicating with the other Software Components.

Let us clean up the diagram. When you create a dictionary, then create a pandas object with the Dataframe constructor, in API lingo, this is an "instance." The data in the dictionary is passed along to the pandas API.

You then use the dataframe to communicate with the API. When you call the method head, the dataframe communicates with the API displaying the first few rows of the dataframe. When you call the method mean the API will calculate the mean and return the values.

REST APIs are another popular type of API; they allow you to communicate through the internet letting you take advantage of resources like storage, access more data, artificial intelligence, algorithms, and much more. The RE stands for Representational, the S stands for State, the T stand for Transfer. In rest APIs your program is called the client. The API communicates with a web service you call through the internet. There is a set of rules regarding Communication, Input or Request, and Output or Response. Here are some common terms.

You or your code can be thought of as a client. The web service is referred to as a resource.

The client finds the service via an endpoint. We will review this more in the next section.

The client sends requests to the resource and the response to the client. HTTP methods are a way of transmitting data over the internet. We tell the Rest APIs what to do by sending a request. The request is usually communicated via an HTTP message. The HTTP message usually contains a JSON file.

This contains instructions for what operation we would like the service to perform. This operation is transmitted to the webservice via the internet. And the service performs the operation. In the similar manner, the webservice returns a response via an HTTP message, where the information is usually returned via a JSON file. And this information is transmitted back to the client.

Crypto Currency data is excellent to use in an API because it is constantly updated

and is vital to CryptoCurrency Trading. We will use the Py-Coin-Gecko Python Client, or Wrapper, for the Coin Gecko API, updated every minute by Coin-Gecko. We use the Wrapper, or Client, because it is easy to use so you can focus on the task of collecting data, we will also introduce pandas time series functions for dealing with time series data.

Using Py-Coin-Gecko to collect data is simple. All we need is to install and import the library the create a client object, and finally use a function to request our data. In this function we are getting data on bitcoin, in US Dollars, for the past 30 days. In this case our response is a JSON file expressed as a Python dictionary of nested lists,

including price, market cap, and total volumes, which contain the Unix timestamp and the price at that time. We are only interested in price so that is what we will select using the key price.

To make things simple, we can convert our nested list to a DataFrame. With the columns time stamp and price, it is difficult to understand the column time stamp.

We will convert it to a more readable format using the pandas function to_datetime. Using this to_datetime function, we create readable time data, the input is the time stamp column; unit of time is set to milliseconds . We append the output to the new column date.

Now we want to create a candle stick plot. To get the data for the daily candlesticks,

we will group by the date to find the minimum, maximum, first, and last price of each day. Finally we will use plotly to create the candlestick chart and plot it.

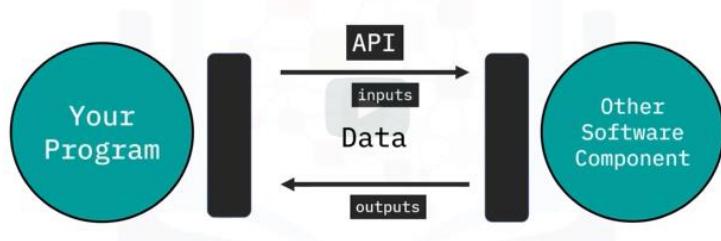
Now we can view the candlestick chart by opening the HTML file and clicking trust HTML in the top left of the tab. It should look something like this.



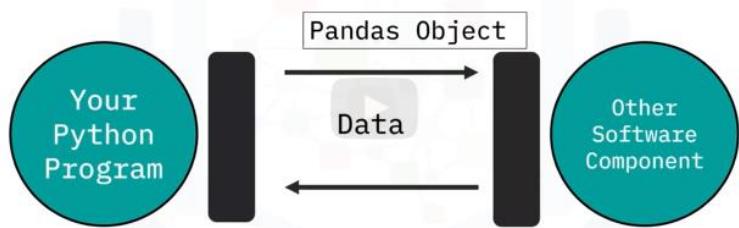
Outline

- What is an API
- API Libraries
- REST API
 - Request and Response
 - An Example with PyCoinGecko

What is an API?



What is an API?



We use the pandas API to process the data by communicating with the other Software Components.

Using an API Library

```
import pandas as pd  
dict_ = {'a':[11, 21, 31],  
        'b':[12, 22, 32]}  
df = pd.DataFrame(dict_)
```



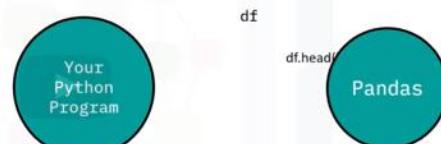
Using an API Library

```
import pandas as pd  
dict_ = {'a':[11, 21, 31],  
        'b':[12, 22, 32]}  
df = pd.DataFrame(dict_)
```



Using an API Library

```
import pandas as pd  
dict_ = {'a':[11, 21, 31],  
        'b':[12, 22, 32]}  
df = pd.DataFrame(dict_)  
  
df.head()
```



Using an API Library

```
import pandas as pd  
dict_ = {'a':[11, 21, 31],  
        'b':[12, 22, 32]}  
df = pd.DataFrame(dict_)  
  
df.head()
```



Using an API Library

```
import pandas as pd  
dict_ = {'a':[11, 21, 31],  
        'b':[12, 22, 32]}  
df = pd.DataFrame(dict_)  
  
df.head()  
  
a b  
0 11 12  
1 21 22  
2 31 32
```



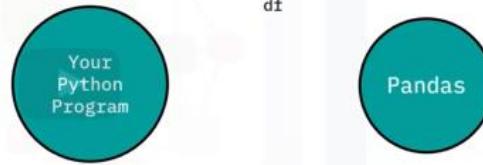
Using an API Library

```
import pandas as pd  
dict_ = {'a':[11, 21, 31],  
        'b':[12, 22, 32]}  
df = pd.DataFrame(dict_)  
  
df.head()  
  
a b  
0 11 12  
1 21 22  
2 31 32  
  
df.mean()
```



Using an API Library

```
import pandas as pd  
  
dict_ = {'a':[11, 21, 31],  
        'b':[12, 22, 32]}  
df = pd.DataFrame(dict_)  
  
df.head()  
   a  b  
0  11 12  
1  21 22  
2  31 32  
  
df.mean()  
a    21.0  
b    22.0  
dtype: float64
```

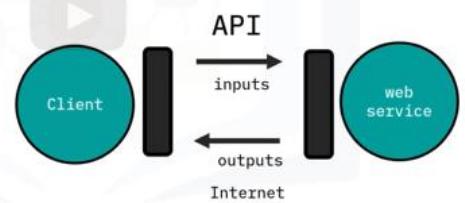


REST APIs

REpresentational State Transfer APIs

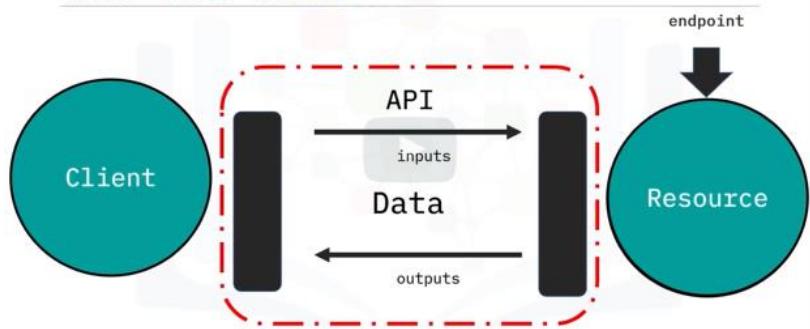
REST APIs

- REST APIs are used to interact with web services, i.e., Applications that you call through the internet
- They have a set of Rules regarding:
 1. Communication
 2. Input or Request
 3. Output or Response



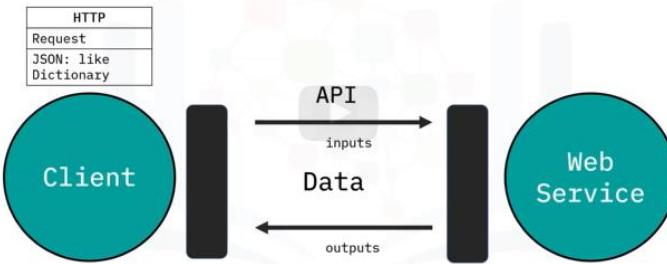
In rest APIs your program is called the **client**. The API communicates with a **web service** you call through the **internet**.

REST API terms

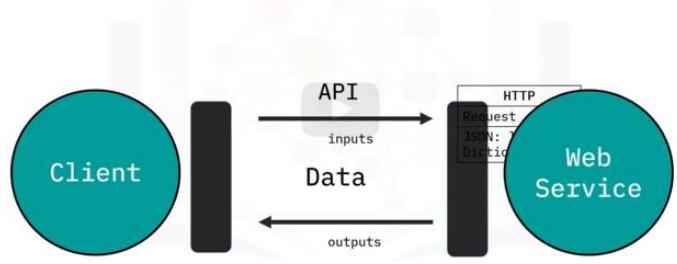


You or your code can be thought of as a **client**. The **web service** is referred to as a **resource**. The **client** finds the **service** via an **endpoint**.

REST API over HTTP



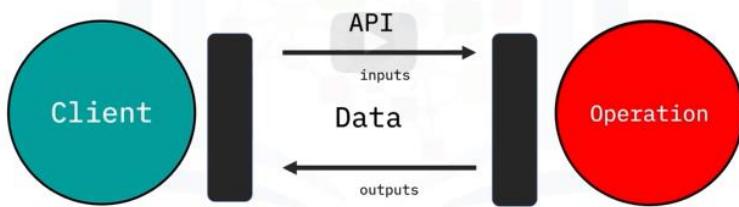
REST API over HTTP



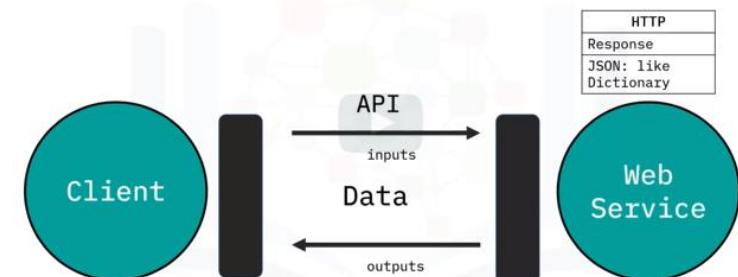
HTTP methods are a way of transmitting data over the internet.

- We tell the Rest APIs what to do by sending a request.
- The request is usually communicated via an HTTP message.
- The HTTP message usually contains a JSON file. This contains instructions for what operation we would like the service to perform. This operation is transmitted to the webservice via the internet.

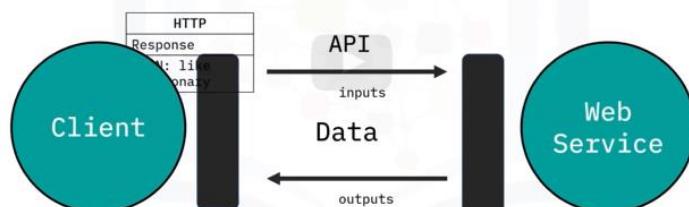
REST API over HTTP



REST API over HTTP



REST API over HTTP

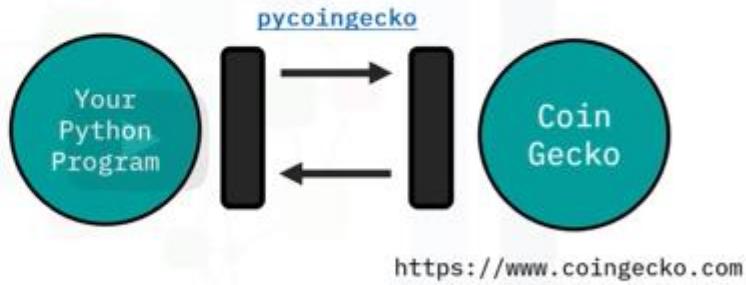


PyCoinGecko for CoinGecko API:

Python Client by Christoforou Emmanouil

```
!pip install pycoingecko
from pycoingecko import CoinGeckoAPI
cg = CoinGeckoAPI()
bitcoin_data = cg.get_coin_market_chart_by_id(id='bitcoin', vs_currency='usd', days=30)
```

```
bitcoin_data:
{'prices':
[[1604138559526,
13927.73252295053],
[1604142031697,
13797.785093120927],
[1604145789426,
13917.92602548896],
[1604150065202,
13873.651848313166],
[1604152981403,
13845.28388774076],
[1604156700430,
13806.349989300157]...]
```



Using Py-Coin-Gecko to collect data is simple. All we need is to install and import the library the create a client object, and finally use a function to request our data. In this function we are getting data on bitcoin, in US Dollars, for the past 30 days. In this case our response is a JSON file expressed as a Python dictionary of nested lists, including price, market cap, and total volumes, which contain the Unix timestamp and the price at that time. We are only interested in price so that is what we will select using the key price.

```
data = pd.DataFrame(bitcoin_price_data, columns=['TimeStamp', 'Price'])
```

| | TimeStamp | Price | Date |
|-----|---------------|--------------|-------------------------|
| 0 | 1604138559526 | 13927.732523 | 2020-10-31 10:02:39.526 |
| 1 | 1604142031697 | 13797.785093 | 2020-10-31 11:00:31.697 |
| 2 | 1604145789426 | 13917.926025 | 2020-10-31 12:03:09.426 |
| 3 | 1604150065202 | 13873.651848 | 2020-10-31 13:14:25.202 |
| 4 | 1604152981403 | 13845.283888 | 2020-10-31 14:03:01.403 |
| ... | ... | ... | ... |
| 716 | 1606716898166 | 18510.032786 | 2020-11-30 06:14:58.166 |
| 717 | 1606720068286 | 18509.742863 | 2020-11-30 07:07:48.286 |
| 718 | 1606724609861 | 18439.969378 | 2020-11-30 08:23:29.861 |
| 719 | 1606727338318 | 18453.345378 | 2020-11-30 09:08:58.318 |
| 720 | 1606728336000 | 18465.880043 | 2020-11-30 09:25:36.000 |



```
candlestick_data = data.groupby(data.Date.dt.date).agg({'Price': ['min', 'max', 'first', 'last']}))
```

| Date | Price | | | |
|------------|--------------|--------------|--------------|--------------|
| | min | max | first | last |
| 2020-10-31 | 13720.425685 | 13917.926025 | 13797.785093 | 13796.028786 |
| 2020-11-01 | 13661.927858 | 13822.470734 | 13778.637638 | 13671.267179 |
| 2020-11-02 | 13313.920358 | 13786.247252 | 13786.247252 | 13554.676070 |
| 2020-11-03 | 13359.343536 | 13847.027174 | 13558.361796 | 13847.027174 |
| 2020-11-04 | 13588.251443 | 14122.045525 | 13973.977815 | 14120.620632 |
| 2020-11-05 | 14095.428885 | 15511.856787 | 14170.702323 | 15511.856787 |
| 2020-11-06 | 15413.862067 | 15855.288073 | 15570.112495 | 15619.662967 |
| 2020-11-07 | 14630.212278 | 15679.761910 | 15583.677358 | 14926.198182 |
| 2020-11-08 | 14749.879849 | 15561.599966 | 14749.879849 | 15506.766777 |

```
fig = go.Figure(data=[go.Candlestick(x= candlestick_data.index,
                                      open=candlestick_data['Price']['first'],
                                      high=candlestick_data['Price']['max'],
                                      low=candlestick_data['Price']['min'],
                                      close=candlestick_data['Price']['last'])
                      ])

fig.update_layout(xaxis_rangeslider_visible=False, xaxis_title='Date',
                  yaxis_title='Price (USD $)', title='Bitcoin Candlestick Chart Over Past 30 Days' )

plot(fig, filename='bitcoin_candlestick_graph.html')
```





Simple APIs Part 2

We will discuss Application Program Interfaces that use some kind of artificial intelligence.

We will transcribe an audio file using the Watson Text to Speech API. We will then translate the text to a new language using the Watson Language Translator API.

In the API call, you will send a copy of the audio file to the API. This is sometimes called a POST request. Then the API will send the text transcription of what the individual is saying.

Under the hood, the API is making a GET request. We then send the text we would like to translate into a second language to a second API. The API will translate the text and send the translation back to you.

In this case, we translate English to Spanish. We then provide an overview of API keys and endpoints, Watson Speech to Text, and Watson Translate. First, we will review API keys and endpoints.

They will give you access to the API. An API key as a way to access the API. It's a unique set of characters that the API uses to identify you and authorize you. Usually, your first call to the API includes the API key.

This will allow you access to the API. In many APIs, you may get charged for each call.

So like your password, you should keep your API key a secret. An endpoint is simply the location of the service. It's used to find the API on the Internet just like a web address.

Now, we will transcribe an audio file using the Watson Text to Speech API. Before you start the lab, you should sign up for an API key. We will download an audio file into your directory.

First, we import SpeechToTextV1 from IBM Watson. The service endpoint is based on the location of the service instance. We store the information in the variable url_s2t.

To find out which URL to use, view the service credentials. You will do the same for your API key.

You create a speech-to-text adapter object. The parameters are the endpoint and API key.

You will use this object to communicate with the Watson Speech to Text service. We have the path of the wav file we would like to convert to text. We create the file object wav

with the wav file using open. We set the mode to rb, which means to read the file in binary format.

The file object allows us access to the wav file that contains the audio. We use the method recognize from the speech to text adapter object. This basically sends the audio file

to Watson Speech to Text service. The parameter audio is the file object. The content type is the audio file format. The service sends a response stored in the object response.

The attribute result contains a python dictionary. The key results value has a list that contains a dictionary. We are interested in the key transcript.

We can assign it to the variable recognized_text as follows. Recognized_text now contains a string with a transcribed text. Now let's see how to translate the text using the Watson Language Translator.

First, we import LanguageTranslatorV3 from ibm_watson. We assign the service endpoint to the variable url_12. You can obtain the service in the lab instructions. You require an API key, see the lab instructions on how to obtain the API key. This API request requires the date of the version, see the documentation. We create a language translator object, LanguageTranslator. We can get a list of the languages that the service can identify as follows.

The method returns the language code. For example, English has a symbol en to Spanish, which has the symbol es. In the last section, we assigned the transcribed texts to the variable to recognized_text. We can use the method translate. This will translate the text.

The result is a detailed response object. The parameter text is the text. model_id is the type of model we would like to use. In this case, we set it to en-es for English to Spanish.

We use the method get result to get the translated text and assign it to the variable translation. The result is a dictionary that includes the translation word count and character count.

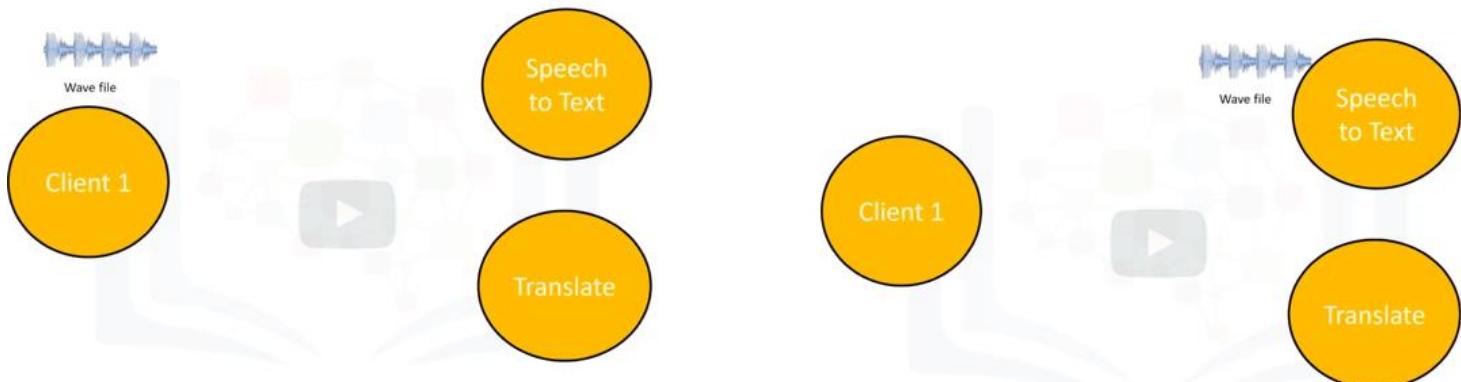
We can obtain the translation and assign it to the variable spanish_translation as follows.

Using the variable spanish_translation, we can translate the text back to English as follows.

The result is a dictionary. We can obtain the string with the text as follows. We can then translate the text to French as follows.

Python

Simple APIs (Part 2)



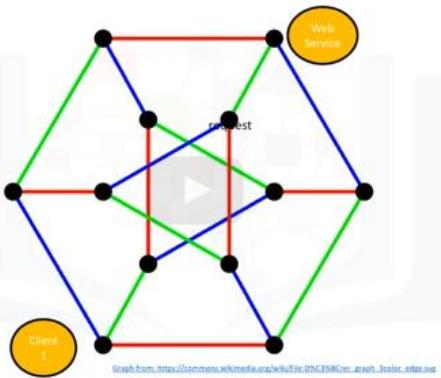
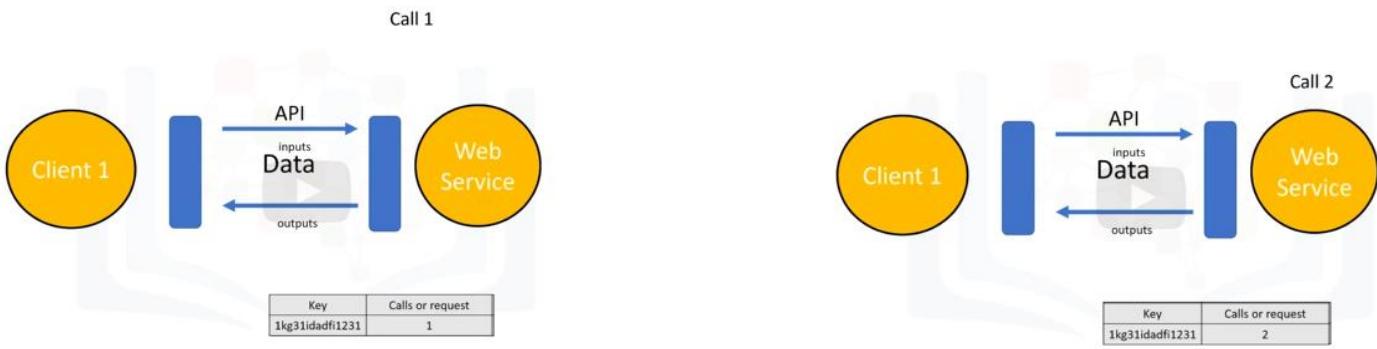


Outline

- API keys and endpoints
- Watson Speech to Text
- Watson Translate

API keys and endpoints

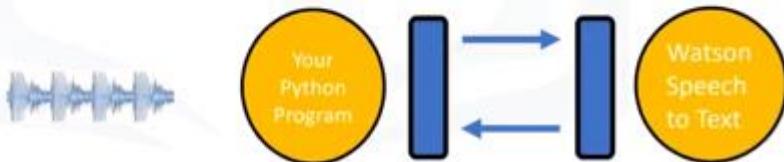




Watson Speech to Text

```
from ibm_watson import SpeechToTextV1

url_s2t="https://stream.watsonplatform.net/speech-to-text/api"
iam_apikey_s2t="EOeiZxxxDxVxxxxxxxxxxxxxxjYen9SKARKW-"
s2t = SpeechToTextV1(iam_apikey=iam_apikey_s2t, url=url_s2t)
```

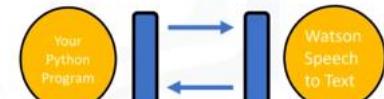


```

filename = response.result
with open(filename, mode="rb") as wav:
    recognized_text=response.result['results'][0]['alternatives'][0]['transcript']

```

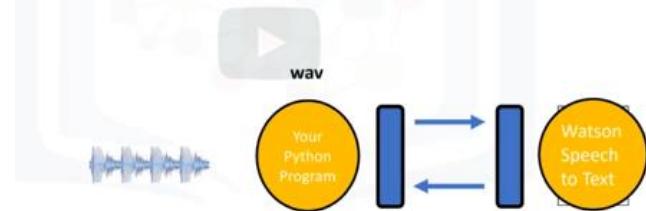
recognized_text:
'hello this is python'



```

filename='hello_this_is_python.wav'
with open(filename, mode="rb") as wav:
    response = s2t.recognize(audio=wav, content_type='audio/wav')

```

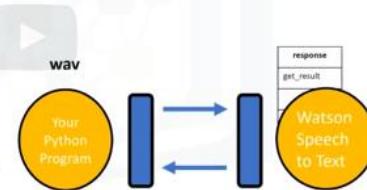


filename='hello_this_is_python.wav'

```

with open(filename, mode="rb") as wav:
    response = s2t.recognize(audio=wav, content_type='audio/wav')

```



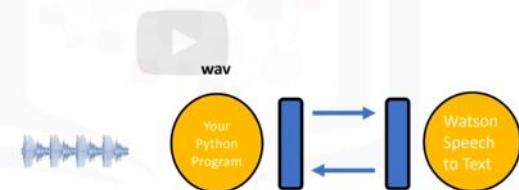
filename='hello_this_is_python.wav'

```

with open(filename, mode="rb") as wav:

```

```
    response = s2t.recognize(audio=wav, content_type='audio/wav')
```

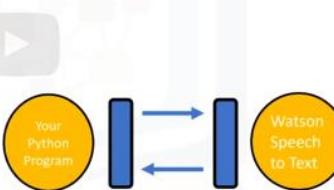


response.result

```

['results': [{'alternatives': [{'confidence': 0.91, 'transcript': 'hello this is
python '}], 'final': True}], 'result_index': 0}

```



response.result

```

['results': [{'alternatives': [{'confidence': 0.91, 'transcript': 'hello this is
python '}], 'final': True}], 'result_index': 0}

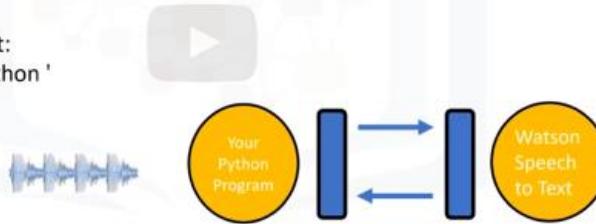
```



```
response.result
```

```
{'results': [{'alternatives': [{'confidence': 0.91, 'transcript': 'hello this is python'}], 'final': True}], 'result_index': 0}  
recognized_text=response.result['results'][0]["alternatives"][0]["transcript"]
```

recognized_text:
'hello this is python '



Watson Language Translator

```
from ibm_watson import LanguageTranslatorV3  
  
url_lt = 'https://gateway.watsonplatform.net/language-translator/api'  
apikey_lt = 'dU2SxxxxxxxxxxxxxxasdfCuasdF'  
version_lt = '2018-05-01'  
  
language_translator = LanguageTranslatorV3(iam_apikey=apikey_lt, url=url_lt, version=version_lt)  
  
language_translator.list_identifiable_languages().get_result()  
  
{'languages': [{"language": "af", "name": "Afrikaans"}, {"language": "ar", "name": "Arabic"}, {"language": "az", "name": "Azerbaijani"}, {"language": "ba", "name": "Bashkir"}, {"language": "be", "name": "Belarusian"}, {"language": "bg", "name": "Bulgarian"}, {"language": "bn", "name": "Bengali"}, {"language": "bs", "name": "Bosnian"}, {"language": "ca", "name": "Catalan"}, {"language": "cs", "name": "Czech"}, {"language": "cv", "name": "Chuvash"}, {"language": "da", "name": "Danish"}, {"language": "de", "name": "German"}, {"language": "el", "name": "Greek"}, {"language": "en", "name": "English"}, {"language": "eo", "name": "Esperanto"}, {"language": "es", "name": "Spanish"}]
```

```
from ibm_watson import LanguageTranslatorV3  
  
url_lt = 'https://gateway.watsonplatform.net/language-translator/api'  
apikey_lt = 'dU2SxxxxxxxxxxxxxxasdfCuasdF'  
version_lt = '2018-05-01'  
  
language_translator = LanguageTranslatorV3(iam_apikey=apikey_lt, url=url_lt, version=version_lt)  
  
language_translator.list_identifiable_languages().get_result()  
  
{'languages': [{"language": "af", "name": "Afrikaans"}, {"language": "ar", "name": "Arabic"}, {"language": "az", "name": "Azerbaijani"}, {"language": "ba", "name": "Bashkir"}, {"language": "be", "name": "Belarusian"}, {"language": "bg", "name": "Bulgarian"}, {"language": "bn", "name": "Bengali"}, {"language": "bs", "name": "Bosnian"}, {"language": "ca", "name": "Catalan"}, {"language": "cs", "name": "Czech"}, {"language": "cv", "name": "Chuvash"}, {"language": "da", "name": "Danish"}, {"language": "de", "name": "German"}, {"language": "el", "name": "Greek"}, {"language": "en", "name": "English"}, {"language": "eo", "name": "Esperanto"}, {"language": "es", "name": "Spanish"}]}
```

```
recognized_text = 'hello this is python'

translation_response = language_translator.translate(text=recognized_text, model_id='en-es')

translation = translation_response.get_result()

translation
{'translations': [{'translation': 'Hola, esta es la pitón.'}],
'word_count': 4, 'character_count': 21}

spanish_translation = translation['translations'][0]['translation']
spanish_translation
'Hola, esta es la pitón.'
```

translation_new = language_translator.translate(text=spanish_translation ,
model_id='es-en').get_result()

translation_eng=translation_new['translations'][0]['translation']
translation_eng
'Hey, this is the python.'

French_translation=language_translator.translate(text= translation_eng,
model_id='en-fr').get_result()
French_translation['translations'][0]['translation']
"Hé, c'est le python."



Application Programming Interface

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Create and Use APIs in Python

Introduction

An API lets two pieces of software talk to each other. Just like a function, you don't have to know how the API works only its inputs and outputs. An essential type of API is a REST API that allows you to access resources via the internet. In this lab, we will review the Pandas Library in the context of an API, we will also review a basic REST API

Table of Contents

- [Pandas is an API](#)
- [REST APIs Basics](#)
- [Quiz on Tuples](#)

```
[1]: !pip install pycoingecko
!pip install plotly
!pip install mplfinance

Collecting pycoingecko
  Downloading pycoingecko-2.2.0-py3-none-any.whl (8.3 kB)
Requirement already satisfied: requests in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from pycoingecko) (2.26.0)
Requirement already satisfied: certifi>=2017.4.17 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from requests->pycoingecko) (2021.10.8)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from requests->pycoingecko) (1.26.7)
Requirement already satisfied: idna<4,>=2.5 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from requests->pycoingecko) (3.1)
Requirement already satisfied: charset-normalizer~=2.0.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from requests->pycoingecko) (2.0.8)
Installing collected packages: pycoingecko
Successfully installed pycoingecko-2.2.0
Requirement already satisfied: plotly in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (5.4.0)
Requirement already satisfied: six in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from plotly) (1.16.0)
Requirement already satisfied: tenacity>=6.2.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from plotly) (8.0.1)
Collecting mplfinance
  Downloading mplfinance-0.12.8b6-py3-none-any.whl (64 kB)
[██████████] 64 kB 2.8 MB/s
Requirement already satisfied: matplotlib in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from mplfinance) (3.5.0)
Requirement already satisfied: pandas in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from mplfinance) (1.3.4)
Requirement already satisfied: python-dateutil>=2.7 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from matplotlib->mplfinance) (2.8.2)
Requirement already satisfied: numpy>=1.17 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from matplotlib->mplfinance) (1.21.4)
Requirement already satisfied: cycler>=0.10 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from matplotlib->mplfinance) (0.11.0)
Requirement already satisfied: packaging>=20.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from matplotlib->mplfinance) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from matplotlib->mplfinance) (3.0.6)
Requirement already satisfied: pillow>=6.2.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from matplotlib->mplfinance) (8.1.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from matplotlib->mplfinance) (1.3.2)
Requirement already satisfied: fonttools>=4.22.0 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from matplotlib->mplfinance) (4.28.2)
Requirement already satisfied: pytz>=2017.3 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from pandas->mplfinance) (2021.3)
Requirement already satisfied: six>=1.5 in /home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from python-dateutil>=2.7->matplotlib->mplfinance) (1.16.0)
Installing collected packages: mplfinance
Successfully installed mplfinance-0.12.8b6
```

Pandas is an API

Pandas is actually set of software components , much of which is not even written in Python.

```
[3]: import pandas as pd  
import numpy as np  
import plotly.graph_objects as go  
from plotly.offline import plot  
import matplotlib.pyplot as plt  
import datetime  
from pycoingecko import CoinGeckoAPI  
from mplfinance.original_flavor import candlestick2_ohlc
```

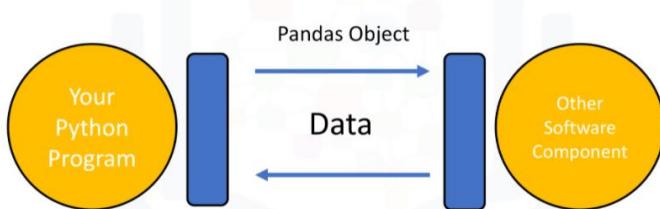
You create a dictionary, this is just data.

```
[4]: dict_={'a':[11,21,31],'b':[12,22,32]}
```

When you create a Pandas object with the Dataframe constructor in API lingo, this is an "instance". The data in the dictionary is passed along to the pandas API. You then use the dataframe to communicate with the API.

```
[5]: df=pd.DataFrame(dict_)  
type(df)
```

```
[5]: pandas.core.frame.DataFrame
```



When you call the method head the dataframe communicates with the API displaying the first few rows of the dataframe.

```
[6]: df.head()
```

```
[6]:   a   b
0  11  12
1  21  22
2  31  32
```

When you call the method mean, the API will calculate the mean and return the value.

```
[7]: df.mean()
```

```
[7]: a    21.0
      b    22.0
      dtype: float64
```

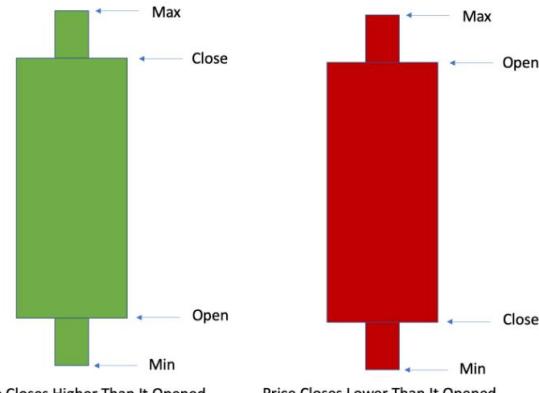
REST APIs

Rest API's function by sending a request, the request is communicated via HTTP message. The HTTP message usually contains a JSON file. This contains instructions for what operation we would like the service or resource to perform. In a similar manner, API returns a response, via an HTTP message, this response is usually contained within a JSON.

In cryptocurrency a popular method to display the movements of the price of a currency.



Here is a description of the candle sticks.



In this lab, we will be

using the CoinGecko API to create one of these candlestick graphs for Bitcoin. We will use the API to get the price data for 30 days with 24 observations per day, 1 per hour. We will find the max, min, open, and close price per day meaning we will have 30 candlesticks and use that to generate the candlestick graph. Although we are using the CoinGecko API we will use a Python client/wrapper for the API called PyCoinGecko. PyCoinGecko will make performing the requests easy and it will deal with the endpoint targeting.

Lets start off by getting the data we need. Using the `get_coin_market_chart_by_id(id, vs_currency, days)`. id is the name of the coin you want, vs_currency is the currency you want the price in, and days is how many days back from today you want.

```
[8]: cg = CoinGeckoAPI()

bitcoin_data = cg.get_coin_market_chart_by_id(id='bitcoin', vs_currency='usd', days=30)

[9]: type(bitcoin_data..)
```

```
[9]: dict
```

The response we get is in the form of a JSON which includes the price, market caps, and total volumes along with timestamps for each observation. We are focused on the prices so we will select that data.

```
[10]: bitcoin_price_data = bitcoin_data['prices']
      | bitcoin_price_data[0:5]
```

```
[10]: [[1640214412874, 49101.85946706944],
      [1640217684063, 48755.89564404483],
      [1640221229117, 48489.90371853652],
      [1640225211921, 48656.43073416629],
      [1640228823569, 48582.79701317183]]
```

Finally lets turn this data into a Pandas DataFrame.

```
[11]: data = pd.DataFrame(bitcoin_price_data, columns=['TimeStamp', 'Price'])
```

Finally lets turn this data into a Pandas DataFrame.

```
[11]: data = pd.DataFrame(bitcoin_price_data, columns=['TimeStamp', 'Price'])
```

Now that we have the DataFrame we will convert the timestamp to datetime and save it as a column called Date. We will map our unix_to_datetime to each timestamp and convert it to a readable datetime.

```
[12]: data['date'] = data['TimeStamp'].apply(lambda d: datetime.date.fromtimestamp(d/1000.0))
```

Using this modified dataset we can now group by the Date and find the min, max, open, and close for the candlesticks.

```
[13]: candlestick_data = data.groupby(data.date, as_index=False).agg({"Price": ['min', 'max', 'first', 'last']})
```

Finally we are now ready to use plotly to create our Candlestick Chart.

```
[14]: fig = go.Figure(data=[go.Candlestick(x=candlestick_data['date'],
                                          open=candlestick_data['Price']['first'],
                                          high=candlestick_data['Price']['max'],
                                          low=candlestick_data['Price']['min'],
                                          close=candlestick_data['Price']['last'])])
fig.update_layout(xaxis_rangeslider_visible=False)
fig.show()
```



Watson Speech to Text Translator

Estimated time needed: 25 minutes

Objectives

After completing this lab you will be able to:

- Create Speech to Text Translator

Introduction

In this notebook, you will learn to convert an audio file of an English speaker to text using a Speech to Text API. Then you will translate the English version to a Spanish version using a Language Translator API. **Note:** You must obtain the API keys and endpoints to complete the lab.

Table of Contents

- [Speech To Text](#)
- [Language Translator](#)
- [Exercise](#)

In [1]:

```
#you will need the following library
!pip install ibm_watson wget
```

Collecting ibm_watson

 Downloading

 https://files.pythonhosted.org/packages/9b/88/395d7d52df29f321ae1150cf9b5a71cef8611570230502597c427bc1e9d9/ibm-watson-5.1.0.tar.gz (382kB)

 |██████████| 389kB 16.3MB/s eta 0:00:01

Requirement already satisfied: wget in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (3.2)

Requirement already satisfied: requests<3.0,>=2.0 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from ibm_watson) (2.25.1)

Requirement already satisfied: python_dateutil>=2.5.3 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from ibm_watson) (2.8.1)

Collecting websocket-client==0.48.0 (from ibm_watson)

```

  Downloading
https://files.pythonhosted.org/packages/8a/a1/72ef9aa26cfe1a75cee09fc1957e4723add9de098c15719416a1ee89386b/websoc
ket_client-0.48.0-py2.py3-none-any.whl (198kB)

|██████████| 204kB 42.2MB/s eta 0:00:01

Collecting ibm_cloud_sdk_core>=3.3.6 (from ibm_watson)

  Downloading
https://files.pythonhosted.org/packages/99/55/ece4d000ca41c052c7331a0d14150e9c9c15e4f65943036cff3bcd14cc7/ibm-
cloud-sdk-core-3.10.0.tar.gz

Requirement already satisfied: idna<3,>=2.5 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from
requests<3.0,>=2.0->ibm_watson) (2.10)

Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from
requests<3.0,>=2.0->ibm_watson) (1.26.4)

Requirement already satisfied: certifi>=2017.4.17 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from
requests<3.0,>=2.0->ibm_watson) (2020.12.5)

Requirement already satisfied: chardet<5,>=3.0.2 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from
requests<3.0,>=2.0->ibm_watson) (4.0.0)

Requirement already satisfied: six>=1.5 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from
python_dateutil>=2.5.3->ibm_watson) (1.16.0)

Collecting PyJWT<3.0.0,>=2.0.1 (from ibm_cloud_sdk_core>=3.3.6->ibm_watson)

  Downloading
https://files.pythonhosted.org/packages/3f/32/d5d3cab27fee7f6b22d7cd7507547ae45d52e26030fa77d1f83d0526c6e5/PyJWT-
2.1.0-py3-none-any.whl

Building wheels for collected packages: ibm-watson, ibm-cloud-sdk-core
  Building wheel for ibm-watson (setup.py) ... done
    Stored in directory:
/home/jupyterlab/.cache/pip/wheels/49/6d/cf/1d91261b96363da78bf9b02699fd2262e6b5dad179500690c1
  Building wheel for ibm-cloud-sdk-core (setup.py) ... done
    Stored in directory:
/home/jupyterlab/.cache/pip/wheels/4a/4e/48/b02ad6dc75235fc4c0742d4e99571fe7d729e60bf365105be4
  Successfully built ibm-watson ibm-cloud-sdk-core
  Installing collected packages: websocket-client, PyJWT, ibm-cloud-sdk-core, ibm-watson
  Successfully installed PyJWT-2.1.0 ibm-cloud-sdk-core-3.10.0 ibm-watson-5.1.0 websocket-client-0.48.0

```

Speech to Text

First we import SpeechToTextV1 from ibm_watson. For more information on the API, please click on this [link](#)

```
In [2]: from ibm_watson import SpeechToTextV1
import json
from ibm_cloud_sdk_core.authenticators import IAMAuthenticator
```

The service endpoint is based on the location of the service instance, we store the information in the variable URL. To find out which URL to use, view the service credentials and paste the url here.

```
In [3]: url_s2t = "https://api.us-south.speech-to-text.watson.cloud.ibm.com/instances/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

You require an API key, and you can obtain the key on the Dashboard .

```
In [4]: iam_apikey_s2t = "xx-xxxxxxxxxxxxx-xxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

You create a [Speech To Text Adapter object](#) the parameters are the endpoint and API key.

```
In [5]: authenticator = IAMAuthenticator(iam_apikey_s2t)
s2t = SpeechToTextV1(authenticator=authenticator)
s2t.set_service_url(url_s2t)
s2t
```

```
Out[5]: <ibm_watson.speech_to_text_v1_adapter.SpeechToTextV1Adapter at 0x7f931c16b0f0>
```

Lets download the audio file that we will use to convert into text.

```
In [6]: !wget -O PolynomialRegressionandPipelines.mp3 https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDriverSkillsNetwork-PY0101EN-SK
```

```
--2021-05-18 21:37:41-- https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDriverSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/PolynomialRegressionandPipelines.mp3
```

```
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)... 169.63.118.104
```

```
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)|169.63.118.104|:443... connected.
```

```
HTTP request sent, awaiting response... 200 OK
```

```
Length: 4234179 (4.0M) [audio/mpeg]
```

```
Saving to: 'PolynomialRegressionandPipelines.mp3'
```

```
PolynomialRegression 100%[=====] 4.04M --.-KB/s in 0.05s
```

```
2021-05-18 21:37:41 (84.3 MB/s) - 'PolynomialRegressionandPipelines.mp3' saved [4234179/4234179]
```

We have the path of the wav file we would like to convert to text

```
In [7]: filename='PolynomialRegressionandPipelines.mp3'
```

We create the file object wav with the wav file using open ; we set the mode to "rb" , this is similar to read mode, but it ensures the file is in binary mode.We use the method recognize to return the recognized text. The parameter audio is the file object wav, the parameter content_type is the format of the audio file.

```
In [9]: with open(filename, mode="rb") as wav:  
    response = s2t.recognize(audio=wav, content_type='audio/mp3')
```

The attribute result contains a dictionary that includes the translation:

```
In [10]: response.result
```

```
Out[10]: {'result_index': 0,  
          'results': [{'final': True,  
                      'alternatives': [{'transcript': 'in this video we will cover polynomial regression and pipelines ',  
                                      'confidence': 0.94}]}],  
          {'final': True,  
           'alternatives': [{'transcript': "what do we do when a linear model is not the best fit for our data let's look into another type of regression model the polynomial regression we transform our data into a polynomial then use linear regression to fit the parameters that we will discuss pipelines pipelines are way to simplify your code ",  
                           'confidence': 0.9}]}],  
          {'final': True,  
           'alternatives': [{'transcript': "polynomial regression is a special case of the general linear regression this method is beneficial for describing curvilinear relationships what is a curvilinear relationship it's what you get by squaring or setting higher order terms of the predictor variables in the model transforming the data the model can be quadratic which means the predictor variable in the model is squared we use a bracket to indicate as an exponent this is the second order polynomial regression with a figure representing the function ",  
                           'confidence': 0.95}]}],  
          {'final': True,  
           'alternatives': [{'transcript': 'the model can be cubic which means the predictor variable is cube this is the third order polynomial regression we see by examining the figure that the function has more variation ',  
                           'confidence': 0.95}]}],  
          {'final': True,  
           'alternatives': [{'transcript': "there also exists higher order polynomial regressions when a good fit hasn't been achieved by second or third order we can see in figures how much the graphs change when we change the order of the polynomial regression the degree of the regression makes a big difference and can result in a better fit if you pick the right value in all cases the relationship between the variable in the parameter is always linear ",  
                           'confidence': 0.91}]}],  
          {'final': True,  
           'alternatives': [{'transcript': "let's look at an example from our data we generate a polynomial regression model ",  
                           'confidence': 0.89}]}],  
          {'final': True,  
           'alternatives': [{'transcript': 'in python we do this by using the poly fit function in this example we develop a third order polynomial regression model base we can print out the model symbolic form for the model is given by the following expression ',  
                           'confidence': 0.92}]}],  
          {'final': True,  
           'alternatives': [{'transcript': "negative one point five five seven X. one cube plus two hundred four point eight X. one squared plus eight thousand nine hundred sixty five X. one plus one point three seven times ten to the power of five we can also have multi dimensional polynomial linear regression the expression can get complicated here are just some of the terms for two dimensional second order polynomial none pies poly fit function cannot perform this type of regression we use the preprocessing librarian scikit learn to create a polynomial feature object the constructor takes the degree of the polynomial as a parameter then we transform the features into a polynomial feature with the fit underscore transform method let's do a more intuitive example ",  
                           'confidence': 0.9}]}],  
          {'final': True,  
           'alternatives': [{'transcript': 'consider the feature shown here applying the method we transform the data we now have a new set of features that are transformed version of our original features as that I mention of the data gets larger we may want to normalize multiple features as scikit learn instead we can use the preprocessing module to simplify many tasks for example we can standardize each feature simultaneously we import standard scaler we train in the object fit the scale object then transform the data into a new data frame on a rate X. underscore scale there are more normalization methods available in the pre processing library as well as other transformations we can simplify our code by using a pipeline library there are many steps to get a prediction for example normalization polynomial transform and linear regression we simplify the process using a pipeline ',  
                           'confidence': 0.9}]}],  
          {'final': True,  
           'alternatives': [{'transcript': 'pipeline sequentially perform a series of transformations the last step carries out a prediction first we import all the modules we need then we import the library pipeline we create a list of tuples the first element in the tuple contains the name of the estimator model the second element contains model constructor we input the list in the pipeline constructor we now have a pipeline object we can train the pipeline by applying the train method to the pipeline object we can also produce a prediction as well ',  
                           'confidence': 0.89}]}],  
          {'final': True,
```

```

'alternatives': [{('transcript'): 'consider the feature shown here applying the method we transform the data we now have a new set of features that are transformed version of our original features as that I mention of the data gets larger we may want to normalize multiple features as scikit learn instead we can use the preprocessing module to simplify many tasks for example we can standardize each feature simultaneously we import standard scaler we train in the object fit the scale object then transform the data into a new data frame on a rate X. underscore scale there are more normalization methods available in the pre processing library as well as other transformations we can simplify our code by using a pipeline library there are many steps to getting a prediction for example normalization polynomial transform and linear regression we simplify the process using a pipeline ',
'confidence': 0.9}]}},
{'final': True,
'alternatives': [{('transcript'): 'pipeline sequentially perform a series of transformations the last step carries out a prediction first we import all the modules we need then we import the library pipeline we create a list of topologies the first element in the topology contains the name of the estimator model the second element contains model constructor we input the list in the pipeline constructor we now have a pipeline object we can train the pipeline by applying the train method to the pipeline object we can also produce a prediction as well ',
'confidence': 0.89}]}},
{'final': True,
'alternatives': [{('transcript'): 'the method normalizes the data performs a polynomial transform then outputs a prediction ',
'confidence': 0.89}]}]}

```

In [14]:

```
from pandas import json_normalize
json_normalize(response.result['results'],"alternatives")
```

Out[14]:

| | transcript | confidence |
|----|--|------------|
| 0 | in this video we will cover polynomial regress... | 0.94 |
| 1 | what do we do when a linear model is not the b... | 0.90 |
| 2 | polynomial regression is a special case of the... | 0.95 |
| 3 | the model can be cubic which means the predict... | 0.95 |
| 4 | there also exists higher order polynomial regre... | 0.91 |
| 5 | let's look at an example from our data we gene... | 0.89 |
| 6 | in python we do this by using the poly fit fun... | 0.92 |
| 7 | negative one point five five seven X. one cute... | 0.90 |
| 8 | consider the feature shown here applying the m... | 0.90 |
| 9 | pipeline sequentially perform a series of tran... | 0.89 |
| 10 | the method normalizes the data performs a poly... | 0.89 |

In [12]:

```
response
```

Out[12]:

```
<ibm_cloud_sdk_core.detailed_response.DetailedResponse at 0x7f931c12ff28>
```

We can obtain the recognized text and assign it to the variable `recognized_text`:

In [17]:

```
recognized_text=response.result['results'][0]["alternatives"][0]["transcript"]
type(recognized_text)
```

Out[17]:

```
str
```

Language Translator

First we import LanguageTranslatorV3 from ibm_watson. For more information on the API click [here](#)

```
In [24]: from ibm_watson import LanguageTranslatorV3
```

The service endpoint is based on the location of the service instance, we store the information in the variable URL. To find out which URL to use, view the service credentials.

```
In [25]: url_lt='https://api.us-south.language-translator.watson.cloud.ibm.com/instances/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
```

You require an API key, and you can obtain the key on the [Dashboard](#).

```
In [26]: apikey_lt='xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx-xxx'
```

API requests require a version parameter that takes a date in the format version=YYYY-MM-DD. This lab describes the current version of Language Translator, 2018-05-01

```
In [27]: version_lt='2018-05-01'
```

we create a Language Translator object language_translator :

```
In [45]: authenticator = IAMAuthenticator(apikey_lt)
language_translator = LanguageTranslatorV3(version=version_lt,authenticator=authenticator)
language_translator.set_service_url(url_lt)
language_translator
```

```
Out[45]: <ibm_watson.language_translator_v3.LanguageTranslatorV3 at 0x7f931ae2e7f0>
```

We can get a Lists the languages that the service can identify. The method Returns the language code. For example English (en) to Spanish (es) and name of each language.

```
In [29]: from pandas import json_normalize
json_normalize(language_translator.list_identifiable_languages().get_result(), "languages")
```

Out[29]:

| | language | name |
|-----|----------|---------------------|
| 0 | af | Afrikaans |
| 1 | ar | Arabic |
| 2 | az | Azerbaijani |
| 3 | ba | Bashkir |
| 4 | be | Belarusian |
| ... | ... | ... |
| 71 | uk | Ukrainian |
| 72 | ur | Urdu |
| 73 | vi | Vietnamese |
| 74 | zh | Simplified Chinese |
| 75 | zh-TW | Traditional Chinese |

76 rows × 2 columns

In [47]:

```
language_translator.list_idifiable_languages().get_result()
```

```
Out[47]: {'languages': [{'language': 'af', 'name': 'Afrikaans'},  
{'language': 'ar', 'name': 'Arabic'},  
{'language': 'az', 'name': 'Azerbaijani'},  
{'language': 'ba', 'name': 'Bashkir'},  
{'language': 'be', 'name': 'Belarusian'},  
{'language': 'bg', 'name': 'Bulgarian'},  
{'language': 'bn', 'name': 'Bengali'},  
{'language': 'ca', 'name': 'Catalan'},  
{'language': 'cs', 'name': 'Czech'},  
{'language': 'cv', 'name': 'Chuvash'},  
{'language': 'cy', 'name': 'Welsh'},  
{'language': 'da', 'name': 'Danish'},  
{'language': 'de', 'name': 'German'},  
{'language': 'el', 'name': 'Greek'},  
{'language': 'en', 'name': 'English'},  
{'language': 'eo', 'name': 'Esperanto'},  
{'language': 'es', 'name': 'Spanish'},  
{'language': 'et', 'name': 'Estonian'},  
{'language': 'eu', 'name': 'Basque'},  
{'language': 'fa', 'name': 'Persian'},  
{'language': 'fi', 'name': 'Finnish'},  
{'language': 'fr', 'name': 'French'},  
{'language': 'ga', 'name': 'Irish'},  
{'language': 'gu', 'name': 'Gujarati'},  
{'language': 'he', 'name': 'Hebrew'},  
{'language': 'hi', 'name': 'Hindi'},  
{'language': 'hr', 'name': 'Croatian'},  
{'language': 'ht', 'name': 'Haitian'},  
{'language': 'hu', 'name': 'Hungarian'},  
{'language': 'hy', 'name': 'Armenian'},  
{'language': 'is', 'name': 'Icelandic'},  
{'language': 'it', 'name': 'Italian'},  
{'language': 'ja', 'name': 'Japanese'},  
{'language': 'ka', 'name': 'Georgian'},  
{'language': 'kk', 'name': 'Kazakh'},  
{'language': 'km', 'name': 'Central Khmer'},  
{'language': 'ko', 'name': 'Korean'},  
{'language': 'ku', 'name': 'Kurdish'},  
{'language': 'ky', 'name': 'Kirghiz'},  
{'language': 'lo', 'name': 'Lao'},  
{'language': 'lt', 'name': 'Lithuanian'},  
{'language': 'lv', 'name': 'Latvian'},  
{'language': 'ml', 'name': 'Malayalam'},  
{'language': 'mn', 'name': 'Mongolian'},  
{'language': 'mr', 'name': 'Marathi'},  
{'language': 'ms', 'name': 'Malay'},  
{'language': 'mt', 'name': 'Maltese'},  
{'language': 'my', 'name': 'Burmese'},  
{'language': 'nb', 'name': 'Norwegian Bokmal'},  
{'language': 'ne', 'name': 'Nepali'},  
{'language': 'nl', 'name': 'Dutch'},  
{'language': 'nn', 'name': 'Norwegian Nynorsk'},  
{'language': 'pa', 'name': 'Punjabi'},  
{'language': 'pa-PK', 'name': 'Punjabi (Shahmukhi script, Pakistan)'},  
{'language': 'pl', 'name': 'Polish'}]
```

```

{'language': 'pl', 'name': 'Polish'},
{'language': 'ps', 'name': 'Pushto'},
{'language': 'pt', 'name': 'Portuguese'},
{'language': 'ro', 'name': 'Romanian'},
{'language': 'ru', 'name': 'Russian'},
{'language': 'si', 'name': 'Sinhala'},
{'language': 'sk', 'name': 'Slovakian'},
{'language': 'sl', 'name': 'Slovenian'},
{'language': 'so', 'name': 'Somali'},
{'language': 'sq', 'name': 'Albanian'},
{'language': 'sr', 'name': 'Serbian'},
{'language': 'sv', 'name': 'Swedish'},
{'language': 'ta', 'name': 'Tamil'},
{'language': 'te', 'name': 'Telugu'},
{'language': 'th', 'name': 'Thai'},
{'language': 'tl', 'name': 'Tagalog'},
{'language': 'tr', 'name': 'Turkish'},
{'language': 'uk', 'name': 'Ukrainian'},
{'language': 'ur', 'name': 'Urdu'},
{'language': 'vi', 'name': 'Vietnamese'},
{'language': 'zh', 'name': 'Simplified Chinese'},
{'language': 'zh-TW', 'name': 'Traditional Chinese'}]}

```

We can use the method translate this will translate the text. The parameter text is the text. Model_id is the type of model we would like to use we use list the language . In this case, we set it to 'en-es' or English to Spanish. We get a Detailed Response object translation_response.

```
In [30]: translation_response = language_translator.translate(\n    text=recognized_text, model_id='en-es')\ntranslation_response
```

```
Out[30]: <ibm_cloud_sdk_core.detailed_response.DetailedResponse at 0x7f931ae920f0>
```

The result is a dictionary.

```
In [31]: translation=translation_response.get_result()\ntranslation
```

```
Out[31]: {'translations': [{'translation': 'en este video cubriremos la regresión polinómica y las tuberías'}],\n          'word_count': 10,\n          'character_count': 64}
```

We can obtain the actual translation as a string as follows:

We can obtain the actual translation as a string as follows:

```
In [32]: spanish_translation = translation['translations'][0]['translation']
spanish_translation
```

```
Out[32]: 'en este video cubriremos la regresión polinómica y las tuberías '
```

We can translate back to English

```
In [33]: translation_new = language_translator.translate(text=spanish_translation ,model_id='es-en').get_result()
```

We can obtain the actual translation as a string as follows:

```
In [34]: translation_eng=translation_new['translations'][0]['translation']
translation_eng
```

```
Out[34]: 'in this video we will cover the polynomial regression and the pipes '
```

Quiz

Translate to French.

```
In [52]: # Write your code below and press Shift+Enter to execute
translation_french = language_translator.translate(text=translation_eng ,
                                                     model_id='en-fr').get_result()

translation_fr = translation_french['translations'][0]['translation']
translation_fr
```



```
Out[52]: 'Dans cette vidéo nous couvrons la régression polynomiale et les tuyaux '
```

Click here for the solution ````python French_translation=language_translator.translate(text=translation_eng , model_id='en-fr').get_result()
French_translation['translations'][0]['translation']````
Translate to Korean from English

```
In [56]: translation_korean = language_translator.translate(text=translation_eng ,  
                                                       model_id='en-ko').get_result()  
translation_ko = translation_korean['translations'][0]['translation']  
translation_ko
```

```
Out[56]: '이 비디오에서 우리는 다항 회귀 및 파이프를 다룰 것이다. '
```

Translate to English from Korean

```
In [59]: translation_ko_eng = language_translator.translate(text=translation_ko ,  
                                                       model_id='ko-en').get_result()  
translation_ko_eng['translations'][0]['translation']
```

```
Out[59]: 'In this video, we will deal with polynomial regression and pipes. '
```

Language Translator

References

<https://cloud.ibm.com/apidocs/speech-to-text?code=python>

<https://cloud.ibm.com/apidocs/language-translator?code=python>

Practice Quiz: Simple APIs

 Bookmarked

Question 1

1/1 point (ungraded)

What does API stand for?

- Application Process Interface
- Automatic Program Interaction
- Application Programming Interface
- Application Programming Interaction



Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 2

1/1 point (ungraded)

The chart used to plot the cryptocurrency data in the lab is a...

- Point and Figure Chart
- Candlestick Chart
- Pole Chart
- Line Chart



Submit

You have used 1 of 2 attempts

Correct (1/1 point)

Question 3

1/1 point (ungraded)

What information are we trying to find for each day in the lab for the chart?

Open (Last), High (Max), Low (Min), Close (First)

Open (Max), High (First), Low (Last), Close (Min)

Open (Min), High (First), Low (Max), Close (Last)

Open (First), High (Max), Low (Min), Close (Last)



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

REST APIs & HTTP Requests - Part 1

In this video, we will discuss the HTTP protocol. Specifically, we will discuss: Uniform Resource Locator: URL Request Response

We touched on REST APIs in the last section. The HTTP protocol can be thought of as a general protocol of transferring information through the web. This includes many types of REST APIs. Recall that REST API's function by sending a request, and the request is communicated via HTTP message.

The HTTP message usually contains a JSON file. When you, the client, use a web page your browser sends an HTTP request to the server where the page is hosted. The server tries to find the desired resource by default "index.html". If your request is successful,

the server will send the object to the client in an HTTP response; this includes information like the type of the resource, the length of the resource, and other information. The table under the Web server represents a list of resources stored in the web server.

In this case, an HTML file, png image, and txt file . When the request is made for the information, the web servers sends the requested information, i.e., one of the files.

Uniform Resource Locator: URL

Uniform resource locator (URL) is the most popular way to find resources on the web.

We can break the URL into three parts.

the scheme this is the protocol, for this lab it will always be http://

Internet address or Base URL this will be used to find the location; some examples include www.ibm.com and www.gitlab.com

route: this is the location on the web server; for example: /images/IDSNLogo.png

Let's review the request and Response process. The following is an example of the request message for the get request method. There are other HTTP methods we can use. In the start line we have the GET method. This is an HTTP method. In this case, it's requesting the file index.html

The Request header passes additional information with an HTTP request. In the GET method the Request header is empty.

Some Requests have a body; we will have an example of a request body later.

The following table represents the response. The response start line contains the version number followed by a descriptive phrase.

In this case, HTTP/1.0 a status code (200) meaning success, and the descriptive phrase, OK.

We have more on status codes later. The response header contains information.

Finally, we have the response body containing the requested file, in this case an HTML document.

Lets look at other status codes. Some status code examples are shown in the table below. The prefix indicates the class; for example, the 100s are informational responses; 100 indicates that everything is OK so far.

The 200s are Successful responses: For example, 200 The request has succeeded.

Anything in the 400s is bad news. 401 means the request is unauthorized. 500's stands for server errors, like 501 for not Implemented. When an HTTP request is made, an HTTP method is sent.

This tells the server what action to perform. A list of several HTTP methods is shown here.

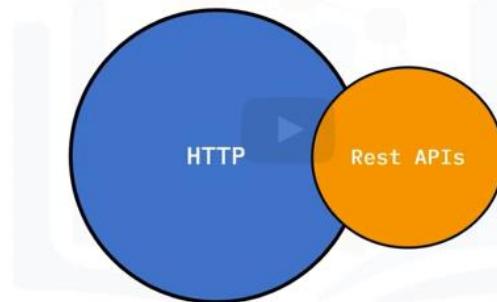
In the next video, we will use Python to apply the GET method that Retrieves data from the server and the post method that sends data to the server.

HTTP Protocol

Outline

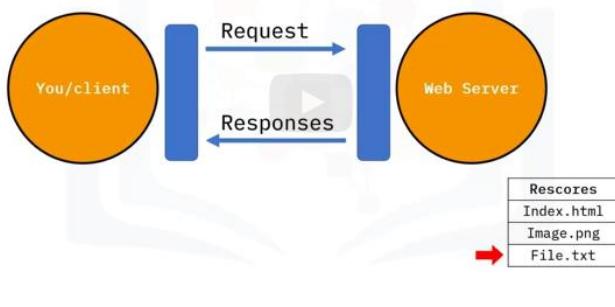
- Uniform Resource Locator: URL
- Request
- Response

HTTP Protocol

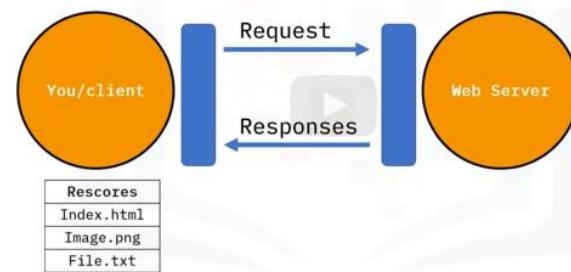


The HTTP protocol can be thought of as a general protocol of transferring information through the web.

HTTP Request



HTTP Request



Uniform Resource Locator: URL

Overview of HTTP

- **Scheme:** the protocol, for this lab it will always be **http://**
- **Internet address or Base URL:** used to find the location, for example: **www.ibm.com** and **www.gitlab.com**
- **Route:** location on the web server, for example: /images/IDSNlogo.png

http://www.ibm.com/images/IDSNlogo.png

Request and Response

Request Message

| | |
|--------------------|---|
| Request Start line | Get/index.html HTTP/1.0 |
| Request Header | User-Agent: python-requests/2.21.0 Accept-Encoding: gzip, deflate : |
| Request Body | |

Response Message

| | |
|---------------------|--|
| Response Start line | HTTP/1.0 200 OK |
| Response Header | Server: Apache-Cache: UNCACHEABLE |
| Response Body | <!DOCTYPE html><html><body><h1>My First Heading</h1><p>My first paragraph.</p></body></html> |

Status Code

| | |
|-----|-------------------------|
| 1XX | Informational |
| 100 | Everything So Far Is OK |
| 2xx | Success |
| 200 | OK |
| 3XX | Redirection |
| 300 | Multiple Choices |

| | |
|-----|----------------|
| 4XX | Client Error |
| 401 | Unauthorized |
| 403 | Forbidden |
| 404 | Not Found |
| 500 | Server Error |
| 501 | No Implemented |

HTTP Methods

| HTTP METHODS | Description |
|--------------|--------------------------------|
| GET | Retrieves data from the server |
| POST | Submits data to server |
| PUT | Updates data already on server |
| DELETE | Deletes data from server |

REST APIs & HTTP Requests - Part 2

In this video, we will discuss the HTTP protocol using the Requests Library a popular method for dealing with the HTTP protocol in Python

In this video, we will review Python library requests for Working with the HTTP protocols.

We will provide an overview of [Get Requests](#) and [Post Requests](#)

Let's review the Request Module in Python. This is one of several libraries including: `httplib`, `urllib`, that can work with the HTTP protocol.

Requests is a python Library that allows you to send HTTP/1.1 requests easily. We can import the library as follows:

You can make a GET request via the method `get` to `www.ibm.com`:

We have the response object '`r`', this has information about the request, like

the status of the request. We can view the status code using the attribute `status_code`, which is 200 for OK. You can view the request headers:

You can view the request body in the following line. As there is no body for a GET request, we get a `None`. You can view the HTTP response header using the attribute `headers`. This returns a python dictionary of HTTP response headers. We can look at the dictionary values.

We can obtain the date the request was sent by using the key `Date`. The key `Content-Type` indicates the type of data.

Using the response object '`r`' , we can also check the encoding: As the `Content-Type` is `text/html`, we can use the attribute `text` to display the HTML in the body. We can review the first 100 characters. You can also download other content, see the lab for more.

You can use the `GET` method to modify the results of your query. For example, retrieving data from an API. In the lab we will use `httpbin.org` A simple HTTP Request & Response Service.

We send a `GET` request to the server. Like before, we have the Base URL in the Route; we append `/get`. This indicates we would like to perform a `GET` request. This is demonstrated in the following table: After `GET` is requested we have the query string. This is a part of a uniform resource locator (URL) and this sends other information to the web server.

The start of the query is a `?`, followed by a series of parameter and value pairs, as shown in the table below. The first parameter name is "name" and the value is "Joseph."

The second parameter name is "ID" and the Value is "123." Each pair, parameter, and value is separated by an equal sign, "`=`." The series of pairs is separated by the ampersand, "`&`."

Let's complete an example in python. We have the Base URL with `GET` appended to the end.

To create a Query string, we use the dictionary payload. The keys are the parameter names, and the values are the value of the Query string.

Then, we pass the dictionary payload to the params parameter of the get() function.

We can print out the URL and see the name and values. We can see the request body. As the info is sent in the URL, the body has a value of None. We can print out the status code.

We can view the response as text: We can look at the key 'Content-Type' to look at the content type.

As the content 'Content-Type' is in the JSON, we format it using the method json(). It

returns a Python dict: The key 'args' has the name and values for the query string.

Like a GET request a POST request is used to send data to a server, but the POST request

sends the data in a request body, not the url. In order to send the Post Request in the URL, we change the route to POST: This endpoint will expect data and it is a convenient way to configure an HTTP request to send data to a server.

We have The Payload dictionary.

To make a POST request, we use the post() function. The variable payload is passed to the parameter data :

Comparing the URL using the attribute url from the response object of the GET and POST request, we see the POST request has no name or value pairs in its url.

We can compare the POST and GET request body. We see only the POST request has a body:

We can view the key form to get the payload.

HTTP Using Requests Library in Python

Outline

- Requests in Python
- Get Request
- Post Requests

Request Module in Python

Requests

```
import requests
url='https://www.ibm.com/'
r=requests.get(url)
r.status_code:200
r.request.headers
{'User-Agent': 'python-requests/2.24.0', 'Accept-Encoding': 'gzip, deflate',
'Accept': '*/*', 'Connection': 'keep-alive', 'Cookie':
'_abck=A5C90067E0241F8BB03ECC70ECD81EC0~~
1-YAAQLc2U0ZalikB1AQAAwucY4QT77mnX/GJQJvRG0V48PDVR70euYZ9FTBsFbF3z4kwEcKgttV+Ot6Sz
P+sjsjPDIFni428WFQn/yUVVddQ2e1ejPilhGVE+eNj14xbIRXxE1U59jQLVi0jd/Q8e6C1E7mw27Qzb5nC
gdMc4ZnULKeu6U5QSVLFRUPnhMlOM40+iWFW1CNH36WrccSUtoKkTsExTltvbsvxIbthIn3znl1racK
tv9WJF1AEdqajkt0qX/frGBULGLK/r47j78DuZHabdMhdalss911MpFxj6kXCFe~-1~1~1;
bm_sz=564D34683F98DB203E92B73A05A3324C~YAAQLc2U0ZwlkB1AQAAwucY4Qn7jwZFH9N07QY5q5S
JvdADHcL95RCSPsAGLlx2RAr+iIH6ht/frpWT8RXkjWBccT529lu91pIEz6zjed+o0yyTH7aL/qkC9nzV
mdZfvAhOjFKnFzewR7xguYLBc7X1AAeG8GrkPFHq0vMT038GwTMCC+xGYQyPm6y'}
```

Requests

```
r.request.body:None
header=r.headers
header:
{'Server': 'Apache', 'x-drupal-dynamic-cache': 'UNCACHEABLE', 'Link':
 '<https://www.ibm.com/ca-en>; rel="canonical", <https://www.ibm.com/ca-en>; rel="revision",
 <https://www.ibm.com/ca-en>; rel="revision", </1.cms.s8ic.com>; rel="preconnect; crossorigin,
 </1.cms.s8ic.com>; rel=dns-prefetch", "x-ua-compatible": "IE=edge", "Content-Language": "en-ca",
 "x-generator": "Drupal 8 (https://www.drupal.org)", "x-dns-prefetch-control": "on", "x-drupal-cache": "MISS", "Last-Modified": "Thu, 19 Nov 2020 10:32:43 GMT", "ETag": "1605781963",
 "Content-Type": "text/html; charset=UTF-8", "x-acquia-host": "www.ibm.com",
 "x-acquia-path": "/ca-en", "x-acquia-site": "", "x-acquia-purge-tags": "", "x-varnish": "18482279_12683819", "x-cache-hits": "9", "x-age": "7030", "Accept-Ranges": "bytes",
 "Content-Encoding": "gzip", "Cache-Control": "public, max-age=300", "Expires": "Thu, 19 Nov 2020 15:26:47 GMT", "X-Akamai-Transformed": "9 11615 0 pmb=mTOE,1", "Date": "Thu, 19 Nov 2020 15:21:47 GMT", "Content-Length": "11725", "Connection": "keep-alive", "Vary": "Accept-Encoding", "x-content-type-options": "nosniff", "X-XSS-Protection": "1; mode=block",
 "Content-Security-Policy": "upgrade-insecure-requests", "Strict-Transport-Security": "max-age=31536000", "x-ibm-trace": "www-dipatcher: dynamic rule"}
```

Requests

```
header['date']:'Thu, 19 Nov 2020 15:21:47 GMT'
header['Content-Type']:'text/html; charset=UTF-8'
r.encoding:'UTF-8'
r.text[0:100]:
'<!DOCTYPE html>\n<html lang="en-ca" dir="ltr">\n<head>\n <meta charset="utf-8" />\n<script>digitalD'
```

Get Request with URL Parameters

GET request

| Base URL | Route |
|-----------------|-------|
| httpbin.org | /get |
| httpbin.org/get | |

Query string

| Start of Query | Parameter Name | Value | Parameter Name | Value |
|--|----------------|----------|----------------|----------|
| ? | name | = Joseph | & | ID = 123 |
| http://httpbin.org/get? Name=Joseph&ID=123 | | | | |

Create Query string

```
url_get='http://httpbin.org/get'
payload={"name":"Joseph","ID":"123"}
r=requests.get(url_get,params=payload)
r.url='http://httpbin.org/get?name=Joseph&ID=123'
r.request.body : None
```

Content-Type

r.text

```
{"args": { "ID": "123", "name": "Joseph" },
"headers": { "Accept": "*/*", "Accept-Encoding": "gzip, deflate", "Host": "httpbin.org", "User-Agent": "python-requests/2.24.0", "X-Amzn-Trace-Id": "Root=1-5fbdd03e-106584ab42513a6d5cef39a9" }, "origin": "99.228.137.181", "url": "http://httpbin.org/get?name=Joseph&ID=123" }
```

r.headers['Content-Type'] 'application/json'

Content-Type

```
r.json()
{'args': {'ID': '123', 'name': 'Joseph'},
'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate', 'Host': 'httpbin.org', 'User-Agent': 'python-requests/2.24.0', 'X-Amzn-Trace-Id': 'Root=1-5fbdd03e-106584ab42513a6d5cef39a9'}, 'origin': '99.228.137.181', 'url': 'http://httpbin.org/get?name=Joseph&ID=123'}
```



```
r.json()['args'] {'ID': '123', 'name': 'Joseph'}
```

Post Requests

POST

```
url_post="http://httpbin.org/post"  
payload={"name": "Joseph", "ID": "123"}  
  
r_post=requests.post(url_post,data=payload)
```



Compare POST and GET

```
print("POST request URL:",r_post.url )  
print("GET request URL:",r.url)  
  
POST request URL: http://httpbin.org/post  
GET request URL: http://httpbin.org/get?name=Joseph&ID=123
```

Compare POST and GET

```
print("POST request body:",r_post.request.body)  
print("GET request body:",r.request.body)  
  
POST request body: name=Joseph&ID=123  
GET request body: None  
  
r_post.json()['form']  
  
{'ID': '123', 'name': 'Joseph'}
```



HTTP and Requests

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Understand HTTP
- Handle the HTTP Requests

Table of Contents

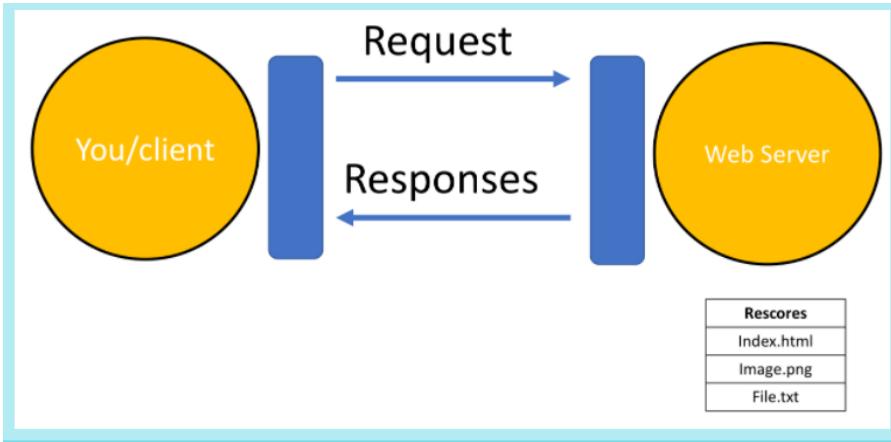
- Overview of HTTP
 - Uniform Resource Locator:URL
 - Request
 - Response
- Requests in Python
 - Get Request with URL Parameters
 - Post Requests

Overview of HTTP

When you, the client, use a web page your browser sends an HTTP request to the server where the page is hosted. The server tries to find the desired resource by default "index.html". If your request is successful, the server will send the object to the client in an HTTP response; this includes information like the type of the resource, the length of the resource, and other information.

The figure below represents the process; the circle on the left represents the client, the circle on the right represents the Web server. The table under the Web server represents a list of resources stored in the web server. In this case an HTML file, png image, and txt file .

The HTTP protocol allows you to send and receive information through the web including webpages, images, and other web resources. In this lab, we will provide an overview of the Requests library for interacting with the HTTP protocol. </p>



Uniform Resource Locator:URL

Uniform resource locator (URL) is the most popular way to find resources on the web. We can break the URL into three parts.

- **scheme** this is this protocol, for this lab it will always be http://
- **Internet address or Base URL** this will be used to find the location here are some examples: www.ibm.com and www.gitlab.com
- **route** location on the web server for example: /images/IDSNIlogo.png

You may also hear the term uniform resource identifier (URI), URL are actually a subset of URIs. Another popular term is endpoint, this is the URL of an operation provided by a Web server.

Request

The process can be broken into the **request** and **response** process. The request using the get method is partially illustrated below. In the start line we have the GET method, this is an HTTP method. Also the location of the resource /index.html and the HTTP version .The Request header passes additional information with an HTTP request:

Request Message

| | |
|---------------------------|---|
| Request Start line | Get/index.html HTTP/1.0 |
| Request Header | User-Agent: python-requests/2.21.0 Accept-Encoding: gzip, deflate : |

When an HTTP request is made, an HTTP method is sent, this tells the server what action to perform. A list of several HTTP methods is shown below. We will go over more examples later.

| HTTP METHODS | Description |
|---------------------|--------------------------------|
| GET | Retrieves Data from the server |
| POST | Submits data to server |
| PUT | Updates data already on server |
| DELETE | Deletes data from server |

When an HTTP request is made, an HTTP method is sent, this tells the server what action to perform. A list of several HTTP methods is shown below. We will go over more examples later.

| HTTP METHODS | Description |
|---------------------|--------------------------------|
| GET | Retrieves Data from the server |
| POST | Submits data to server |
| PUT | Updates data already on server |
| DELETE | Deletes data from server |

Response

The figure below represents the response; the response start line contains the version number HTTP/1.0, a status code (200) meaning success, followed by a descriptive phrase (OK). The response header contains useful information. Finally, we have the response body containing the requested file an HTML document. It should be noted that some request have headers.

Response Message

| | |
|----------------------------|--|
| Response Start line | HTTP/1.0 200 OK |
| Response Header | Server: Apache-Cache:UNCACHEABLE |
| Response Body | <!DOCTYPE html> <html> <body> <h1>My First Heading</h1> <p>My first paragraph.</p> </body> </html> |

Some status code examples are shown in the table below, the prefix indicates the class; these are shown in yellow, with actual status codes shown in white. Check out the following [link](#) for more descriptions.

| | |
|-----|------------------|
| 1XX | Informational |
| 2xx | Success |
| 200 | OK |
| 3XX | Redirection |
| 300 | Multiple Choices |
| 4XX | Client Error |
| 401 | Unauthorized |
| 403 | Forbidden |
| 404 | Not Found |

Requests in Python

Requests is a python Library that allows you to send HTTP/1.1 requests easily. We can import the library as follows:

```
In [1]: import requests
```

We will also use the following libraries

```
In [2]: import os
from PIL import Image
from IPython.display import IFrame
```

You can make a `GET` request via the method `get` to www.ibm.com:

```
In [3]: url='https://www.ibm.com/'
r=requests.get(url)
```

We have the response object `r`, this has information about the request, like the status of the request. We can view the status code using the attribute `status_code`

```
In [4]: r.status_code
```

```
Out[4]: 200
```

You can view the request headers:

```
In [11]: print(r.request.headers)
```

```
{'User-Agent': 'python-requests/2.25.1', 'Accept-Encoding': 'gzip, deflate', 'Accept': '*/*', 'Connection': 'keep-alive', 'Cookie': 'bm_sz=F26A528E8DCFC8E58AEDA125CB69EC34~YAAQjicwFy+ueZB5AQAAEqOikQv0lzuizwxEU8+K9TtBPQxweOKJqLq7ZqdDs6tRnxMCo32LETwcvwr2ov4zT9CL8/aWYSaxlocqxTdiXI77xmHGHDN5I3NIE4nRgotUqQvPTiiDq7oWqurSxUxd8RIIEI3RL6hHwfKE7cIej+FYqR0t+2bUvkj/zDA; _abck=8D80B32F26B70BCC8756A4CF004458D2~-1~YAAQjicwFzCueZB5AQAAEqOikQVvoq+FbCT/ikfQsoMwlbLWN4760FIN0iZES3GcLxZ+HNPalQdvnIfMlc3krbRmZ6rL2ZCuz1vDxSaCzc0RbeBi4fcldW/Zhxwszg7yWHLm+k1KxkZ0Q4s77fsaC5ooxqnBXW+mx8/1iLAI7U+Rdhw8wObo6FPL6SqSGx2d00s/zaQ8Jvb+hQzstWaxDTwzkOi7vx4JQLNlrVtvFGsJhzuVDFPM45Mo0kf8LeDRltI/N36cQcs62gvz5nibfnDZvpmSgbVxWcOadAxdyupLI/rGHgV0/OGFhqq/pntlYz+sW0LN+Gc82f4uRmTv/DUpXsWxJm2BM7s7IFHM2xCqtJUextz4=~-1~-1~-1'}
```

You can view the request body, in the following line, as there is no body for a get request we get a None :

```
In [15]: print("request body:", r.request.body)
```

```
request body: None
```

You can view the HTTP response header using the attribute headers. This returns a python dictionary of HTTP response headers.

```
In [16]: header=r.headers  
print(r.headers)
```

```
{'Cache-Control': 'max-age=301', 'Expires': 'Tue, 18 May 2021 17:09:58 GMT', 'Last-Modified': 'Tue, 18 May 2021 14:44:54 GMT', 'ETag': '"17af1-5c29bc0d2175b"', 'Accept-Ranges': 'bytes', 'Content-Encoding': 'gzip', 'Content-Type': 'text/html', 'X-Akamai-Transformed': '9 17376 0 pmb=mTOE,1', 'Date': 'Sat, 22 May 2021 01:13:45 GMT', 'Content-Length': '17481', 'Connection': 'keep-alive', 'Vary': 'Accept-Encoding', 'x-content-type-options': 'nosniff', 'X-XSS-Protection': '1; mode=block', 'Content-Security-Policy': 'upgrade-insecure-requests', 'Strict-Transport-Security': 'max-age=31536000'}
```

We can obtain the date the request was sent using the key Date

```
In [17]:
```

```
header['date']
```

```
Out[17]:
```

```
'Sat, 22 May 2021 01:13:45 GMT'
```

Content-Type indicates the type of data:

```
In [18]:
```

```
header['Content-Type']
```

```
Out[18]:
```

```
'text/html'
```

You can also check the encoding :

```
In [19]:
```

```
r.encoding
```

```
Out[19]:
```

```
'ISO-8859-1'
```

As the Content-Type is text/html we can use the attribute text to display the HTML in the body. We can review the first 100 characters:

```
In [20]:
```

```
r.text[0:100]
```

```
Out[20]:
```

```
'<!DOCTYPE html><html lang="en-US"><head><meta name="viewport" content="width=device-width"/><meta ch'
```

```
In [24]:
```

```
r.text[:300]
```

```
Out[24]: '<!DOCTYPE html><html lang="en-US"><head><meta name="viewport" content="width=device-width"/><meta charset="utf-8"/><title>IBM - United States</title><link rel="canonical" href="https://www.ibm.com/us-en/"><meta name="robots" content="index,follow"/><meta name="description" content="For more than a '
```

You can load other types of data for non-text requests like images, consider the URL of the following image:

```
In [25]:
```

```
# Use single quotation marks for defining string
```

```
url='https://gitlab.com/ibm/skills-network/courses/placeholder101/-/raw/master/labs/module%201/images/IDSNlogo.png'
```

We can make a get request:

```
In [26]:
```

```
r=requests.get(url)
```

We can look at the response header:

```
In [27]:
```

```
print(r.headers)
```

```
{'Date': 'Sat, 22 May 2021 01:22:56 GMT', 'Content-Type': 'image/png', 'Content-Length': '21590', 'Connection': 'keep-alive', 'Cache-Control': 'max-age=60, public', 'Content-Disposition': 'inline', 'Etag': 'W/"c26d88d0ca290ba368620273781ea37c"', 'Permissions-Policy': 'interest-cohort=()', 'X-Content-Type-Options': 'nosniff', 'X-Download-Options': 'noopener', 'X-Frame-Options': 'DENY', 'X-Gitlab-Feature-Category': 'source_code_management', 'X-Permitted-Cross-Domain-Policies': 'none', 'X-Request-Id': '01F68TP2PPNPC3W0DWBNR7RXW', 'X-Runtime': '0.059387', 'X-Ua-Compatible': 'IE=edge', 'X-Xss-Protection': '1; mode=block', 'Strict-Transport-Security': 'max-age=31536000', 'Referrer-Policy': 'strict-origin-when-cross-origin', 'GitLab-LB': 'fe-12-lb-gprd', 'GitLab-SV': 'web-15-sv-gprd', 'CF-Cache-Status': 'EXPIRED', 'Accept-Ranges': 'bytes', 'cf-request-id': '0a334422a800005ed82a0be000000001', 'Expect-CT': 'max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"', 'Vary': 'Accept-Encoding', 'Server': 'cloudflare', 'CF-RAY': '65323c7ddc7c5ed8-IAD'}
```

We can see the 'Content-Type'

```
In [28]: r.headers['Content-Type']
```

```
Out[28]: 'image/png'
```

An image is a response object that contains the image as a bytes-like object. As a result, we must save it using a file object. First, we specify the file path and name

```
In [29]: path=os.path.join(os.getcwd(),'image.png')  
path
```

```
Out[29]: '/resources/labs/image.png'
```

We save the file, in order to access the body of the response we use the attribute content then save it using the open function and write method:

```
In [30]: with open(path,'wb') as f:  
    f.write(r.content)
```

We can view the image:

```
In [31]: Image.open(path)
```

```
Out[31]:
```



Question 1: write wget

In the previous section, we used the wget function to retrieve content from the web server as shown below. Write the python code to perform the same task. The code should be the same as the one used to download the image, but the file name should be 'example.txt'.

```
!wget -O /resources/data/Example1.txt
```

```
In [36]: url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/Example1.txt'
path=os.path.join(os.getcwd(),'example1.txt')
print(path)
r=requests.get(url)
with open(path,'wb') as f:
    f.write(r.content)
```

/resources/labs/example1.txt

Click here for the solution ````python url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/Example1.txt'
path=os.path.join(os.getcwd(),'example1.txt') r=requests.get(url) with open(path,'wb') as f:
f.write(r.content)````

Get Request with URL Parameters

You can use the GET method to modify the results of your query, for example retrieving data from an API. We send a GET request to the server. Like before we have the Base URL, in the Route we append /get this indicates we would like to perform a GET request, this is demonstrated in the following table:

| Base URL | Route |
|----------------|-------|
| httbin.org | /get |
| httbin.org/get | |

The Base URL is for

[http://httpbin.org/](http://httpbin.org/?utm_email=Email&utm_source=Nurture&utm_content=000026UJ&utm_term=10006555&utm_campaign=PLACEHOLDER&utm_id=SkillsNetwork-Courses-IBMDveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395) is a simple HTTP Request & Response Service. The URL in Python is given by:

```
In [37]: url_get='http://httpbin.org/get'
```

A [query string](#) is a part of a uniform resource locator (URL), this sends other information to the web server. The start of the query is a ?, followed by a series of parameter and value pairs, as shown in the table below. The first parameter name is name and the value is Joseph the second parameter name is ID and the Value is 123. Each pair, parameter and value is separated by an equals sign, =. The series of pairs is separated by the ampersand &.

| Start of Query | Parameter Name | Value | Parameter Name | Value |
|---|----------------|----------|----------------|----------|
| ? | name | = Joseph | & | ID = 123 |
| http://httpbin.org/get?Name=Joseph&ID=123 | | | | |

To create a Query string, add a dictionary. The keys are the parameter names, and the values are the value of the Query string.

```
In [38]: payload={"name":"Joseph","ID":"123"}
```

Then passing the dictionary `payload` to the `params` parameter of the `get()` function:

```
In [39]: r=requests.get(url_get,params=payload)
```

We can print out the `URL` and see the name and values

```
In [40]: r.url
```

```
Out[40]: 'http://httpbin.org/get?name=Joseph&ID=123'
```

There is no request body

```
In [41]: print("request body:", r.request.body)
```

```
request body: None
```

We can print out the status code

```
In [42]: print(r.status_code)
```

```
200
```

We can view the response as text:

```
In [43]: print(r.text)
```

```
{
  "args": {
    "ID": "123",
    "name": "Joseph"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.25.1",
    "X-Amzn-Trace-Id": "Root=1-60a85e91-1c6f09a81f8f14502d1b26b4"
  },
  "origin": "169.63.179.135",
  "url": "http://httpbin.org/get?name=Joseph&ID=123"
}
```

We can look at the 'Content-Type' .

```
In [44]: r.headers['Content-Type']
```

```
Out[44]: 'application/json'
```

As the content 'Content-Type' is in the JSON format we can use the method json() , it returns a Python dict:

```
In [45]: r.json()
```

```
Out[45]: {'args': {'ID': '123', 'name': 'Joseph'},
          'headers': {'Accept': '*/*',
                      'Accept-Encoding': 'gzip, deflate',
                      'Host': 'httpbin.org',
                      'User-Agent': 'python-requests/2.25.1',
                      'X-Amzn-Trace-Id': 'Root=1-60a85e91-1c6f09a81f8f14502d1b26b4'},
          'origin': '169.63.179.135',
          'url': 'http://httpbin.org/get?name=Joseph&ID=123'}
```

The key args had the name and values:

```
In [46]: r.json()['args']
```

```
Out[46]: {'ID': '123', 'name': 'Joseph'}
```

```
In [50]: r.json()['headers']['Host']
```

```
Out[50]: 'httpbin.org'
```

Post Requests

Like a GET request a POST is used to send data to a server, but the POST request sends the data in a request body. In order to send the Post Request in Python in the URL we change the route to POST:

```
In [51]: url_post='http://httpbin.org/post'
```

This endpoint will expect data as a file or as a form, a from is convenient way to configure an HTTP request to send data to a server.

To make a POST request we use the post() function, the variable payload is passed to the parameter data :

```
In [53]: r_post=requests.post(url_post,data=payload)
```

Comparing the URL from the response object of the GET and POST request we see the POST request has no name or value pairs.

```
In [54]: print("POST request URL:",r_post.url )  
print("GET request URL:",r.url)
```

```
POST request URL: http://httpbin.org/post  
GET request URL: http://httpbin.org/get?name=Joseph&ID=123
```

We can compare the `POST` and `GET` request body, we see only the `POST` request has a body:

```
In [55]: print("POST request body:",r_post.request.body)  
print("GET request body:",r.request.body)
```

```
POST request body: name=Joseph&ID=123  
GET request body: None
```

We can view the form as well:

```
In [56]: r_post.json()['form']
```

```
Out[56]: {'ID': '123', 'name': 'Joseph'}
```

There is a lot more you can do check out [Requests](#) for more.

```
In [57]: r_post.json()
```

```
Out[57]: {'args': {},  
          'data': '',  
          'files': {},  
          'form': {'ID': '123', 'name': 'Joseph'},  
          'headers': {'Accept': '*/*',  
                      'Accept-Encoding': 'gzip, deflate',  
                      'Content-Length': '18',  
                      'Content-Type': 'application/x-www-form-urlencoded',  
                      'Host': 'httpbin.org',  
                      'User-Agent': 'python-requests/2.25.1',  
                      'X-Amzn-Trace-Id': 'Root=1-60a85f54-18241f155de3264625e3346d'},  
          'json': None,  
          'origin': '169.63.179.135',  
          'url': 'http://httpbin.org/post'}
```

HTML for Webscraping

In this video we will review Hypertext Markup Language or HTML for Webscraping.

Lots of useful data is available on web pages, such as real estate prices and solutions to coding questions.

The website Wikipedia is a repository of the world's information. If you have an understanding of HTML, you can use Python to extract this information.

In this video, you will: review the HTML of a basic web page; understand the Composition of an HTML Tag; understand HTML Trees; and understand HTML Tables.

Let's say you were asked to find the name and salary of players in a National Basketball League from the following web page.

The web page is comprised of HTML. It consists of text surrounded by a series of blue text elements enclosed in angle brackets called tags. The tags tell the browser how to display the content. The data we require is in this text.

The first portion contains the "DOCTYPE html" which declares this document is an HTML document.

element is the root element of an HTML page, and element contains meta information about the HTML page.

Next, we have the body, this is what's displayed on the web page. This is usually the data we are interested in, we see the elements with an "h3",

this means type 3 heading, makes the text larger and bold. These tags have the names of the players, notice the data is enclosed in the elements. It starts with a h3 in brackets and ends in a slash h3 in brackets.

There is also a different tag "p", this means paragraph, each p tag contains a player's salary. Let's take a closer look at the composition of an HTML tag. Here is an example of an HTML Anchor tag. It will display IBM and when you click it, it will send you to IBM.com.

We have the tag name, in this case "a". This tag defines a hyperlink, which is used to link from one page to another. It's helpful to think of each tag name as a class in Python and each individual tag as an instance.

We have the opening or start tag and we have the end tag. This has the tag name preceded by a slash.

These tags contain the content, in this case what's displayed on the web page. We have the attribute, this is composed of the

Attribute Name and Attribute Value.

In this case it the url to the destination web page. Real web pages are more complex, depending on your browser you can select the HTML element, then click Inspect. The result will give you the ability to inspect the HTML. There is also other types of content such as CSS and JavaScript that we will not go over in this course.

The actual element is shown here. Each HTML document can actually be referred to as a document tree. Let's go over a simple example. Tags may contain strings and other tags.

These elements are the tag's children. We can represent this as a family tree. Each nested tag is a level in the tree. The tag HTML tag contains the head and body tag. The Head and body tag are the descendants of the html tag.

In particular they are the children of the HTML tag.

HTML tag is their parent.

The head and body tag are siblings as they are on the same level.

Title tag is the child of the head tag and its parent is the head tag.

The title tag is a descendant of the HTML tag but not its child.

The heading and paragraph tags are the children of the body tag;

and as they are all children of the body tag they are siblings of each other.

The bold tag is a child of the heading tag,

the content of the tag is also part of the tree but this can get unwieldy to draw.

Next, let's review HTML tables.

To define an HTML table we have the table tag. tags, each defines a table cell.

For the first row first cell we have; for the first row second cell we have; and so on.

For the second row we have; and for the second row first cell we have; for the second row second cell we have; and so on. We now have some basic knowledge of HTML.

Now let's try and extract some data from a web page.

The slide has a blue header 'HTML for Web Scraping'. Below it is a section titled 'Web Pages' with three examples:

- Zillow: A real estate search website showing a map of a neighborhood with house listings.
- Wikipedia: The free encyclopedia, showing the main page with various articles and categories.
- Stack Overflow: A Q&A site for programmers, showing a question about Python loops.

At the bottom left, there is a list of learning objectives:

- Review the HTML of a basic web page
- Understand the Composition of an HTML Tag
- Understand HTML Trees
- Understand HTML Tables

HTML Tags

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

Lebron James

Salary: \$ 92,000,000

Stephen Curry

Salary: \$85,000, 000

Kevin Durant

Salary: \$73,200, 000

HTML Composition

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

Lebron James

Salary: \$ 92,000,000

Stephen Curry

Salary: \$85,000, 000

Kevin Durant

Salary: \$73,200, 000

HTML Composition

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

Lebron James

Salary: \$ 92,000,000

Stephen Curry

Salary: \$85,000, 000

Kevin Durant

Salary: \$73,200, 000

The first portion contains the "DOCTYPE html" which declares this document is an HTML document element is the root element of an HTML page, and element contains meta information about the HTML page

HTML Composition

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

Lebron James

Salary: \$ 92,000,000

Stephen Curry

Salary: \$85,000, 000

Kevin Durant

Salary: \$73,200, 000

Next, we have the body, this is what's displayed on the web page. This is usually the data we are interested in, we see the elements with an "h3", this means type 3 heading, makes the text larger and bold. These tags have the names of the players, notice the data is enclosed in the elements. It starts with a h3 in brackets and ends in a slash h3 in brackets. There is also a different tag "p", this means paragraph, each p tag contains a player's salary.

HTML Elements

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000,000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200,000</p>
</body>
</html>
```

Lebron James

Salary: \$ 92,000,000

Stephen Curry

Salary: \$85,000,000

Kevin Durant

Salary: \$73,200,000

Next, we have the body, this is what's displayed on the web page. This is usually the data we are interested in, we see the elements with an "h3", this means type 3 heading, makes the text larger and bold. These tags have the names of the players, notice the data is enclosed in the elements. It starts with a h3 in brackets and ends in a slash h3 in brackets. There is also a different tag "p", this means paragraph, each p tag contains a player's salary.

Composition of an HTML Tag

HTML Anchor Tag



it will send you to IBM.com. We have the tag name, in this case "a". This tag defines a hyperlink, which is used to link from one page to another. It's helpful to think of each tag name as a class in Python and each individual tag as an instance.

Hyperlink Tag



It's helpful to think of each tag name as a class in Python and each individual tag as an instance. We have the opening or start tag and we have the end tag. This has the tag name preceded by a slash. These tags contain the content, in this case what's displayed on the web page.

Opening and End Tags



We have the opening or start tag and we have the end tag. This has the tag name preceded by a slash. These tags contain the content, in this case what's displayed on the web page.

Opening and End Tags



We have the opening or start tag and we have the end tag. This has the tag name preceded by a slash. These tags contain the content, in this case what's displayed on the web page.

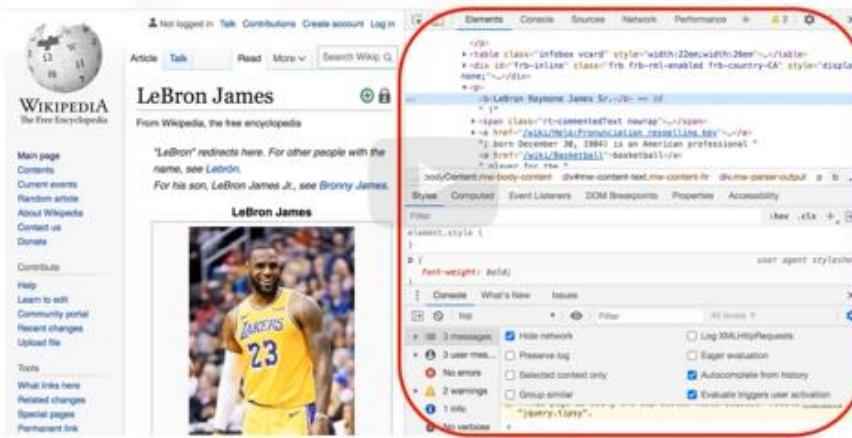
Attributes

Attributes



We have the attribute, this is composed of the Attribute Name and Attribute Value. In this case it is the url to the destination web page.

Inspect HTML



click Inspect. The result will give you the ability to inspect the HTML. There is also other types of content such as CSS and JavaScript that we will not go over in this course. The actual element is shown here.

HTML Trees

Document Tree

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h3> <b id='boldest'> Lebron James </b></h3>
    <p> Salary: $ 92,000,000 </p>
    <h3> Stephen Curry</h3>
    <p> Salary: $85,000, 000 </p>
    <h3> Kevin Durant </h3>
    <p> Salary: $73,200, 000</p>
  </body>
</html>
```



Each HTML document can actually be referred to as a document tree. Let's go over a simple example. Tags may contain strings and other tags. These elements are the tag's children. We can represent this as a family tree. Each nested tag is a level in the tree. The tag HTML tag contains the head and body tag. The Head and body tag are the descendants of the html tag. In particular they are the children of the HTML tag. HTML tag is their parent. The head and body tag are siblings as they are on the same level.

Document Tree

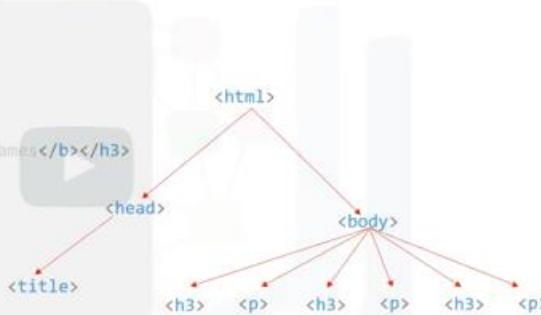
```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h3> <b id='boldest'> Lebron James </b></h3>
    <p> Salary: $ 92,000,000 </p>
    <h3> Stephen Curry</h3>
    <p> Salary: $85,000, 000 </p>
    <h3> Kevin Durant </h3>
    <p> Salary: $73,200, 000</p>
  </body>
</html>
```



The title tag is a descendant of the HTML tag but not its child. The heading and paragraph tags are the children of the body tag; and as they are all children of the body tag they are siblings of each other. The bold tag is a child of the heading tag, the content of the tag is also part of the tree but this can get unwieldy to draw.

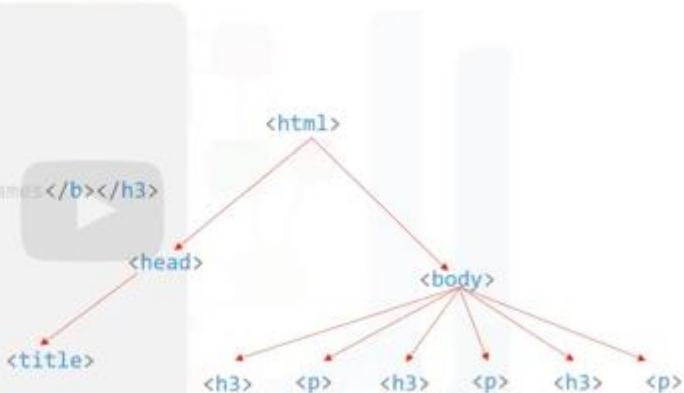
Document Tree

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h3> <b id='boldest'> Lebron James</b></h3>
    <p> Salary: $ 92,000,000 </p>
    <h3> Stephen Curry</h3>
    <p> Salary: $85,000, 000 </p>
    <h3> Kevin Durant </h3>
    <p> Salary: $73,200, 000</p>
  </body>
</html>
```



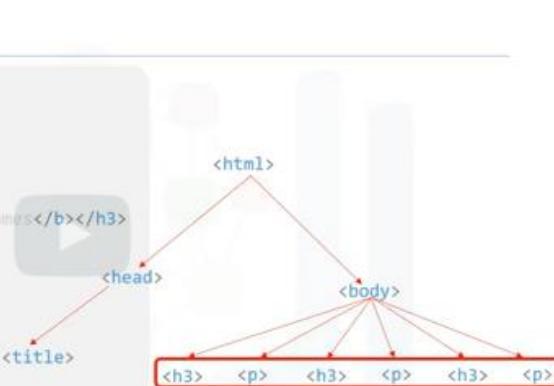
Document Tree

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h3> <b id='boldest'> Lebron James</b></h3>
    <p> Salary: $ 92,000,000 </p>
    <h3> Stephen Curry</h3>
    <p> Salary: $85,000, 000 </p>
    <h3> Kevin Durant </h3>
    <p> Salary: $73,200, 000</p>
  </body>
</html>
```



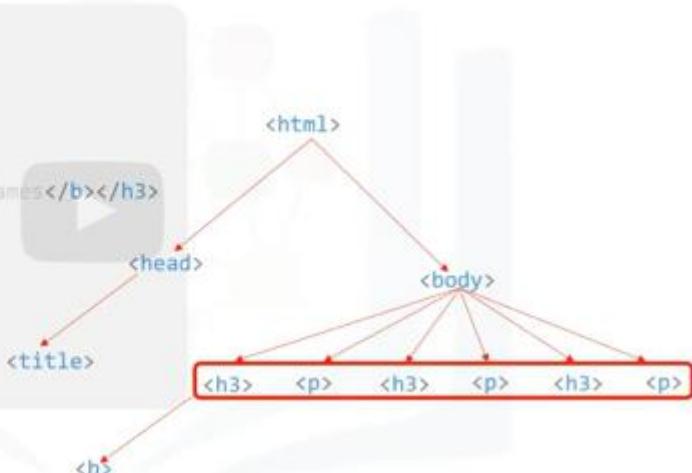
Document Tree

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h3> <b id='boldest'> Lebron James</b></h3>
    <p> Salary: $ 92,000,000 </p>
    <h3> Stephen Curry</h3>
    <p> Salary: $85,000, 000 </p>
    <h3> Kevin Durant </h3>
    <p> Salary: $73,200, 000</p>
  </body>
</html>
```



Document Tree

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h3> <b id='boldest'> Lebron James</b></h3>
    <p> Salary: $ 92,000,000 </p>
    <h3> Stephen Curry</h3>
    <p> Salary: $85,000,000 </p>
    <h3> Kevin Durant </h3>
    <p> Salary: $73,200,000</p>
  </body>
</html>
```



The title tag is a descendant of the HTML tag but not its child. The heading and paragraph tags are the children of the body tag; and as they are all children of the body tag they are siblings of each other. The bold tag is a child of the heading tag, the content of the tag is also part of the tree but this can get unwieldy to draw.

HTML Tables

HTML Tables

```
<table>
  <tr>
    <td>Pizza Place</td>
    <td>Orders</td>
    <td>Slices </td>
  </tr>
  <tr>
    <td>Domino Pizza</td>
    <td>10</td>
    <td>100</td>
  </tr>
  <tr>
    <td>Little Caesars</td>
    <td>12</td>
    <td>144 </td>
  </tr>
</table>
```

| Pizza Place | Orders | Slices |
|----------------|--------|--------|
| Domino's Pizza | 10 | 100 |
| Little Caesars | 12 | 144 |

HTML Tables

```
<table>
  <tr>
    <td>Pizza Place</td>
    <td>Orders</td>
    <td>Slices </td>
  </tr>
  <tr>
    <td>Domino Pizza</td>
    <td>10</td>
    <td>100</td>
  </tr>
  <tr>
    <td>Little Caesars</td>
    <td>12</td>
    <td>144 </td>
  </tr>
</table>
```

| Pizza Place | Orders | Slices |
|----------------|--------|--------|
| Domino's Pizza | 10 | 100 |
| Little Caesars | 12 | 144 |

HTML Tables

```
<table>
<tr>
  <td>Pizza Place</td>
  <td>Orders</td>
  <td>Slices </td>
</tr>
<tr>
  <td>Domino Pizza</td>
  <td>10</td>
  <td>100</td>
</tr>
<tr>
  <td>Little Caesars</td>
  <td>12</td>
  <td>144 </td>
</tr>
</table>
```

| Pizza Place | Orders | Slices |
|----------------|--------|--------|
| Domino's Pizza | 10 | 100 |
| Little Caesars | 12 | 144 |

HTML Tables

```
<table>
<tr>
  <td>Pizza Place</td>
  <td>Orders</td>
  <td>Slices </td>
</tr>
<tr>
  <td>Domino Pizza</td>
  <td>10</td>
  <td>100</td>
</tr>
<tr>
  <td>Little Caesars</td>
  <td>12</td>
  <td>144 </td>
</tr>
</table>
```

| Pizza Place | Orders | Slices |
|----------------|--------|--------|
| Domino's Pizza | 10 | 100 |
| Little Caesars | 12 | 144 |

HTML Tables

```
<table>
<tr>
  <td>Pizza Place</td>
  <td>Orders</td>
  <td>Slices </td>
</tr>
<tr>
  <td>Domino Pizza</td>
  <td>10</td>
  <td>100</td>
</tr>
<tr>
  <td>Little Caesars</td>
  <td>12</td>
  <td>144 </td>
</tr>
</table>
```

| Pizza Place | Orders | Slices |
|----------------|--------|--------|
| Domino's Pizza | 10 | 100 |
| Little Caesars | 12 | 144 |

To define an HTML table we have the `table` tag. `td` tags, each defines a table cell. For the first row first cell we have; for the first row second cell we have; and so on. For the second row we have; and for the second row first cell we have; for the second row second cell we have; and so on. We now have some basic knowledge of HTML.

Webscraping

In this video we will cover Webscraping. After watching this video you will be able to: define web scraping; understand the role of BeautifulSoup Objects; apply the find_all method; and webscrape a website. What would you do if you wanted to analyze hundreds of points of data to find the best players of a sports team?

Would you start manually copying and pasting information from different websites into a spreadsheet? Spending hours trying to find the right data, and eventually giving up because the task was to overwhelming?

That's where webscraping can help. Webscraping is a process that can be used to automatically extract information from a website, and can easily be accomplished within a matter of minutes and not hours. To get started we just need a little Python code and the help of two modules named Requests and Beautiful Soup.

Let's say you were asked to find the name and salary of players in a National Basketball League, from the following webpage. We import BeautifulSoup. We can store the webpage HTML as a string in the variable HTML.

To parse a document, pass it into the BeautifulSoup constructor. We get the BeautifulSoup object , soup, which represents the document as a nested data structure.

BeautifulSoup represents HTML as a set of Tree like objects with methods used to parse the HTML. We will review the BeautifulSoup object Using the BeautifulSoup object, soup, we created The tag object corresponds to an HTML tag in the original document. For example, the tag "title." Consider the tag . If there is more than one tag with the same name, the first element with that tag is selected. In this case with Lebron James, we see the name is Enclosed in the bold attribute "b". To extract it, use the Tree representation.

Let's use the Tree representation. The variable tag-object is located here. We can access the child of the tag or navigate down the branch as follows: You can navigate up the tree by using the parent attribute.

The variable tag child is located here. We can access the parent. This is the original tag object. We can find the sibling of "tag object." We simply use the next sibling attribute.

We can find the sibling of sibling one. We simply use the next sibling attribute.

Consider the tag child object. You can access the attribute name and value as a key value pair in a dictionary as follows. You can return the content as a Navigable string, this is like a Python string that supports BeautifulSoup functionality.

Let's review the method find_all. This is a filter, you can use filters to filter based on a tag's name, it's attributes, the text of a string, or on some combination of these.

Consider the list of pizza places. Like before, create a BeautifulSoup object. But this time, name it table.

The `find_all()` method looks through a tag's descendants and retrieves all descendants

The result is a Python iterable just like a list. This corresponds to each row in the list-including the table header. Each element is a tag object. Consider the first row.

For example, we can extract the first table cell. We can also iterate through each table cell.

First, we iterate through the list "table rows," via the variable `row`. Each element corresponds to a row in the table. We can apply the method `find_all` to find all the table cells, then we can iterate through the variable cells for each row. For each iteration, the variable cell corresponds to an element in the table for that particular row.

We continue to iterate through each element and repeat the process for each row.

Let's see how to apply BeautifulSoup to a webpage. To scrape a webpage we also need the Requests library. The first step is to import the modules that are needed. Use the `get` method from the requests library to download the webpage. The input is the URL. Use the `text` attribute to get the text and assign it to the variable `page`. Then, create a BeautifulSoup object 'soup' from the variable `page`. It will allow you to parse through the HTML page.

You can now scrape the Page. Check out the labs for more.

Webscraping

Objectives

- Define Webscraping
- Beautiful Soup Objects
- `Find_all`
- Webscrape a website

Introduction

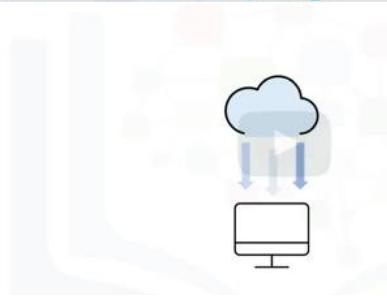


Introduction

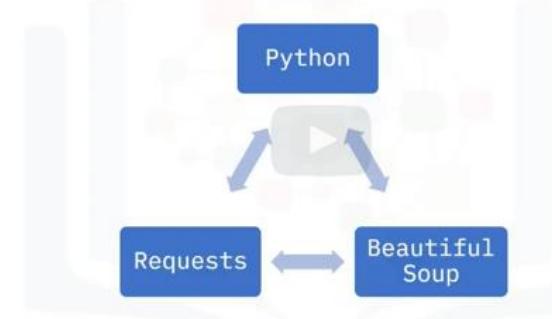


What would you do if you wanted to analyze hundreds of points of data to find the best players of a sports team?

What is Webscraping?



What is Webscraping?



Webscraping is a process that can be used to automatically extract information from a website, and can easily be accomplished within a matter of minutes and not hours. To get started we just need a little Python code and the help of two modules named Requests and BeautifulSoup.

Problem: Let's say you were asked to find the name and salary of players in a National Basketball League, from the following webpage.

Webscraping example

Lebron James

Salary: \$ 92,000,000

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

Stephen Curry

Salary: \$85,000, 000

Kevin Durant

Salary: \$73,200, 000

Solution:

Beautiful Soup

```
from bs4 import BeautifulSoup

html="<!DOCTYPE html><html><head><title>Page
Title</title></head><body><h3><b id='boldest'>Lebron
James</b></h3><p> Salary: $ 92,000,000 </p><h3> Stephen
Curry</h3><p> Salary: $85,000, 000 </p><h3> Kevin Durant </h3>
Salary: $73,200, 000</p></body></html>

soup = BeautifulSoup(html, 'html5lib')
```

1. We import BeautifulSoup.

2. We can store the webpage HTML as a string in the variable HTML.
3. To parse a document, pass it into the BeautifulSoup constructor.
4. We get the Beautiful Soup object , soup, which represents the document as a nested data structure.

Beautiful Soup Objects

BeautifulSoup represents HTML as a set of Tree like objects with methods used to parse the HTML.

Tag Object

```
tag_object=soup.title
tag_object:
<title>Page Title</title>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

Using the BeautifulSoup object, soup, we created The tag object corresponds to an HTML tag in the original document. For example, the tag "title."

Tag Object

```
tag_object=soup.title
tag_object:
<title>Page Title</title>
```

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: $ 92,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

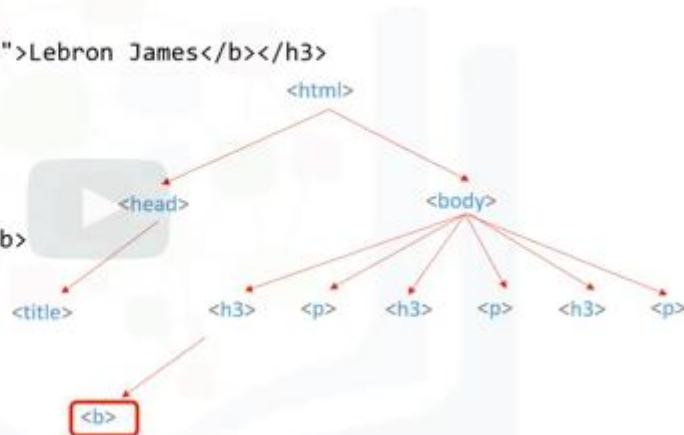
Consider the tag <h3>. If there is more than one tag with the same name, the first element with that tag is selected. In this case with Lebron James, we see the name is Enclosed in the bold attribute "b". To extract it, use the Tree representation.

HTML Tree

```
tag_object: <h3><b id="boldest">Lebron James</b></h3>
```

```
tag_child =tag_object.b  
tag_child:
```

```
<b id="boldest">Lebron James</b>
```



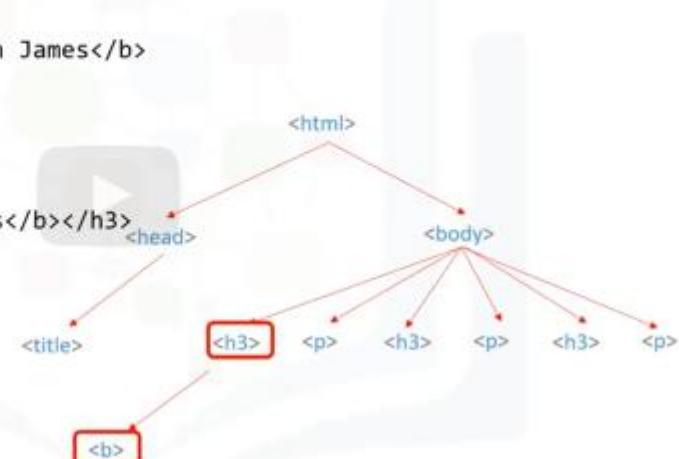
1. The variable tag-object is located here.
2. We can access the child of the tag or navigate down the branch as follows:
3. You can navigate up the tree by using the parent attribute.
4. The variable tag child is located here.

Parent attribute

```
tag_child:<b id="boldest">Lebron James</b>
```

```
parent_tag=tag_child.parent  
parent_tag:
```

```
<h3><b id="boldest">Lebron James</b></h3>
```



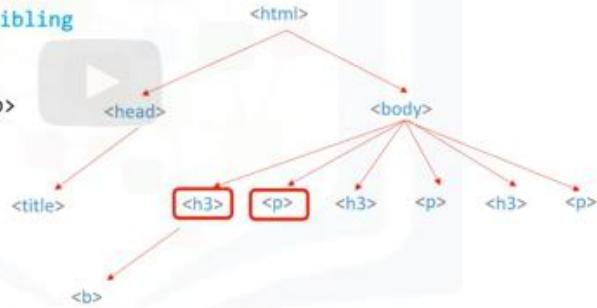
5. We can access the parent. This is the original tag object.

Next-sibling attribute

```
tag_object: <h3><b id="boldest">Lebron James</b></h3>
```

```
sibling_1= tag_object.next_sibling  
sibling_1:
```

```
<p> Salary: $ 92,000,000 </p>
```



Next-sibling attribute

```
sibling_1: <p> Salary: $ 92,000,000 </p>
```

```
sibling_2=sibling_1.next_sibling  
sibling_2:
```

```
<h3> Stephen Curry</h3>
```



1. We simply use the next sibling attribute.
2. We can find the sibling of sibling one.
3. We simply use the next sibling attribute.
4. Consider the tag child object.
5. You can access the attribute name and value as a key value pair in a dictionary as follows.

Navigable string

```
tag_child:<b id="boldest">Lebron James</b>
tag_child.attrs:
{'id': 'boldest'}
tag_child.string:
'Lebron James'

<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h3><b id='boldest'> Lebron James</b></h3>
<p> Salary: > $2,000,000 </p>
<h3> Stephen Curry</h3>
<p> Salary: $85,000, 000 </p>
<h3> Kevin Durant </h3>
<p> Salary: $73,200, 000</p>
</body>
</html>
```

1. Consider the tag child object.
2. You can access the attribute name and value as a key value pair in a dictionary as follows.
3. You can return the content as a Navigable string, this is like a Python string that supports BeautifulSoup functionality.

find_all

Pizza place list

```
<table >
<tr>
<td>Pizza Place</td>
<td>Orders</td>
<td>Slices </td>
</tr>
<tr>
<td>Domino's Pizza</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Little Caesars</td>
<td>12</td>
<td >144 </td>
</tr>
</table>
```

| Pizza Place | Orders | Slices |
|----------------|--------|--------|
| Domino's Pizza | 10 | 100 |
| Little Caesars | 12 | 144 |

Beautiful Soup object

```
from bs4 import BeautifulSoup

html= "<table><tr><td>Pizza Place</td><td>Orders</td>
<td>Slices </td></tr><tr><td>Domino's Pizza</td>
<td>10</td><td>100</td></tr><tr><td>Little
Caesars</td><td>12</td><td>144</td></table>

table = BeautifulSoup(html, 'html5lib')
```

Python iterable

```
table_row=table.find_all(name='tr')
table_row:
```

| Pizza Place | Orders | Slices |
|----------------|--------|--------|
| Domino's Pizza | 10 | 100 |
| Little Caesars | 12 | 144 |

```
[<tr><td>Pizza Place</td><td>Orders</td><td>Slices </td></tr>,
<tr><td>Domino's Pizza</td><td>10</td><td>100</td></tr>,
<tr><td>Little Caesars</td><td>12</td><td>144</td></tr>,
```

Tag object

```
first_row =table_row[0]
first_row:
<tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></tr>

first_row.td :
<td>Pizza Place</td>
```

Each element is a tag object. Consider the first row.

For example, we can extract the first table cell.

Variable row

| Pizza Place | Orders | Slices |
|----------------|--------|--------|
| Domino's Pizza | 10 | 100 |
| Little Caesars | 12 | 144 |
| Papa John's | 15 | 166 |

```
for i, row in enumerate(table_rows):
    print("row", i)
    cells = row.find_all("td")

    for j, cell in enumerate(cells):
        print("column", j, "cell", cell)
```

We can also iterate through each table cell. First, we iterate through the list “table rows,” via the variable row.

Elements

| Pizza Place | Orders | Slices |
|----------------|--------|--------|
| Domino's Pizza | 10 | 100 |
| Little Caesars | 12 | 144 |
| Papa John's | 15 | 166 |

```
for i, row in enumerate(table_rows):
    print("row", i)
    cells = row.find_all("td")

    for j, cell in enumerate(cells):
        print("column", j, "cell", cell)
```

Each element corresponds to a row in the table. We can apply the method find all to find all the table cells, then we can iterate through the variable cells for each row. For each iteration, the variable cell corresponds to an element in the table for that particular row. We continue to iterate through each element and repeat the process for each row.

A Webpage Example

Overview Python Program

```
import requests
from bs4 import BeautifulSoup

page = requests.get("http://EnterWebsiteURL...").text
#Creates a BeautifulSoup object
soup = BeautifulSoup(page, "html.parser")

# Pulls all instances of <a> tag
artists = soup.find_all('a')

#Clears data of all tags
for artist in artists:
    names = artist.contents[0]
    fullLink = artist.get('href')
    print(names)
    print(fullLink)
```

1. Let's see how to apply BeautifulSoup to a webpage.
2. To scrape a webpage we also need the Requests library.

Overview Python Program

```
import requests
from bs4 import BeautifulSoup

page = requests.get("http://EnterWebsiteURL...").text
#Creates a BeautifulSoup object
soup = BeautifulSoup(page, "html.parser")

# Pulls all instances of <a> tag
artists = soup.find_all('a')

#Clears data of all tags
for artist in artists:
    names = artist.contents[0]
    fullLink = artist.get('href')
    print(names)
    print(fullLink)
```

1. The first step is to import the modules that are needed.

Overview Python Program

```
import requests
from bs4 import BeautifulSoup

page = requests.get("http://EnterWebsiteURL...").text
#Creates a BeautifulSoup object
soup = BeautifulSoup(page, "html.parser")
# Pulls all instances of <a> tag
artists = soup.find_all('a')
#Clears data of all tags
for artist in artists:
    names = artist.contents[0]
    fullLink = artist.get('href')
    print(names)
    print(fullLink)
```

2. Use the get method from the requests library to download the webpage. The input is the URL. Use the text attribute to get the text and assign it to the variable page.

Overview Python Program

```
import requests
from bs4 import BeautifulSoup

page = requests.get("http://EnterWebsiteURL...").text
#Creates a BeautifulSoup object
soup = BeautifulSoup(page, "html.parser")
# Pulls all instances of <a> tag
artists = soup.find_all('a')
#Clears data of all tags
for artist in artists:
    names = artist.contents[0]
    fullLink = artist.get('href')
    print(names)
    print(fullLink)
```

3. Then, create a BeautifulSoup object 'soup' from the variable page. It will allow you to parse through the HTML page. It will allow you to parse through the HTML page. You can now scrape the Page.



Web Scraping Lab

Estimated time needed: 30 minutes

Objectives

After completing this lab you will be able to:

Table of Contents

- Beautiful Soup Object
 - Tag
 - Children, Parents, and Siblings
 - HTML Attributes
 - Navigable String
- Filter
 - findAll
 - find
 - HTML Attributes
 - Navigable String
- Downloading And Scraping The Contents Of A Web

Estimated time needed: **25 min**

For this lab, we are going to be using Python and several Python libraries. Some of these libraries might be installed in your lab environment or in SN Labs. Others may need to be installed by you. The cells below will install these libraries when executed.

```
In [1]: !pip install bs4  
#!/usr/bin/env python3
```

Collecting bs4

 Downloading
 https://files.pythonhosted.org/packages/10/ed/7e8b97591f6f456174139ec089c769f89a94a1a4025fe967691de971f314/bs4-0.0.1.tar.gz

Collecting beautifulsoup4 (from bs4)

 Downloading
 https://files.pythonhosted.org/packages/d1/41/e6495bd7d3781cee623ce23ea6ac73282a373088fc0ddc809a047b18eae/beautifulsoup4-4.9.3-py3-none-any.whl (115kB)

[██████████] 122kB 18.8MB/s eta 0:00:01

Collecting soupsieve>1.2; python_version >= "3.0" (from beautifulsoup4->bs4)

```
Downloading  
https://files.pythonhosted.org/packages/36/69/d82d04022f02733bf9a72bc3b96332d360c0c5307096d76f6bb7489f7e57/soupsieve-2.2.1-py3-none-any.whl  
Building wheels for collected packages: bs4  
  Building wheel for bs4 (setup.py) ... done  
    Stored in directory:  
    /home/jupyterlab/.cache/pip/wheels/a0/b0/b2/4f80b9456b87abedbc0bf2d52235414c3467d8889be38dd472  
Successfully built bs4  
Installing collected packages: soupsieve, beautifulsoup4, bs4  
Successfully installed beautifulsoup4-4.9.3 bs4-0.0.1 soupsieve-2.2.1  
Import the required modules and functions
```

```
In [2]:  
from bs4 import BeautifulSoup # this module helps in web scrapping.  
import requests # this module helps us to download a web page
```

Beautiful Soup Objects

Beautiful Soup is a Python library for pulling data out of HTML and XML files, we will focus on HTML files. This is accomplished by representing the HTML as a set of objects with methods used to parse the HTML. We can navigate the HTML as a tree and/or filter out what we are looking for.

Consider the following HTML:

```
In [4]:  
%%html  
<!DOCTYPE html>  
<html>  
<head>  
<title>Page Title</title>  
</head>  
<body>  
<h3><b id='boldest'>Lebron James</b></h3>  
<p> Salary: $ 92,000,000 </p>  
<h3> Stephen Curry</h3>  
<p> Salary: $85,000,000 </p>  
<h3> Kevin Durant </h3>  
<p> Salary: $73,200,000</p>  
</body>  
</html>
```

Lebron James
Salary: \$ 92,000,000

Stephen Curry
Salary: \$85,000,000

Kevin Durant
Salary: \$73,200,000

We can store it as a string in the variable HTML:

```
In [5]: html = "<!DOCTYPE html><html><head><title>Page Title</title></head><body><h3><b id='boldest'>Lebron James</b></h3><p> Salary: $ 92,000,000 </p><h3> Stephen Curry</h3><h3> Kevin Durant</h3></body></html>"
```

To parse a document, pass it into the BeautifulSoup constructor, the BeautifulSoup object, which represents the document as a nested data structure:

```
In [6]: soup = BeautifulSoup(html, 'html5lib')
```

First, the document is converted to Unicode, (similar to ASCII), and HTML entities are converted to Unicode characters. Beautiful Soup transforms a complex HTML document into a complex tree of Python objects. The BeautifulSoup object can create other types of objects. In this lab, we will cover BeautifulSoup and Tag objects that for the purposes of this lab are identical, and NavigableString objects.

We can use the method prettify() to display the HTML in the nested structure:

```
In [7]: print(soup.prettify())
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>
      Page Title
    </title>
  </head>
  <body>
    <h3>
      <b id="boldest">
        Lebron James
      </b>
    </h3>
    <p>
      Salary: $ 92,000,000
    </p>
    <h3>
      Stephen Curry
    </h3>
    <p>
      Salary: $85,000, 000
    </p>
    <h3>
      Kevin Durant
    </h3>
    <p>
      Salary: $73,200, 000
    </p>
  </body>
</html>
```

Tags

Let's say we want the title of the page and the name of the top paid player we can use the Tag. The Tag object corresponds to an HTML tag in the original document, for example, the tag title.

```
In [8]: tag_object=soup.title
print("tag object:",tag_object)
```

```
tag object: <title>Page Title</title>
```

```
we can see the tag type bs4.element.Tag
```

```
In [9]: print("tag object type:",type(tag_object))
```

```
tag object type: <class 'bs4.element.Tag'>
```

If there is more than one Tag with the same name, the first element with that Tag name is called, this corresponds to the most paid player:

```
In [10]: tag_object=soup.h3  
tag_object  
  
Out[10]: <h3><b id="boldest">Lebron James</b></h3>  
- - - - -
```

Enclosed in the bold attribute b, it helps to use the tree representation. We can navigate down the tree using the child attribute to get the name.

Children, Parents, and Siblings

As stated above the Tag object is a tree of objects we can access the child of the tag or navigate down the branch as follows:

```
In [11]: tag_child =tag_object.b  
tag_child  
  
Out[11]: <b id="boldest">Lebron James</b>
```

You can access the parent with the parent

```
In [12]: parent_tag=tag_child.parent  
parent_tag  
  
Out[12]: <h3><b id="boldest">Lebron James</b></h3>
```

this is identical to

```
In [13]: tag_object  
  
Out[13]: <h3><b id="boldest">Lebron James</b></h3>  
  
tag_object parent is the body element.
```

```
In [14]: tag_object.parent  
  
Out[14]: <body><h3><b id="boldest">Lebron James</b></h3><p> Salary: $ 92,000,000 </p><h3> Stephen  
Curry</h3><p> Salary: $85,000, 000 </p><h3> Kevin Durant </h3><p> Salary: $73,200, 000</p></body>
```

```
In [15]: sibling_1=tag_object.next_sibling  
sibling_1
```

```
Out[15]: <p> Salary: $ 92,000,000 </p>
```

sibling_2 is the header element which is also a sibling of both sibling_1 and tag_object

```
In [16]: sibling_2=sibling_1.next_sibling  
sibling_2
```

```
Out[16]: <h3> Stephen Curry</h3>
```

Exercise: next_sibling

Using the object sibling_2 and the method next_sibling to find the salary of Stephen Curry:

```
In [17]: sibling_2.next_sibling
```

```
Out[17]: <p> Salary: $85,000, 000 </p>
```

[Click here for the solution](#) ````sibling_2.next_sibling````

HTML Attributes

If the tag has attributes, the tag id="boldest" has an attribute id whose value is boldest. You can access a tag's attributes by treating the tag like a dictionary:

```
In [18]: tag_child['id']
```

```
Out[18]: 'boldest'
```

You can access that dictionary directly as attrs :

```
In [19]: tag_child.attrs
```

```
Out[19]: {'id': 'boldest'}
```

You can also work with Multi-valued attribute check out [\[1\]](#) for more.

We can also obtain the content if the attribute of the tag using the Python get() method.

```
In [19]: tag_child.attrs
```

```
Out[19]: {'id': 'boldest'}
```

Navigable String

A string corresponds to a bit of text or content within a tag. Beautiful Soup uses the NavigableString class to contain this text. In our HTML we can obtain the name of the first player by extracting the sting of the Tag object tag_child as follows:

```
In [21]: tag_string=tag_child.string  
tag_string
```

```
Out[21]: 'Lebron James'
```

we can verify the type is Navigable String

```
In [22]: type(tag_string)
```

```
Out[22]: bs4.element.NavigableString
```

A NavigableString is just like a Python string or Unicode string, to be more precise. The main difference is that it also supports some BeautifulSoup features. We can covert it to sting object in Python:

```
In [23]: unicode_string = str(tag_string)  
unicode_string
```

```
Out[23]: 'Lebron James'
```

Filter

Filters allow you to find complex patterns, the simplest filter is a string. In this section we will pass a string to a different filter method and Beautiful Soup will perform a match against that exact string. Consider the following HTML of rocket launches:

```
In [24]:
```

```
%%html


|                                                                                                                    |                                                                                               |             |
|--------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------|-------------|
| Launch site</td> <td>Payload mass&lt;/td&gt; </td>                                                                 | Payload mass</td>                                                                             |             |
| 1</td> <td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a>&lt;/td&gt; <td>300 kg&lt;/td&gt; </td></td> | <a href="https://en.wikipedia.org/wiki/Florida">Florida</a> </td> <td>300 kg&lt;/td&gt; </td> | 300 kg</td> |
| 2</td> <td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a>&lt;/td&gt; <td>94 kg&lt;/td&gt; </td></td>      | <a href="https://en.wikipedia.org/wiki/Texas">Texas</a> </td> <td>94 kg&lt;/td&gt; </td>      | 94 kg</td>  |
| 3</td> <td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a>&lt;/td&gt; <td>80 kg&lt;/td&gt; </td></td>  | <a href="https://en.wikipedia.org/wiki/Florida">Florida</a> </td> <td>80 kg&lt;/td&gt; </td>  | 80 kg</td>  |


```

| Flight No | Launch site | Payload mass |
|-----------|---|--------------|
| 1 | Florida | 300 kg |
| 2 | Texas | 94 kg |
| 3 | Florida | 80 kg |

We can store it as a string in the variable `table`:

```
In [25]:
```

```
table=<table><tr><td id='flight'>Flight No</td><td>Launch site</td> <td>Payload mass</td></tr><tr> <td>1</td><td><a href='https://en.wikipedia.org/wik
```

```
In [26]:
```

```
table_bs = BeautifulSoup(table, 'html5lib')
```

find All

The `find_all()` method looks through a tag's descendants and retrieves all descendants that match your filters.

The Method signature for `find_all(name, attrs, recursive, string, limit, **kwargs)`

Name

When we set the name parameter to a tag name, the method will extract all the tags with that name and its children.

```
In [38]: print(type(first_row))
```

```
<class 'bs4.element.Tag'>
```

```
In [39]: print(type(second_row))
```

```
<class 'bs4.element.Tag'>
```

we can obtain the child

```
In [40]: first_row.td
```

```
Out[40]: <td id="flight">Flight No</td>
```

```
In [41]: second_row.td
```

```
Out[41]: <td>1</td>
```

If we iterate through the list, each element corresponds to a row in the table:

```
In [42]: for i,row in enumerate(table_rows):
    print("row",i,"is",row)
```

```
In [27]: table_rows=table_bs.find_all('tr')
table_rows
```

```
Out[27]: [<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>,
<tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a></a></td><td>300 kg</td></tr>,
<tr><td>2</td><td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg</td></tr>,
<tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a> </a></td><td>80 kg</td></tr>]
```

```
In [34]: table_bs.find_all('a')
```

```
Out[34]: [<a href="https://en.wikipedia.org/wiki/Florida">Florida</a>,
<a></a>,
<a href="https://en.wikipedia.org/wiki/Texas">Texas</a>,
<a href="https://en.wikipedia.org/wiki/Florida">Florida</a>,
<a> </a>]
```

The result is a Python Iterable just like a list, each element is a `tag` object:

```
In [35]: first_row =table_rows[0]
first_row
```

```
Out[35]: <tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>
```

```
In [37]: second_row = table_rows[1]
second_row
```

```
Out[37]: <tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a></a></td><td>300 kg</td></tr>
```

The type is `tag`

```
In [38]: print(type(first_row))
```

```
<class 'bs4.element.Tag'>
```

```
In [38]: print(type(first_row))
```

```
<class 'bs4.element.Tag'>
```

```
In [39]: print(type(second_row))
```

```
<class 'bs4.element.Tag'>
```

we can obtain the child

```
In [40]: first_row.td
```

```
Out[40]: <td id="flight">Flight No</td>
```

```
In [41]: second_row.td
```

```
Out[41]: <td>1</td>
```

If we iterate through the list, each element corresponds to a row in the table:

```
In [42]: for i,row in enumerate(table_rows):
print("row",i,"is",row)
```

```
row 0 is <tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>
row 1 is <tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a></a></td><td>300
kg</td></tr>
row 2 is <tr><td>2</td><td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg</td></tr>
row 3 is <tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a> </a></td><td>80
kg</td></tr>
```

As row is a cell object, we can apply the method find_all to it and extract table cells in the object cells using the tag td, this is all the children with the name td. The result is a list, each element corresponds to a cell and is a Tag object, we can iterate through this list as well. We can extract the content using the string attribute.

```
In [43]:  
for i, row in enumerate(table_rows):  
    print("row", i)  
    cells = row.find_all('td')  
    for j, cell in enumerate(cells):  
        print('column', j, "cell", cell)  
  
row 0  
column 0 cell <td id="flight">Flight No</td>  
column 1 cell <td>Launch site</td>  
column 2 cell <td>Payload mass</td>  
row 1  
column 0 cell <td>1</td>  
column 1 cell <td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a></a></td>  
column 2 cell <td>300 kg</td>  
row 2  
column 0 cell <td>2</td>  
column 1 cell <td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td>  
column 2 cell <td>94 kg</td>  
row 3  
column 0 cell <td>3</td>  
column 1 cell <td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a> </a></td>  
column 2 cell <td>80 kg</td>
```

If we use a list we can match against any item in that list.

```
In [45]:  
list_input=table_bs.find_all(name=["tr", "td"])
```

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

Attributes

If the argument is not recognized it will be turned into a filter on the tag's attributes. For example the id argument, Beautiful Soup will filter against each tag's id attribute. For example, the first td elements have a value of id of flight, therefore we can filter based on that id value.

```
In [46]: table_bs.find_all(id="flight")
```

```
Out[46]: [<td id="flight">Flight No</td>]
```

We can find all the elements that have links to the Florida Wikipedia page:

```
In [47]: list_input=table_bs.find_all(href="https://en.wikipedia.org/wiki/Florida")
list_input
```

```
Out[47]: [<a href="https://en.wikipedia.org/wiki/Florida">Florida</a>,
           <a href="https://en.wikipedia.org/wiki/Florida">Florida</a>]
```

If we set the href attribute to True, regardless of what the value is, the code finds all tags with href value:

```
In [48]: table_bs.find_all(href=True)
```

```
Out[48]: [<a href="https://en.wikipedia.org/wiki/Florida">Florida</a>,
           <a href="https://en.wikipedia.org/wiki/Texas">Texas</a>,
           <a href="https://en.wikipedia.org/wiki/Florida">Florida</a>]
```

There are other methods for dealing with attributes and other related methods; Check out the following link

Exercise: find_all

Using the logic above, find all the elements without href value

```
In [49]: table_bs.find_all(href=False)
```

```

Out[49]: [<html><head></head><body><table><tbody><tr><td id="flight">Flight No</td><td>Launch site</td><td>Payload mass</td></tr><tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg</td></tr><tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>80 kg</td></tr></tbody></table></body></html>,
<tbody><tr><td id="flight">Flight No</td><td>Launch site</td><td>Payload mass</td></tr><tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg</td></tr><tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>80 kg</td></tr></tbody></table>,
<tbody><tr><td id="flight">Flight No</td><td>Launch site</td><td>Payload mass</td></tr><tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg</td></tr><tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>80 kg</td></tr></tbody>,
<tbody><tr><td id="flight">Flight No</td><td>Launch site</td><td>Payload mass</td></tr><tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg</td></tr><tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>80 kg</td></tr></tbody>,
<td id="flight">Flight No</td>, <td>Launch site</td>, <td>Payload mass</td>, <tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>300 kg</td></tr><tr><td>2</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg</td></tr><tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>80 kg</td></tr></tbody></table>,
<tbody><tr><td>2</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg</td></tr><tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida</a><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>80 kg</td></tr></tbody>,
<td>2</td>, <td>3</td>, <td>80 kg</td>]

```

Click here for the solution [``` table_bs.find_all\(href=False\) ```](#)

Using the soup object `soup`, find the element with the `id` attribute content set to "boldest".

In [51]: `soup.find_all(id='boldest')`

Out[51]: `[<b id="boldest">Lebron James]`

Click here for the solution [``` soup.find_all\(id="boldest"\) ```](#)

string

With string you can search for strings instead of tags, where we find all the elements with Florida:

In [52]: `table_bs.find_all(string="Florida")`

Out[52]: `['Florida', 'Florida']`

find

The `find_all()` method scans the entire document looking for results, it's if you are looking for one element you can use the `find()` method to find the first element in the document. Consider the following two table:

```
In [3]: %%html
<h3>Rocket Launch </h3>

<p>
<table class='rocket'>
  <tr>
    <td>Flight No</td>
    <td>Launch site</td>
    <td>Payload mass</td>
  </tr>
  <tr>
    <td>1</td>
    <td>Florida</td>
    <td>300 kg</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Texas</td>
    <td>94 kg</td>
  </tr>
  <tr>
    <td>3</td>
    <td>Florida </td>
    <td>80 kg</td>
  </tr>
</table>
</p>
<p>

<h3>Pizza Party </h3>
```

```
<table class='pizza'>
  <tr>
    <td>Pizza Place</td>
    <td>Orders</td>
    <td>Slices </td>
  </tr>
  <tr>
    <td>Domino's Pizza</td>
    <td>10</td>
    <td>100</td>
  </tr>
  <tr>
    <td>Little Caesars</td>
    <td>12</td>
    <td>144 </td>
  </tr>
  <tr>
    <td>Papa John's </td>
    <td>15 </td>
    <td>165</td>
  </tr>
```

Rocket Launch

| Flight No | Launch site | Payload mass |
|-----------|-------------|--------------|
| 1 | Florida | 300 kg |
| 2 | Texas | 94 kg |
| 3 | Florida | 80 kg |

Pizza Party

We store the HTML as a Python string and assign `two_tables`:

```
In [53]: two_tables=<h3>Rocket Launch </h3><p><table class='rocket'><tr><td>Flight No</td><td>Launch site</td> <td>Payload mass</td></tr><tr><td>1</td><td>Florida</td><td>300 kg</td></tr><tr><td>2</td><td>Texas</td><td>94 kg</td></tr><tr><td>3</td><td>Florida </td><td>80 kg</td></tr></table></p><p>h3>Pizza Party </h3><table class='pizza'><tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></tr><tr><td>Domino's Pizza</td><td>10</td><td>100</td></tr><tr><td>Little Caesars</td><td>12</td><td>144 </td></tr><tr><td>Papa John's </td><td>15 </td><td>165</td></tr>"
```

We create a `BeautifulSoup` object `two_tables_bs`

```
In [54]: two_tables_bs= BeautifulSoup(two_tables, 'html.parser')
```

We can find the first table using the tag name `table`

```
In [55]: two_tables_bs.find("table")
```

```
<table class="rocket"><tr><td>Flight No</td><td>Launch site</td> <td>Payload mass</td></tr><tr><td>1</td><td>Florida</td><td>300 kg</td></tr><tr><td>2</td><td>Texas</td><td>94 kg</td></tr><tr><td>3</td><td>Florida </td><td>80 kg</td></tr></table>
```

We can filter on the class attribute to find the second table, but because `class` is a keyword in Python, we add an underscore.

```
In [56]: two_tables_bs.find("table",class_="pizza")
```

```
<table class="pizza"><tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></tr><tr><td>Domino's Pizza</td><td>10</td><td>100</td></tr><tr><td>Little Caesars</td><td>12</td><td>144 </td></tr><tr><td>Papa John's </td><td>15 </td><td>165</td></tr>"
```

Downloading And Scraping The Contents Of A Web Page

We Download the contents of the web page:

```
In [57]: url = "http://www.ibm.com"
```

We use `get` to download the contents of the webpage in text format and store in a variable called `data`:

```
In [58]: data = requests.get(url).text
```

We create a `BeautifulSoup` object using the `BeautifulSoup` constructor

```
In [59]: soup = BeautifulSoup(data, "html5lib") # create a soup object using the variable 'data'
```

Scrape all links

```
In [60]: for link in soup.find_all('a', href=True): # in html anchor/link is represented by the tag <a>
    print(link.get('href'))
```

```

#main-content
http://www.ibm.com
https://www.ibm.com/cloud/paks?lnk=ushpv18l1
https://www.ibm.com/cloud/strategy?lnk=ushpv18f1
https://www.ibm.com/blogs/business-partners/ibm-announces-new-benefits/?lnk=ushpv18f2
https://www.ibm.com/analytics/data-and-ai-forum?lnk=ushpv18f3
https://www.ibm.com/thought-leadership/institute-business-value/report/cloud-security-cyber-resilience?lnk=ushpv18f4
https://www.ibm.com/products/offers-and-discounts?link=ushpv18t5&lnk2=trial_mktpl_MPDISC
https://www.ibm.com/cloud/watson-assistant?lnk=ushpv18t1&lnk2=trial_WatAssist&psrc=none&pexp=def
https://www.ibm.com/cloud/watson-discovery?lnk=ushpv18t2&lnk2=trial_WatDiscovery&psrc=none&pexp=def
https://www.ibm.com/products/cloud-pak-for-data?lnk=ushpv18t3&lnk2=trial_CloudPakData&psrc=none&pexp=def
https://www.ibm.com/products/hosted-security-intelligence?lnk=ushpv18t4&lnk2=trial_QRadarCloud&psrc=none&pexp=def
https://www.ibm.com/search?lnk=ushpv18srch&locale=en-us&q=
https://www.ibm.com/products?lnk=ushpv18p1&lnk2=trial_mktpl&psrc=none&pexp=def
https://developer.ibm.com/depmodels/cloud/?lnk=ushpv18ct16
https://developer.ibm.com/technologies/artificial-intelligence?lnk=ushpv18ct19
https://www.ibm.com/demos?lnk=ushpv18ct12
https://developer.ibm.com/?lnk=ushpv18ct9
https://www.ibm.com/docs/en?lnk=ushpv18ct14
https://www.redbooks.ibm.com/?lnk=ushpv18ct10
https://www.ibm.com/support/home/?lnk=ushpv18ct11
https://www.ibm.com/training/?lnk=ushpv18ct15
https://www.ibm.com/cloud/hybrid?lnk=ushpv18ct20
https://www.ibm.com/cloud/learn/public-cloud?lnk=ushpv18ct17
https://www.ibm.com/cloud/redhat?lnk=ushpv18ct13
https://www.ibm.com/artificial-intelligence?lnk=ushpv18ct3
https://www.ibm.com/quantum-computing?lnk=ushpv18ct18
https://www.ibm.com/cloud/learn/kubernetes?lnk=ushpv18ct8
https://www.ibm.com/products/spss-statistics?lnk=ushpv18ct7
https://www.ibm.com/blockchain?lnk=ushpv18ct1
https://www-03.ibm.com/employment/technicaltalent/developer/?lnk=ushpv18ct2
https://www.ibm.com/search?lnk=ushpv18srch&locale=en-us&q=
https://www.ibm.com/products?lnk=ushpv18p1&lnk2=trial_mktpl&psrc=none&pexp=def
https://www.ibm.com/cloud/hybrid?lnk=ushpv18pt14&bv=true
https://www.ibm.com/watson?lnk=ushpv18pt17&bv=true
https://www.ibm.com/us-en/products/categories?technologyTopics[0][0]=cat.topic:Blockchain&isIBMOffering[0]=true&lnk=ushpv18pt4&bv=true
https://www.ibm.com/us-en/products/category/technology/analytics?lnk=ushpv18pt1&bv=true
https://www.ibm.com/financing?lnk=ushpv18pt3&bv=true
https://www.ibm.com/cloud/public?lnk=ushpv18pt15&bv=true
https://www.ibm.com/garage?lnk=ushpv18pt13&bv=true
https://www.ibm.com/thought-leadership/institute-business-value/?lnk=ushpv18pt12&bv=true
https://www.ibm.com/us-en/products/category/technology/security?lnk=ushpv18pt9&bv=true
https://www.ibm.com/quantum-computing?lnk=ushpv18pt16&bv=true
https://www.ibm.com/cloud/hybrid?lnk=ushpv18ct20
https://www.ibm.com/cloud/public?lnk=ushpv18ct17
https://www.ibm.com/cloud/redhat?lnk=ushpv18ct13
https://www.ibm.com/artificial-intelligence?lnk=ushpv18ct3
https://www.ibm.com/quantum-computing?lnk=ushpv18ct18
https://www.ibm.com/cloud/learn/kubernetes?lnk=ushpv18ct8
https://www.ibm.com/products/spss-statistics?lnk=ushpv18ct7
https://www.ibm.com/blockchain?lnk=ushpv18ct1
https://www-03.ibm.com/employment/technicaltalent/developer/?lnk=ushpv18ct2
https://www.ibm.com/

```

Scrape all images Tags

```
In [61]: for link in soup.find_all('img'): # in html image is represented by the tag <img>
    print(link)
    print(link.get('src'))
```


data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iMTA1NSIgaGvpZ2h0PSI1MjcuNSIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=" =

https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/u1/g/ce/a9/20210517-ls-cloud-paks-25904-720x360.jpg

data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjMyMCigeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/u1/g/1b/17/20210517-Cloud-strategy-consulting-services-b-25901-444x320.jpg

data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjMyMCigeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/u1/g/5d/9c/20210517-partnerworld-ecosystem-444x320.jpg

data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjMyMCigeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/u1/g/39/32/20210517-IBV-Cloud-Security-25813-444x320.jpg

data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCigeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
data:image/gif;base64,R0lGODlhAQABAIAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7
data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCigeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
data:image/gif;base64,R0lGODlhAQABAIAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7
data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCigeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
data:image/gif;base64,R0lGODlhAQABAIAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7
data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCigeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
data:image/gif;base64,R0lGODlhAQABAIAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7

Scrape data from HTML tables

```
In [62]: #The below url contains an html table with data about colors and color codes.  
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-SkillsNetwork/labs/datasets/HTMLColorCodes.html"
```

Before proceeding to scrape a web site, you need to examine the contents, and the way data is organized on the website. Open the above url in your browser and check how many rows and columns are there in the color table.

```
In [63]: # get the contents of the webpage in text format and store in a variable called data  
data = requests.get(url).text
```

```
In [64]: soup = BeautifulSoup(data,"html5lib")
```

```
In [67]: #find a html table in the web page  
table = soup.find('table') # in html table is represented by the tag <table>  
table
```

```
<table border="1" class="main-table">  
  <tbody><tr>  
    <td>Number </td>  
    <td>Color</td>  
    <td>Color Name</td>  
    <td>Hex Code<br/>#RRGGBB</td>  
    <td>Decimal Code<br/>(R,G,B)</td>  
  </tr>  
  <tr>  
    <td>1</td>  
    <td style="background:lightsalmon;"> </td>  
    <td>lightsalmon</td>  
    <td>#FFA07A</td>  
    <td>rgb(255,160,122)</td>  
  </tr>  
  <tr>  
    <td>2</td>  
    <td style="background:salmon;"> </td>  
    <td>salmon</td>  
    <td>#FA8072</td>  
    <td>rgb(250,128,114)</td>  
  </tr>  
  <tr>  
    <td>3</td>  
    <td style="background:darksalmon;"> </td>  
    <td>darksalmon</td>  
    <td>#E9967A</td>  
    <td>rgb(233,150,122)</td>  
  </tr>  
  <tr>  
    <td>4</td>  
    <td style="background:lightcoral;"> </td>  
    <td>lightcoral</td>  
    <td>#F08080</td>  
    <td>rgb(240,128,128)</td>  
  </tr>  
  <tr>  
    <td>5</td>  
    <td style="background:coral;"> </td>  
    <td>coral</td>  
    <td>#FF7F50</td>  
    <td>rgb(255,127,80)</td>
```

```
<tr>
    <td>6</td>
    <td style="background:tomato;"> </td>
    <td>tomato</td>
    <td>#FF6347</td>
    <td>rgb(255,99,71)</td>
</tr>
<tr>
    <td>7</td>
    <td style="background:orangered;"> </td>
    <td>orangered</td>
    <td>#FF4500</td>
    <td>rgb(255,69,0)</td>
</tr>
<tr>
    <td>8</td>
    <td style="background:gold;"> </td>
    <td>gold</td>
    <td>#FFD700</td>
    <td>rgb(255,215,0)</td>
</tr>
<tr>
    <td>9</td>
    <td style="background:orange;"> </td>
    <td>orange</td>
    <td>#FFA500</td>
    <td>rgb(255,165,0)</td>
</tr>
<tr>
    <td>10</td>
    <td style="background:darkorange;"> </td>
    <td>darkorange</td>
    <td>#FF8C00</td>
    <td>rgb(255,140,0)</td>
</tr>
<tr>
    <td>11</td>
    <td style="background:lightyellow;"> </td>
    <td>lightyellow</td>
    <td>#FFFFE0</td>
    <td>rgb(255,255,224)</td>
</tr>
<tr>
```

```

</tr>
<tr>
    <td>12</td>
    <td style="background:lemonchiffon;"> </td>
    <td>lemonchiffon</td>
    <td>#FFFACD</td>
    <td>rgb(255,250,205)</td>
</tr>
<tr>
    <td>13</td>
    <td style="background:papayawhip;"> </td>
    <td>papayawhip</td>
    <td>#FFEFDD</td>
    <td>rgb(255,239,213)</td>
</tr>
<tr>
    <td>14</td>
    <td style="background:moccasin;"> </td>
    <td>moccasin</td>
    <td>#FFE4B5</td>
    <td>rgb(255,228,181)</td>
</tr>
<tr>
    <td>15</td>
    <td style="background:peachpuff;"> </td>
    <td>peachpuff</td>
    <td>#FFDAB9</td>
    <td>rgb(255,218,185)</td>
</tr>
<tr>
    <td>16</td>
    <td style="background:palegoldenrod;"> </td>
    <td>palegoldenrod</td>
    <td>#EEE8AA</td>
    <td>rgb(238,232,170)</td>
</tr>
<tr>
    <td>17</td>
    <td style="background:khaki;"> </td>
    <td>khaki</td>
    <td>#F0E68C</td>
    <td>rgb(240,230,140)</td>
</tr>
<tr>
    <td>18</td>
    <td style="background:darkkhaki;"> </td>
    <td>darkkhaki</td>
    <td>#BDB76B</td>
    <td>rgb(189,183,107)</td>
</tr>
<tr>

```

```
</tr>
<tr>
    <td>19</td>
    <td style="background:yellow;"> </td>
    <td>yellow</td>
    <td>#FFFF00</td>
    <td>rgb(255,255,0)</td>
</tr>
<tr>
    <td>20</td>
    <td style="background:lawngreen;"> </td>
    <td>lawngreen</td>
    <td>#7CFC00</td>
    <td>rgb(124,252,0)</td>
</tr>
<tr>
    <td>21</td>
    <td style="background:chartreuse;"> </td>
    <td>chartreuse</td>
    <td>#7FFF00</td>
    <td>rgb(127,255,0)</td>
</tr>
<tr>
    <td>22</td>
    <td style="background:limegreen;"> </td>
    <td>limegreen</td>
    <td>#32CD32</td>
    <td>rgb(50,205,50)</td>
</tr>
<tr>
    <td>23</td>
    <td style="background:lime;"> </td>
    <td>lime</td>
    <td>#00FF00</td>
    <td>rgb(0.255.0)</td>
</tr>
<tr>
    <td>24</td>
    <td style="background:forestgreen;"> </td>
    <td>forestgreen</td>
    <td>#228B22</td>
    <td>rgb(34,139,34)</td>
</tr>
<tr>
    <td>25</td>
    <td style="background:green;"> </td>
    <td>green</td>
    <td>#008000</td>
    <td>rgb(0,128,0)</td>
</tr>
<tr>
    <td>26</td>
```

```
// ...  
<tr>  
    <td>26</td>  
    <td style="background:powderblue;"> </td>  
    <td>powderblue</td>  
    <td>#B0E0E6</td>  
    <td>rgb(176,224,230)</td>  
</tr>  
<tr>  
    <td>27</td>  
    <td style="background:lightblue;"> </td>  
    <td>lightblue</td>  
    <td>#ADD8E6</td>  
    <td>rgb(173,216,230)</td>  
</tr>  
<tr>  
    <td>28</td>  
    <td style="background:lightskyblue;"> </td>  
    <td>lightskyblue</td>  
    <td>#87CEFA</td>  
    <td>rgb(135,206,250)</td>  
</tr>  
<tr>  
    <td>29</td>  
    <td style="background:skyblue;"> </td>  
    <td>skyblue</td>  
    <td>#87CEEB</td>  
    <td>rgb(135,206,235)</td>  
</tr>  
<tr>  
    <td>30</td>  
    <td style="background:deepskyblue;"> </td>  
    <td>deepskyblue</td>  
    <td>#00BFFF</td>  
    <td>rgb(0,191,255)</td>  
</tr>  
<tr>  
    <td>31</td>  
    <td style="background:lightsteelblue;"> </td>  
    <td>lightsteelblue</td>  
    <td>#B0C4DE</td>  
    <td>rgb(176,196,222)</td>  
</tr>  
<tr>  
    <td>32</td>  
    <td style="background:dodgerblue;"> </td>  
    <td>dodgerblue</td>  
    <td>#1E90FF</td>  
    <td>rgb(30,144,255)</td>  
</tr>  
</tbody></table>
```

```
In [68]: #Get all rows from the table
for row in table.find_all('tr'): # in html table row is represented by the tag <tr>
    # Get all columns in each row.
    cols = row.find_all('td') # in html a column is represented by the tag <td>
    color_name = cols[2].string # store the value in column 3 as color_name
    color_code = cols[3].string # store the value in column 4 as color_code
    color_rgb = cols[4].string # store the value in column 5 as color_rgb
    print("{}--->{}--->{}".format(color_name,color_code,color_rgb))
```

Color Name--->None--->None
lightsalmon--->#FFA07A--->rgb(255,160,122)
salmon--->#FA8072--->rgb(250,128,114)
darksalmon--->#E9967A--->rgb(233,150,122)
lightcoral--->#F08080--->rgb(240,128,128)
coral--->#FF7F50--->rgb(255,127,80)
tomato--->#FF6347--->rgb(255,99,71)
orangered--->#FF4500--->rgb(255,69,0)
gold--->#FFD700--->rgb(255,215,0)
orange--->#FFA500--->rgb(255,165,0)
darkorange--->#FF8C00--->rgb(255,140,0)
lightyellow--->#FFFFE0--->rgb(255,255,224)
lemonchiffon--->#FFFACD--->rgb(255,250,205)
papayawhip--->#FFEFD5--->rgb(255,239,213)
moccasin--->#FFE4B5--->rgb(255,228,181)
peachpuff--->#FFDAB9--->rgb(255,218,185)
palegoldenrod--->#EEE8AA--->rgb(238,232,170)
khaki--->#F0E68C--->rgb(240,230,140)
darkkhaki--->#BDB76B--->rgb(189,183,107)
yellow--->#FFFF00--->rgb(255,255,0)
lawngreen--->#7CFC00--->rgb(124,252,0)
chartreuse--->#7FFF00--->rgb(127,255,0)
limegreen--->#32CD32--->rgb(50,205,50)
lime--->#00FF00--->rgb(0,255,0)
forestgreen--->#228B22--->rgb(34,139,34)
green--->#008000--->rgb(0,128,0)
powderblue--->#B0E0E6--->rgb(176,224,230)
lightblue--->#ADD8E6--->rgb(173,216,230)
lightskyblue--->#87CEFA--->rgb(135,206,250)
skyblue--->#87CEEB--->rgb(135,206,235)
deepskyblue--->#00BFFF--->rgb(0,191,255)
lightsteelblue--->#B0C4DE--->rgb(176,196,222)
dodgerblue--->#1E90FF--->rgb(30,144,255)

Scrape data from HTML tables into a DataFrame using BeautifulSoup and Pandas

```
In [69]: import pandas as pd
In [70]: #The below url contains html tables with data about world population.
url = "https://en.wikipedia.org/wiki/World_population"
Before proceeding to scrape a web site, you need to examine the contents, and the way data is organized on the website. Open the above url in your browser and check the tables on the webpage.
In [71]: # get the contents of the webpage in text format and store in a variable called data
data = requests.get(url).text
In [72]: soup = BeautifulSoup(data,"html5lib")
In [76]: #find all html tables in the web page
tables = soup.find_all('table') # in html table is represented by the tag <table>
In [74]: # we can see how many tables were found by checking the Length of the tables list
len(tables)
```

26

Assume that we are looking for the 10 most densely populated countries table, we can look through the tables list and find the right one we are looking for based on the data in each table or we can search for the table name if it is in the table but this option might not always work.

```
In [88]: for index,table in enumerate(tables):
    if ("10 most densely populated countries" in str(table)):
        table_index = index
print(table_index)
1
See if you can locate the table name of the table, 10 most densely populated countries , below.
In [78]: print(tables[table_index].prettify())
```

```

<table class="wikitable sortable" style="text-align:right">
<caption>
10 most densely populated countries
<small>
(with population above 5 million)
</small>
</caption>
<tbody>
<tr>
<th>
Rank
</th>
<th>
Country
</th>
<th>
Population
</th>
<th>
Area
<br/>
<small>
(km
<sup>
2
</sup>
)
</small>
</th>
<th>
Density
<br/>
<small>
(pop/km
<sup>
2
</sup>
)
</small>
</th>
</tr>
<tr>
<td>
1
</td>
<td align="left">
<span class="flagicon">

```

```

In [83]: population_data = pd.DataFrame(columns=["Rank", "Country", "Population", "Area", "Density"])

for row in tables[table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col != []):
        rank = col[0].text
        country = col[1].text
        population = col[2].text.strip()
        area = col[3].text.strip()
        density = col[4].text.strip()
        population_data = population_data.append({"Rank":rank, "Country":country, "Population":population, "Area":area, "Density":density}, ignore_index=True)

population_data

```

| Pizza Place | Orders | Slices | | |
|----------------|-------------|---------------|-----------|---------|
| Domino's Pizza | 10 | 100 | | |
| Little Caesars | 12 | 144 | | |
| Papa John's | 15 | 165 | | |
| Rank | Country | Population | Area | Density |
| 0 | Singapore | 5,704,000 | 710 | 8,033 |
| 1 | Bangladesh | 170,710,000 | 143,998 | 1,185 |
| 2 | Lebanon | 6,856,000 | 10,452 | 656 |
| 3 | Taiwan | 23,604,000 | 36,193 | 652 |
| 4 | South Korea | 51,781,000 | 99,538 | 520 |
| 5 | Rwanda | 12,374,000 | 26,338 | 470 |
| 6 | Haiti | 11,578,000 | 27,065 | 428 |
| 7 | Netherlands | 17,600,000 | 41,526 | 424 |
| 8 | Israel | 9,350,000 | 22,072 | 424 |
| 9 | India | 1,377,240,000 | 3,287,240 | 419 |

```
In [89]: for index, table in enumerate(tables):
    if ("Population by continent (2020 estimates)" in str(table)):
        table_index = index
print(table_index)
```

1

```
In [90]: population_continent = pd.DataFrame(columns=["Continent", "Density", "Population", "Most populous country", "Most Populous city"])

for row in tables[table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col != []):
        continent = col[0].text
        density = col[1].text
        population = col[2].text.strip()
        most_pop_country = col[3].text.strip()
        most_pop_city = col[4].text.strip()
        population_continent = population_continent.append({"Continent":continent, "Density":density, "Population":population, "Most populous country":most_pop_country, "Most Populous city":most_pop_city})

population_continent
```

| | Continent | Density | Population | Most populous country | Most Populous city |
|---|----------------------------|---------|---|--|---|
| 0 | Asia\n | 104.1\n | 4,641 | 1,439,323,000[note 1] – China | 37,393,000/13,929,000 – Greater Tokyo Area/Tokyo |
| 1 | Africa\n | 44.4\n | 1,340 | 0206,139,000 – Nigeria | 20,900,000 – Cairo[17] |
| 2 | Europe\n | 73.4\n | 747 0145,934,000 – Russia; approx. 110 million in ... | 16,855,000/12,537,000 – Moscow metropolitan area | |
| 3 | Latin America\n | 24.1\n | 653 | 0212,559,000 – Brazil | 22,043,000/12,176,000 – São Paulo Metro Area/Belo Horizonte |
| 4 | Northern America[note 2]\n | 14.9\n | 368 | 0331,002,000 – United States | 23,724,000/8,323,000 – New York metropolitan area |
| 5 | Oceania\n | 5\n | 42 | 0025,499,000 – Australia | 4,925,000 – Sydney |
| 6 | Antarctica\n | ~0\n | 0.004[16] | N/A[note 3] | 1,258 – McMurdo Station |

Scrape data from HTML tables into a DataFrame using BeautifulSoup and read_html

Using the same url, data, soup, and tables object as in the last section we can use the read_html function to create a DataFrame.

Remember the table we need is located in tables[table_index]

We can now use the pandas function read_html and give it the string version of the table as well as the flavor which is the parsing engine bs4.

```
In [84]: pd.read_html(str(tables[5]), flavor='bs4')
```

```
Out[84]: [   Rank      Country  Population  Area(km2)  Density(pop/km2)
0     1  Singapore    5704000       710        8033
1     2  Bangladesh  170710000     143998       1185
2     3    Lebanon   6856000       10452        656
3     4    Taiwan   23604000      36193        652
4     5  South Korea  51781000      99538        520
5     6    Rwanda   12374000      26338        470
6     7    Haiti    11578000      27065        428
7     8  Netherlands  17600000      41526        424
8     9    Israel    9350000      22072        424
9    10    India   1377240000     3287240        419]
```

The function `read_html` always returns a list of DataFrames so we must pick the one we want out of the list.

```
In [85]: population_data_read_html = pd.read_html(str(tables[5]), flavor='bs4')[0]
population_data_read_html
```

```
Out[85]:   Rank      Country  Population  Area(km2)  Density(pop/km2)
0     1  Singapore    5704000       710        8033
1     2  Bangladesh  170710000     143998       1185
2     3    Lebanon   6856000       10452        656
3     4    Taiwan   23604000      36193        652
4     5  South Korea  51781000      99538        520
5     6    Rwanda   12374000      26338        470
6     7    Haiti    11578000      27065        428
7     8  Netherlands  17600000      41526        424
8     9    Israel    9350000      22072        424
9    10    India   1377240000     3287240        419
```

```
In [91]: pd.read_html(str(tables[1]), flavor='bs4')
```

```

Out[91]: [
    Continent Density(inhabitants/km2) Population(millions) \
0           Asia          104.1        4641
1           Africa         44.4        1340
2           Europe          73.4        747
3           Latin America      24.1        653
4 Northern America[note 2]      14.9        368
5           Oceania            5          42
6           Antarctica       ~0     0.004[16]

    Most populous country \
0 1,439,323,000[note 1] - China
1 0206,139,000 - Nigeria
2 0145,934,000 - Russia;approx. 110 million in E...
3 0212,559,000 - Brazil
4 0331,002,000 - United States
5 0025,499,000 - Australia
6 N/A[note 3]

    Most populous city (metropolitan area)
0 37,393,000/13,929,000 - Greater Tokyo Area/Tok...
1 20,900,000 - Cairo[17]
2 16,855,000/12,537,000 - Moscow metropolitan ar...
3 22,043,000/12,176,000 - São Paulo Metro Area/S...
4 23,724,000/8,323,000 - New York metropolitan a...
5 4,925,000 - Sydney
6 1,258 - McMurdo Station ]

```

```

In [93]: population_continent_read_html = pd.read_html(str(tables[1]), flavor='bs4')[0]

population_continent_read_html

```

| | Continent | Density(inhabitants/km2) | Population(millions) | Most populous country | Most populous city (metropolitan area) |
|---|--------------------------|--------------------------|----------------------|---|---|
| 0 | Asia | 104.1 | 4641 | 1,439,323,000[note 1] - China | 37,393,000/13,929,000 – Greater Tokyo Area/Tok... |
| 1 | Africa | 44.4 | 1340 | 0206,139,000 - Nigeria | 20,900,000 – Cairo[17] |
| 2 | Europe | 73.4 | 747 | 0145,934,000 - Russia;approx. 110 million in E... | 16,855,000/12,537,000 – Moscow metropolitan ar... |
| 3 | Latin America | 24.1 | 653 | 0212,559,000 - Brazil | 22,043,000/12,176,000 – São Paulo Metro Area/S... |
| 4 | Northern America[note 2] | 14.9 | 368 | 0331,002,000 – United States | 23,724,000/8,323,000 – New York metropolitan a... |
| 5 | Oceania | 5 | 42 | 0025,499,000 - Australia | 4,925,000 – Sydney |
| 6 | Antarctica | ~0 | 0.004[16] | N/A[note 3] | 1,258 – McMurdo Station |

```

In [94]: population_milestones_read_html = pd.read_html(str(tables[2]), flavor='bs4')[0]

population_milestones_read_html

```

Out[94]:

World population milestones in billions (Worldometers estimates)

| Population | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|------------|---------------|------|------|------|------|------|------|------|------|------|------|
| 0 | Year | 1804 | 1927 | 1960 | 1974 | 1987 | 1999 | 2011 | 2023 | 2037 | 2057 |
| 1 | Years elapsed | — | 123 | 33 | 14 | 13 | 12 | 12 | 12 | 14 | 20 |

Scrape data from HTML tables into a DataFrame using read_html

We can also use the `read_html` function to directly get DataFrames from a `url`.

In [95]:

```
dataframe_list = pd.read_html(url, flavor='bs4')
```

We can see there are 25 DataFrames just like when we used `find_all` on the `soup` object.

In [96]:

```
len(dataframe_list)
```

Out[96]:

26

Finally we can pick the DataFrame we need out of the list.

In [97]:

```
dataframe_list[5]
```

We can also use the `match` parameter to select the specific table we want. If the table contains a string matching the text it will be read.

In [98]:

```
pd.read_html(url, match="10 most densely populated countries", flavor='bs4')[0]
```

Out[98]:

| | Rank | Country | Population | Area(km2) | Density(pop/km2) |
|---|------|-------------|------------|-----------|------------------|
| 0 | 1 | Singapore | 5704000 | 710 | 8033 |
| 1 | 2 | Bangladesh | 170710000 | 143998 | 1185 |
| 2 | 3 | Lebanon | 6856000 | 10452 | 656 |
| 3 | 4 | Taiwan | 23604000 | 36193 | 652 |
| 4 | 5 | South Korea | 51781000 | 99538 | 520 |
| 5 | 6 | Rwanda | 12374000 | 26338 | 470 |
| 6 | 7 | Haiti | 11578000 | 27065 | 428 |
| 7 | 8 | Netherlands | 17600000 | 41526 | 424 |
| 8 | 9 | Israel | 9350000 | 22072 | 424 |
| 9 | 10 | India | 1377240000 | 3287240 | 419 |

In [99]:

```
dataframe_list[1]
```

| Out[99]: | Continent | Density(inhabitants/km2) | Population(millions) | Most populous country | Most populous city (metropolitan area) |
|----------|--------------------------|--------------------------|----------------------|--|---|
| 0 | Asia | 104.1 | 4641 | 1,439,323,000[note 1] – China | 37,393,000/13,929,000 – Greater Tokyo Area/Tok.. |
| 1 | Africa | 44.4 | 1340 | 0206,139,000 – Nigeria | 20,900,000 – Cairo[17] |
| 2 | Europe | 73.4 | 747 | 0145,934,000 – Russia; approx. 110 million in E... | 16,855,000/12,537,000 – Moscow metropolitan ar... |
| 3 | Latin America | 24.1 | 653 | 0212,559,000 – Brazil | 22,043,000/12,176,000 – São Paulo Metro Area/S... |
| 4 | Northern America[note 2] | 14.9 | 368 | 0331,002,000 – United States | 23,724,000/8,323,000 – New York metropolitan a... |
| 5 | Oceania | 5 | 42 | 0025,499,000 – Australia | 4,925,000 – Sydney |
| 6 | Antarctica | ~0 | 0.004[16] | N/A[note 3] | 1,258 – McMurdo Station |

4.5 APIs, and Data Collection

Seongjoo Brenden Song edited this page on May 25, 2021 · 7 revisions

This module delves into the unique ways to collect data by the use of APIs and webscraping. It further explores data collection by explaining how to read and collect data when dealing with different file formats.

Learning Objectives

- Explain how the URL Request Response HTTP protocol works.
- Explain the use of the HTTP protocol using the Requests Library method.
- Practice the basics of webscraping in JupyterLab.
- Practice working with different file formats in JupyterLab.
- Work with different file formats

-
- [Simple APIs](#)
 - [Hands-On Lab: Introduction to API](#)
-

- [REST APIs & HTTP Requests](#)
- [Hands-on Lab: Access REST APIs & Request HTTP](#)
- [Webscraping](#)
- [Hands-on Lab: Webscraping](#)
- [Working with different file formats \(csv, xml, json, xlsx\)](#)
- [Hands-on Lab: Working with different file formats](#)
- [Practice Quiz & Graded Quiz: REST APIs, Webscraping, and Working with Files](#)

Working with different file formats (csv, xml, json, xlsx)

Hello. Welcome to Working With Different File Formats

After watching this video, you will be able to:

Define different file formats such as `csv`, `xml`, and `json`

Write simple programs to read and output data What Python libraries are needed to extract data

When collecting data you will find there are many different file formats that need to be collected or read in order to complete a data driven story or analysis.

When gathering the data Python can make the process simpler with its predefined libraries, but before we explore Python let's first check out some of the various file formats. When looking at a file name you will notice an extension at the end of the title. **These extensions let you know what type of file it is and what it needed to open it.** For instance if you see a title like

`"FileExample.csv"` you will know this is a "csv" file. This is only one example of different file types as there are many more such as "json" or "xml". When coming across these different file formats and trying to access their data we need to utilize Python libraries to make this process easier. **The first Python library to become familiar with is called Pandas.** By importing this library in the beginning of the code we are then be able to easily read the different file types.

Since we have now imported the Panda library let's use it to read the first "csv" file.

In this instance we have come across the `"FileExample.csv"` file. The first step is to assign the file to a variable. Then create another variable to read the file with the help of the Panda library. We can then call `read_csv` function output the data to the screen.

With this example there were no headers for the data so it added the first line as the header.

Since we don't want the first line of data as the header let's find out how to correct this issue.

Now that we have learned how to read and output the data from a "csv" file let's make it look a little more organized. From the last example we were able to print out the data but because the file had no headers it printed the first line of data as a header.

We easily solve this by adding a `dataframe` attribute. We use the variable `"df"` to call the file and then add the `"columns"` attribute. By adding this one line to our program we can then neatly organize the data output into the specified headers for each column.

The next file format we will explore is the "json" file format. In this type of file the text is written in a language independent data format and is similar to a Python dictionary. The first step in reading this type of file is to import `json`. After importing "json" we can add a line to open the file call the `"load"` attribute of "json" to begin and read the file and lastly we can then

print the file. The next file format is "xml". This type of file is also known as **Extensible Markup Language**. While the Pandas library does not have an attribute to read this type of file let's explore how to parse this type of file. **The first step to read this type of file is to import xml.** By importing this library we can then use the "etree" attribute to parse the "xml" file. We then add the column headers and assign them to the dataframe. Then create a loop to go through the document to collect the necessary data and append the data to a dataframe.

In this video, you learned: How to recognize different file types
How to use Python libraries to extract data How to use dataframes when collecting data

Working With Different File Formats

Objectives

After watching this video, you will be able to:

- Define different file formats such as csv, xml, and json
- Write simple programs to read and output data
- List what Python libraries are needed to extract data

Introduction



Understanding File Formats

FileExample.csv
FileExample.json
FileExample.xml

Python Pandas Library

Pandas Library

import pandas as pd

The first Python library to become familiar with is called Pandas. By importing this library in the beginning of the code we are then be able to easily read the different file types.

Reading CSV Files

```
import pandas as pd
file = "FileExample.csv"
df = pd.read_csv(file)
```

In this instance we have come across the "FileExample.csv" file. The first step is to assign the file to a variable. Then create another variable to read the file with the help of the Panda library. We can then call read_csv function output the data to the screen.

Reading CSV Files

| | | |
|-------------|--------------|-----------|
| Bill Smith | 258-541-2598 | 5/25/1984 |
| 1 Jerod Rue | 857-968-8542 | 12/5/1986 |
| 2 Kris Ann | 875-256-1452 | 5/17/1978 |

With this example there were no headers for the data so

it added the first line as the header.

To correct and add a header:

Using Dataframes

| | | | |
|---|------------|--------------|-----------|
| | Bill Smith | 258-541-2598 | 5/25/1984 |
| 1 | Jerod Rue | 857-968-8542 | 12/5/1986 |
| 2 | Kris Ann | 875-256-1452 | 5/17/1978 |

Using Dataframes

```
df.columns = ['Name', 'Phone  
Number', 'Birthday']
```

So:

Using Dataframes

| | Name | Phone Number | Birthday |
|---|------------|--------------|-----------|
| 1 | Bill Smith | 258-541-2598 | 5/25/1984 |
| 2 | Jerod Rue | 857-968-8542 | 12/5/1986 |
| 3 | Kris Ann | 875-256-1452 | 5/17/1978 |

Reading JSON Files

```
import json  
  
with open('filesample.json', 'r') as openfile:  
    json_object = json.load(openfile)  
  
print(json_object)
```

The first step in reading this type of file is to import json. After importing "json" we can add a line to open the file call the "load" attribute of "json" to begin and read the file and lastly we can then print the file.

Reading XML Files

```
import pandas as pd
import xml.etree.ElementTree as etree
tree = etree.parse("fileExample.xml")
root = tree.getroot()
columns = ["Name", "Phone Number", "Birthday"]
df = pd.DataFrame(columns = columns)
```

The next file format is “xml”. This type of file is also known as **Extensible Markup Language**. While the Pandas library does not have an attribute to read this type of file let’s explore how to parse this type of file. The first step to read this type of file is to import `xml`. By importing this library we can then use the “`etree`” attribute to parse the “`xml`” file. We then add the column headers and assign them to the dataframe.

Reading XML Files

```
for node in root:
    name = node.find("name").text
    phonenumber = node.find("phonenumber").text
    birthday = node.find("birthday").text
    df = df.append(pd.Series([name, phonenumber,
    birthday], index = columns)
    ...., ignore_index = True)
```

We then add the column headers and assign them to the dataframe. Then create a loop to go through the document to collect the necessary data and append the data to a dataframe.

Summary

In this video, you learned:

- How to recognize different file types
- How to use Python libraries to extract data
- How to use dataframes when collecting data

Table of Contents

- 1. Data Engineering
- 2. Data Engineering Process
- 3. Working with different file formats
- 4. Data Analysis

Data Engineering

Data engineering is one of the most critical and foundational skills in any data scientist's toolkit.

Data Engineering Process

There are several steps in Data Engineering process.

1. **Extract** :- Data extraction is getting data from multiple sources. Ex. Data extraction from a website using Web scraping or gathering information from the data that are stored in different formats(JSON, CSV, XLSX etc.).
2. **Transform** :- Transforming the data means removing the data that we don't need for further analysis and converting the data in the format that all the data from the multiple sources is in the same format.
3. **Load** :- Loading the data inside a data warehouse. Data warehouse essentially contains large volumes of data that are accessed to gather insights.

Working with different file formats

In real-world, people rarely get neat tabular data. Thus, it is mandatory for any data scientist (or a data engineer) to be aware of different file formats, common challenges in handling them and the best / efficient ways to handle this data in real life. We have reviewed some of this content in other modules.

File Format

A file format is a standard way in which information is encoded for storage in a file. First, the file format specifies whether the file is a binary or ASCII file. Second, it shows how the information is organized. For example, comma-separated values (CSV) file format stores tabular data in plain text.

To identify a file format, you can usually look at the file extension to get an idea. For example, a file saved with name "Data" in "CSV" format will appear as "Data.csv". By noticing ".csv" extension we can clearly identify that it is a "CSV" file and data is stored in a tabular format.

There are various formats for a dataset, .csv, .json, .xlsx etc. The dataset can be stored in different places, on your local machine or sometimes online.

In this section, you will learn how to load a dataset into our Jupyter Notebook.

Now, we will look at some file formats and how to read them in Python:

Comma-separated values(CSV) file format

Comma-separated values file format falls under spreadsheet file format.

In spreadsheet file format, data is stored in cells. Each cell is organized in rows and columns. A column in the spreadsheet file can have different types. For example, a column can be of string type, a date type or an integer type.

Each line in CSV file represents an observation or commonly called a record. Each record may contain one or more fields which are separated by a comma.

Reading the data from CSV in Python

The **Pandas** Library is a useful tool that enables us to read various datasets into a data frame

Let us look at how to read a CSV file in Pandas Library.

We use **pandas.read_csv()** function to read the csv file. In the bracket, we put the file path along with a quotation mark, so that pandas will read the file into a data frame from that address. The file path can be either an URL or your local file address

In [25]: `import pandas as pd`

In [26]: `url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/addressess.csv'`
`df = pd.read_csv(url)`

In [27]: `df`

Out[27]:

| | John | Doe | 120 jefferson st. | Riverside | NJ | 08075 | |
|---|-----------------------|----------|----------------------------------|-------------|----------|-------|-----|
| 0 | Jack | McGinnis | 220 hobo Av. | Phila | PA | 9119 | |
| 1 | John "Da Man" | Repici | 120 Jefferson St. | Riverside | NJ | 8075 | |
| 2 | Stephen | Tyler | 7452 Terrace "At the Plaza" road | SomeTown | SD | 91234 | |
| 3 | NaN | Blankman | | NaN | SomeTown | SD | 298 |
| 4 | Joan "the bone", Anne | Jet | 9th, at Terrace plc | Desert City | CO | 123 | |

Adding column name to the DataFrame

We can add columns to an existing DataFrame using its **columns** attribute.

In [28]:

```
df.columns = ['First Name', 'Last Name', 'Location', 'City', 'State', 'Area Code']
```

In [29]:

```
df
```

Out[29]:

| | First Name | Last Name | Location | City | State | Area Code | |
|---|-----------------------|-----------|----------------------------------|-------------|----------|-----------|-----|
| 0 | Jack | McGinnis | 220 hobo Av. | Phila | PA | 9119 | |
| 1 | John "Da Man" | Repici | 120 Jefferson St. | Riverside | NJ | 8075 | |
| 2 | Stephen | Tyler | 7452 Terrace "At the Plaza" road | SomeTown | SD | 91234 | |
| 3 | NaN | Blankman | | NaN | SomeTown | SD | 298 |
| 4 | Joan "the bone", Anne | Jet | 9th, at Terrace plc | Desert City | CO | 123 | |

Selecting a single column

To select the first column 'First Name', you can pass the column name as a string to the indexing operator.

```
In [7]: df["First Name"]
```

```
Out[7]: 0          Jack
1      John "Da Man"
2          Stephen
3            NaN
4  Joan "the bone", Anne
Name: First Name, dtype: object
```

```
In [8]: df['Last Name']
```

```
Out[8]: 0    McGinnis
1      Repici
2        Tyler
3  Blankman
4        Jet
Name: Last Name, dtype: object
```

```
In [9]: df['City']
```

```
Out[9]: 0      Phila
1  Riverside
2   SomeTown
3   SomeTown
4  Desert City
Name: City, dtype: object
```

```
In [9]: df['City']
```

```
Out[9]: 0      Phila
1  Riverside
2   SomeTown
3   SomeTown
4  Desert City
Name: City, dtype: object
```

Selecting multiple columns

To select multiple columns, you can pass a list of column names to the indexing operator.

```
In [30]: df = df[['First Name', 'Last Name', 'Location', 'City', 'State', 'Area Code']]
df
```

```
Out[30]:
```

| | First Name | Last Name | Location | City | State | Area Code |
|---|-----------------------|-----------|----------------------------------|-------------|-------|-----------|
| 0 | Jack | McGinnis | 220 hobo Av. | Phila | PA | 9119 |
| 1 | John "Da Man" | Repici | 120 Jefferson St. | Riverside | NJ | 8075 |
| 2 | Stephen | Tyler | 7452 Terrace "At the Plaza" road | SomeTown | SD | 91234 |
| 3 | NaN | Blankman | NaN | SomeTown | SD | 298 |
| 4 | Joan "the bone", Anne | Jet | 9th, at Terrace plc | Desert City | CO | 123 |

Selecting rows using .iloc and .loc

Now, let's see how to use .loc for selecting rows from our DataFrame.

loc() : loc() is label based data selecting method which means that we have to pass the name of the row or column which we want to select.

```
In [31]: # To select the first row  
df.loc[0]
```

```
Out[31]: First Name      Jack  
          Last Name     McGinnis  
          Location      220 hobo Av.  
          City           Phila  
          State          PA  
          Area Code      9119  
          Name: 0, dtype: object
```

```
In [32]: # To select the 0th,1st and 2nd row of "First Name" column only  
df.loc[[0,1,2], "First Name"]
```

```
Out[32]: 0      Jack  
1  John "Da Man"  
2      Stephen  
Name: First Name, dtype: object
```

```
In [35]: df.loc[[1,2,3], 'City']
```

```
Out[35]: 1    Riverside  
2    SomeTown  
3    SomeTown  
Name: City, dtype: object
```

```
In [55]: df.loc[1:3]
```

Out[55]:

| | First Name | Last Name | Location | City | State | Area Code | |
|---|---------------|-----------|----------------------------------|-----------|----------|-----------|-----|
| 1 | John "Da Man" | Repici | 120 Jefferson St. | Riverside | NJ | 8075 | |
| 2 | Stephen | Tyler | 7452 Terrace "At the Plaza" road | SomeTown | SD | 91234 | |
| 3 | NaN | Blankman | | NaN | SomeTown | SD | 298 |

In [56]:

```
df.loc[:2]
```

Out[56]:

| | First Name | Last Name | Location | City | State | Area Code |
|---|---------------|-----------|----------------------------------|-----------|-------|-----------|
| 0 | Jack | McGinnis | 220 hobo Av. | Phila | PA | 9119 |
| 1 | John "Da Man" | Repici | 120 Jefferson St. | Riverside | NJ | 8075 |
| 2 | Stephen | Tyler | 7452 Terrace "At the Plaza" road | SomeTown | SD | 91234 |

Now, let's see how to use .iloc for selecting rows from our DataFrame.

iloc() : iloc() is a indexed based selecting method which means that we have to pass integer index in the method to select specific row/column

In [36]:

```
# To select the 0th,1st and 2nd row of "First Name" column only
df.iloc[[0,1,2], 0]
```

Out[36]:

| | |
|---|---------------|
| 0 | Jack |
| 1 | John "Da Man" |
| 2 | Stephen |

Name: First Name, dtype: object

In [38]:

```
#Index number 3 = 'City'
df.iloc[[0,1,2,3,4], 3]
```

Out[38]:

| | |
|---|-------------|
| 0 | Phila |
| 1 | Riverside |
| 2 | SomeTown |
| 3 | SomeTown |
| 4 | Desert City |

Name: City, dtype: object

In [54]:

```
df.iloc[1:3]
```

Out[54]:

| | First Name | Last Name | Location | City | State | Area Code |
|---|---------------|-----------|----------------------------------|-----------|-------|-----------|
| 1 | John "Da Man" | Repici | 120 Jefferson St. | Riverside | NJ | 8075 |
| 2 | Stephen | Tyler | 7452 Terrace "At the Plaza" road | SomeTown | SD | 91234 |

In [57]:

```
df.iloc[:2]
```

Out[57]:

| | First Name | Last Name | Location | City | State | Area Code |
|---|---------------|-----------|-------------------|-----------|-------|-----------|
| 0 | Jack | McGinnis | 220 hobo Av. | Phila | PA | 9119 |
| 1 | John "Da Man" | Repici | 120 Jefferson St. | Riverside | NJ | 8075 |

For more information please read the [documentation](#).

Let's perform some basic transformation in pandas.

Transform Function in Pandas

Python's Transform function returns a self-produced dataframe with transformed values after applying the function specified in its parameter.

Let's see how Transform function works.

In [58]:

```
#import library
import pandas as pd
import numpy as np
```

In [59]:

```
#creating a dataframe
df=pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), columns=['a', 'b', 'c'])
df
```

```
Out[59]:   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

Let's say we want to add 10 to each element in a dataframe:

```
In [60]: #applying the transform function
df = df.transform(func = lambda x : x + 10)
df
```

```
Out[60]:   a  b  c
0  11 12 13
1  14 15 16
2  17 18 19
```

```
In [66]: df_1=pd.DataFrame(np.array([[1,3,5,7],[2,4,6,8],[1,4,7,9]]), columns=['A', 'B', 'C', 'D'])
df_1
```

```
Out[66]:   A  B  C  D
0  1  3  5  7
1  2  4  6  8
2  1  4  7  9
```

```
In [69]: df_1.transform(func = lambda x : x * 2 - x / 2)
```

```
Out[69]:   A    B    C    D
0  1.5  4.5  7.5 10.5
1  3.0  6.0  9.0 12.0
2  1.5  6.0 10.5 13.5
```

Now we will use DataFrame.transform() function to find the square root to each element of the dataframe.

```
In [70]: result = df.transform(func = ['sqrt'])

In [71]: result
```

| | a | b | c |
|---|----------|----------|----------|
| | sqrt | sqrt | sqrt |
| 0 | 3.316625 | 3.464102 | 3.605551 |
| 1 | 3.741657 | 3.872983 | 4.000000 |
| 2 | 4.123106 | 4.242641 | 4.358899 |


```
In [73]: df.transform([np.sqrt])
```

| | a | b | c |
|---|----------|----------|----------|
| | sqrt | sqrt | sqrt |
| 0 | 3.316625 | 3.464102 | 3.605551 |
| 1 | 3.741657 | 3.872983 | 4.000000 |
| 2 | 4.123106 | 4.242641 | 4.358899 |


```
In [74]: df.transform('sum')
```

| | a | b | c |
|---|----|----|--------------|
| 0 | 42 | 45 | 48 |
| | | | dtype: int64 |


```
In [75]: df.transform('max')
```

| | a | b | c |
|---|----|----|--------------|
| 0 | 17 | 18 | 19 |
| | | | dtype: int64 |

For more information about the `transform()` function please read the [documentation](#).

JSON file Format

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write.

JSON is built on two structures:

1. A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
2. An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

JSON is a language-independent data format. It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data. It is a very common data format, with a diverse range of applications.

The text in JSON is done through quoted string which contains the value in key-value mapping within { }. It is similar to the dictionary in Python.

Python supports JSON through a built-in package called **json**. To use this feature, we import the json package in Python script.

Writing JSON to a File

This is usually called **serialization**. It is the process of converting an object into a special format which is suitable for transmitting over the network or storing in file or database.

To handle the data flow in a file, the JSON library in Python uses **dump()** or **dumps()** function to convert the Python objects into their respective JSON object, so it makes easy to write data to files.

```
In [76]: import json
```

```
In [77]: import json
person = {
    'first_name' : 'Mark',
    'last_name' : 'abc',
    'age' : 27,
    'address': {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    }
}
```

serialization using dump() function

json.dump() method can be used for writing to JSON file.

Syntax: `json.dump(dict, file_pointer)`

Parameters:

1. **dictionary** – name of dictionary which should be converted to JSON object.
2. **file pointer** – pointer of the file opened in write or append mode.

```
In [78]: with open('person.json', 'w') as f: # writing JSON object
    json.dump(person, f)
```

serialization using dumps() function

json.dumps() that helps in converting a dictionary to a JSON object.

It takes two parameters:

1. **dictionary** – name of dictionary which should be converted to JSON object.
2. **indent** – defines the number of units for indentation

```
In [79]: # Serializing json
json_object = json.dumps(person, indent = 4)

# Writing to sample.json
with open("sample.json", "w") as outfile:
    outfile.write(json_object)
```

```
In [80]: print(json_object)
```

```
{
    "first_name": "Mark",
    "last_name": "abc",
    "age": 27,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    }
}
```

```
In [89]: import json
person_1 = {
    'first_name' : 'Brenden',
    'last_name' : 'SJS',
    'age' : 40,
    'phone number': '123-456-7890',
    'address': {
        "streetAddress": "222 3rd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3230"
    }
}
with open("person_1.json", 'w') as f_1:
    json.dump(person_1, f_1)

json_object_1 = json.dumps(person_1, indent = 4)
with open("sample_1.json", 'w') as outfile_1:
    outfile_1.write(json_object_1)
print(json_object_1)
```

```
{
    "first_name": "Brenden",
    "last_name": "SJS",
    "age": 40,
    "phone number": "123-456-7890",
    "address": {
        "streetAddress": "222 3rd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3230"
    }
}
```

Our Python objects are now serialized to the file. To deserialize it back to the Python object we use the `load()` function.

Reading JSON to a File

This process is usually called **Deserialization**: It is the reverse of serialization. It converts the special format returned by the serialization back into a usable object.

Using `json.load()`

The JSON package has `json.load()` function that loads the json content from a json file into a dictionary.

It takes one parameter:

File pointer: A file pointer that points to a JSON file.

```
In [90]: import json

# Opening JSON file
with open('sample.json', 'r') as openfile:

    # Reading from json file
    json_object = json.load(openfile)

    print(json_object)
    print(type(json_object))

{'first_name': 'Mark', 'last_name': 'abc', 'age': 27, 'address': {'streetAddress': '21 2nd Street', 'city': 'New York', 'state': 'NY', 'postalCode': '10021-3100'}}
<class 'dict'>
```

```
In [95]: with open('sample_1.json', 'r') as openfile_1:
    json_object_1 = json.load(openfile_1)

    json_sample_object = json.dumps(json_object_1, indent = 4)

    print(json_object_1)
    print(type(json_object_1))
    print("")
    print(json_sample_object)
    print(type(json_sample_object))

{'first_name': 'Brenden', 'last_name': 'SJS', 'age': 40, 'phone number': '123-456-7890', 'address': {'streetAddress': '222 3rd Street', 'city': 'New York', 'state': 'NY', 'postalCode': '10021-3230'}}
<class 'dict'>

{
    "first_name": "Brenden",
    "last_name": "SJS",
    "age": 40,
    "phone number": "123-456-7890",
    "address": {
        "streetAddress": "222 3rd Street",
```

```
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3230"
    }
}

<class 'str'>
```

XLSX file format

XLSX is a Microsoft Excel Open XML file format. It also comes under the Spreadsheet file format.

In XLSX data is organized under the cells and columns in a sheet.

Reading the data from XLSX file

Let's load the data from XLSX file and define the sheet name. For loading the data you can use the Pandas library in python.

Reading the data from XLSX file

Let's load the data from XLSX file and define the sheet name. For loading the data you can use the Pandas library in python.

```
In [97]: import pandas as pd
```

```
In [98]: import urllib.request
urllib.request.urlretrieve("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Mo
```

```
In [99]:
```

```
df
```

```
Out[99]:
```

| | 0 | First Name | Last Name | Gender | Country | Age | Date | Id |
|---|---|------------|-----------|--------|---------------|-----|------------|------|
| 0 | 1 | Dulce | Abril | Female | United States | 32 | 15/10/2017 | 1562 |
| 1 | 2 | Mara | Hashimoto | Female | Great Britain | 25 | 16/08/2016 | 1582 |
| 2 | 3 | Philip | Gent | Male | France | 36 | 21/05/2015 | 2587 |
| 3 | 4 | Kathleen | Hanner | Female | United States | 25 | 15/10/2017 | 3549 |
| 4 | 5 | Nereida | Magwood | Female | United States | 58 | 16/08/2016 | 2468 |
| 5 | 6 | Gaston | Brumm | Male | United States | 24 | 21/05/2015 | 2554 |
| 6 | 7 | Etta | Hurn | Female | Great Britain | 56 | 15/10/2017 | 3598 |
| 7 | 8 | Earlean | Melgar | Female | United States | 27 | 16/08/2016 | 2456 |
| 8 | 9 | Vincenza | Weiland | Female | United States | 40 | 21/05/2015 | 6548 |

```
In [111...:
```

```
df.loc[[0,2,4,5], ["Age", "Gender"]]
```

```
Out[111...:
```

| | Age | Gender |
|---|-----|--------|
| 0 | 32 | Female |
| 2 | 36 | Male |
| 4 | 58 | Female |
| 5 | 24 | Male |

```
In [103...:
```

```
df.iloc[0:4]
```

```
Out[103...:
```

| | 0 | First Name | Last Name | Gender | Country | Age | Date | Id |
|---|---|------------|-----------|--------|---------------|-----|------------|------|
| 0 | 1 | Dulce | Abril | Female | United States | 32 | 15/10/2017 | 1562 |
| 1 | 2 | Mara | Hashimoto | Female | Great Britain | 25 | 16/08/2016 | 1582 |
| 2 | 3 | Philip | Gent | Male | France | 36 | 21/05/2015 | 2587 |
| 3 | 4 | Kathleen | Hanner | Female | United States | 25 | 15/10/2017 | 3549 |

XML file format

XML is also known as Extensible Markup Language. As the name suggests, it is a markup language. It has certain rules for encoding data. XML file format is a human-readable and machine-readable file format.

Pandas does not include any methods to read and write XML files. Here, we will take a look at how we can use other modules to read data from an XML file, and load it into a Pandas DataFrame.

Writing with `xml.etree.ElementTree`

The **`xml.etree.ElementTree`** module comes built-in with Python. It provides functionality for parsing and creating XML documents. ElementTree represents the XML document as a tree. We can move across the document using nodes which are elements and sub-elements of the XML file.

For more information please read the [xml.etree.ElementTree](#) documentation.

```
In [112...  
import xml.etree.ElementTree as ET  
  
# create the file structure  
employee = ET.Element('employee')  
details = ET.SubElement(employee, 'details')  
first = ET.SubElement(details, 'firstname')  
second = ET.SubElement(details, 'lastname')  
third = ET.SubElement(details, 'age')  
first.text = 'Shiv'  
second.text = 'Mishra'  
third.text = '23'  
  
# create a new XML file with the results  
mydata1 = ET.ElementTree(employee)  
# myfile = open("items2.xml", "wb")  
# myfile.write(mydata)  
with open("new_sample.xml", "wb") as files:  
    mydata1.write(files)
```

Reading with xml.etree.ElementTree

Let's have a look at a one ways to read XML data and put it in a Pandas DataFrame. You can see the XML file in the Notepad of your local machine.

```
In [113...  
import pandas as pd  
  
import xml.etree.ElementTree as etree  
  
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDriverSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/Sample-e
```

```
--2021-05-24 18:17:56-- https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDriverSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/Sample-employee-XML-file.xml  
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)... 169.45.118.108  
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud)|169.45.118.108|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 1016 [application/xml]  
Saving to: 'Sample-employee-XML-file.xml'
```

```
Sample-employee-XML 100%[=====] 1016 --.-KB/s in 0s
```

```
2021-05-24 18:17:56 (36.5 MB/s) - 'Sample-employee-XML-file.xml' saved [1016/1016]
```

You would need to firstly parse an XML file and create a list of columns for data frame. then extract useful information from the XML file and add to a pandas data frame.

Here is a sample code that you can use.:

```
In [114...]  
tree = etree.parse("Sample-employee-XML-file.xml")  
  
root = tree.getroot()  
columns = ["firstname", "lastname", "title", "division", "building", "room"]  
  
datatframe = pd.DataFrame(columns = columns)  
  
for node in root:  
  
    firstname = node.find("firstname").text  
  
    lastname = node.find("lastname").text  
  
    title = node.find("title").text  
  
    division = node.find("division").text  
  
    building = node.find("building").text  
  
    room = node.find("room").text  
  
    datatframe = datatframe.append(pd.Series([firstname, lastname, title, division, building, room], index = columns), ignore_index = True)
```

In [115...]
datatframe

| | firstname | lastname | title | division | building | room |
|---|-----------|----------|-----------|----------|----------|------|
| 0 | Shiv | Mishra | Engineer | Computer | 301 | 11 |
| 1 | Yuh | Datta | developer | Computer | 303 | 02 |
| 2 | Rahil | Khan | Tester | Computer | 304 | 10 |
| 3 | Deep | Parekh | Designer | Computer | 305 | 14 |

Save Data

Correspondingly, Pandas enables us to save the dataset to csv by using the **dataframe.to_csv()** method, you can add the file path and name along with quotation marks in the brackets.

For example, if you would save the dataframe df as **employee.csv** to your local machine, you may use the syntax below:

```
In [117...]  
datatframe.to_csv("employee.csv", index=False)
```

We can also read and save other file formats, we can use similar functions to **pd.read_csv()** and **df.to_csv()** for other data formats, the functions are listed in the following table:

Read/Save Other Data Formats

| Data Format | Read | Save |
|-------------|------------------------------|----------------------------|
| csv | <code>pd.read_csv()</code> | <code>df.to_csv()</code> |
| json | <code>pd.read_json()</code> | <code>df.to_json()</code> |
| excel | <code>pd.read_excel()</code> | <code>df.to_excel()</code> |
| hdf | <code>pd.read_hdf()</code> | <code>df.to_hdf()</code> |
| sql | <code>pd.read_sql()</code> | <code>df.to_sql()</code> |
| ... | ... | ... |

Let's move ahead and perform some **Data Analysis**.

Binary File Format

"Binary" files are any files where the format isn't made up of readable characters. It contain formatting information that only certain applications or processors can understand. While humans can read text files, binary files must be run on the appropriate software or processor before humans can read them.

Binary files can range from image files like JPEGs or GIFs, audio files like MP3s or binary document formats like Word or PDF.

Let's see how to read an **Image** file.

Reading the Image file

Python supports very powerful tools when comes to image processing. Let's see how to process the images using **PIL** library.

PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities.

```
In [118...]: # importing PIL
from PIL import Image

import urllib.request
# Downloading dataset
urllib.request.urlretrieve("https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/images/dog-puppy-on-garden-royalty-free-image-1586966191.jpg", "dog.jpg")
```



```
Out[118...]: ('dog.jpg', <http.client.HTTPMessage at 0x7f141e04c438>)
```



```
In [119...]: # Read image
img = Image.open('dog.jpg')

# Output Images
display(img)
```

Data Analysis

In this section, you will learn how to approach data acquisition in various ways, and obtain necessary insights from a dataset. By the end of this lab, you will successfully load the data into Jupyter Notebook, and gain some fundamental insights via Pandas Library.

In our case, the **Diabetes Dataset** is an online source, and it is in CSV (comma separated value) format. Let's use this dataset as an example to practice data reading.

About this Dataset

Context This dataset is originally from the **National Institute of Diabetes and Digestive and Kidney Diseases**. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Content The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

We have 768 rows and 9 columns. The first 8 columns represent the features and the last column represent the target/label.

```
In [120...]: # Import pandas Library
import pandas as pd

In [121...]: path = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/diabet...
df = pd.read_csv(path)

In [124...]: df.loc[0:3]

Out[124...]:
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome | |
|---|-------------|---------|---------------|---------------|---------|------|--------------------------|-------|---------|---|
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | | 0.167 | 21 | 0 |

After reading the dataset, we can use the **dataframe.head(n)** method to check the top n rows of the dataframe; where n is an integer. Contrary to **dataframe.head(n)**, **dataframe.tail(n)** will show you the bottom n rows of the dataframe.

```
In [123...     # show the first 5 rows using dataframe.head() method
print("The first 5 rows of the dataframe")
df.head(5)

The first 5 rows of the dataframe
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome | |
|---|-------------|---------|---------------|---------------|---------|------|--------------------------|-------|---------|---|
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | | 0.627 | 50 | 1 |
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | | 0.351 | 31 | 0 |
| 2 | 8 | 183 | 64 | 0 | 0 | 23.3 | | 0.672 | 32 | 1 |
| 3 | 1 | 89 | 66 | 23 | 94 | 28.1 | | 0.167 | 21 | 0 |
| 4 | 0 | 137 | 40 | 35 | 168 | 43.1 | | 2.288 | 33 | 1 |

```
In [142... df.loc[0, 'BMI']

Out[142... 33.6

To view the dimensions of the dataframe, we use the .shape parameter.
```

```
In [128... df.shape

Out[128... (768, 9)
```

Statistical Overview of dataset

```
In [129... df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Pregnancies      768 non-null    int64  
 1   Glucose          768 non-null    int64  
 2   BloodPressure    768 non-null    int64  
 3   SkinThickness    768 non-null    int64  
 4   Insulin          768 non-null    int64  
 5   BMI              768 non-null    float64 
 6   DiabetesPedigreeFunction 768 non-null    float64 
 7   Age              768 non-null    int64  
 8   Outcome          768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

In [130... df.describe()

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|--------------|-------------|------------|---------------|---------------|------------|------------|--------------------------|------------|------------|
| count | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| mean | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | 0.471876 | 33.240885 | 0.348958 |
| std | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 | 11.760232 | 0.476951 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 | 21.000000 | 0.000000 |
| 25% | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 | 24.000000 | 0.000000 |
| 50% | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| 75% | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| max | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

Pandas **describe()** is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values. When this method is applied to a series of string, it returns a different output

Identify and handle missing values

We use Python's built-in functions to identify these missing values. There are two methods to detect missing data:

.isnull()

.notnull()

The output is a boolean value indicating whether the value that is passed into the argument is in fact missing data.

The output is a boolean value indicating whether the value that is passed into the argument is in fact missing data.

In [131... missing_data = df.isnull()
missing_data.head(5)

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|-------------|---------|---------------|---------------|---------|-------|--------------------------|-------|---------|
| 0 | False | False | False | False | False | False | False | False | False |
| 1 | False | False | False | False | False | False | False | False | False |
| 2 | False | False | False | False | False | False | False | False | False |
| 3 | False | False | False | False | False | False | False | False | False |
| 4 | False | False | False | False | False | False | False | False | False |

"True" stands for missing value, while "False" stands for not missing value.

Count missing values in each column

Using a for loop in Python, we can quickly figure out the number of missing values in each column. As mentioned above, "True" represents a missing value, "False" means the value is present in the dataset. In the body of the for loop the method ".value_counts()" counts the number of "True" values.

```
In [132]:  
for column in missing_data.columns.values.tolist():  
    print(column)  
    print(missing_data[column].value_counts())  
    print("")  
  
Pregnancies  
False    768  
Name: Pregnancies, dtype: int64  
  
Glucose  
False    768  
Name: Glucose, dtype: int64  
  
BloodPressure  
False    768  
Name: BloodPressure, dtype: int64  
  
SkinThickness  
False    768  
Name: SkinThickness, dtype: int64  
  
Insulin  
False    768  
Name: Insulin, dtype: int64  
  
BMI  
False    768  
Name: BMI, dtype: int64  
  
DiabetesPedigreeFunction  
False    768  
Name: DiabetesPedigreeFunction, dtype: int64  
  
Age  
False    768  
Name: Age, dtype: int64  
  
Outcome  
False    768  
Name: Outcome, dtype: int64
```

As you can see above there is no missing values in the dataset.

Correct data format

Check all data is in the correct format (int, float, text or other).

In Pandas, we use

.dtype() to check the data type

.astype() to change the data type

Numerical variables should have type '**float**' or '**int**'.

```
In [133... df.dtypes
```

```
Out[133... Pregnancies          int64
Glucose            int64
BloodPressure      int64
SkinThickness      int64
Insulin            int64
BMI                float64
DiabetesPedigreeFunction float64
Age                int64
Outcome            int64
dtype: object
```

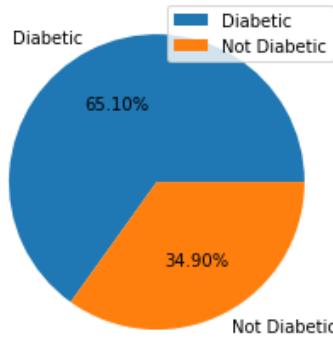
As we can see above, All columns have the correct data type.

Visualization

Visualization is one of the best way to get insights from the dataset. **Seaborn** and **Matplotlib** are two of Python's most powerful visualization libraries.

```
In [154... # import Libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

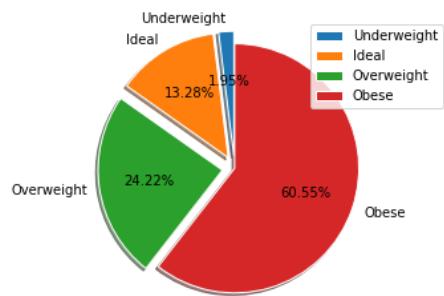
```
In [155... labels= 'Diabetic','Not Diabetic'
plt.pie(df['Outcome'].value_counts(),labels=labels,autopct='%.02f%%')
plt.legend()
plt.show()
```



As you can see above 65.10% females are Diabetic and 34.90% are Not Diabetic.

As you can see above 65.10% females are Diabetic and 34.90% are Not Diabetic.

```
In [177]:  
    labels_BMI = ['Underweight', 'Ideal', 'Overweight', 'Obese']  
  
    count_under = 0  
    count_ideal = 0  
    count_over = 0  
    count_obese = 0  
  
    i = 0  
  
    while i < df.shape[0]:  
        value = df.loc[i, 'BMI']  
  
        if value < 18.5:  
            count_under = count_under + 1  
        elif value >= 18.5 and value < 25:  
            count_ideal = count_ideal + 1  
        elif value >= 25 and value <=30:  
            count_over = count_over + 1  
        else:  
            count_obese = count_obese + 1  
  
        i = i + 1  
  
    sizes = [count_under, count_ideal, count_over, count_obese]  
    explode = (0.1, 0.1, 0.1, 0)  
  
    plt.pie(sizes, explode = explode, labels = labels_BMI,  
            autopct='%.02f%%', shadow=True, startangle=90,  
            wedgeprops = { 'linewidth' : 1, 'edgecolor' : "white" })  
    plt.axis('equal')  
    plt.legend()  
    plt.show()
```



Thank you for completing this Notebook

Practice Quiz: REST APIs, Webscraping, and Working with Files

Bookmarked

Question 1

1/1 point (ungraded)

What is the function of "GET" in HTTP requests?

- Deletes a specific resource
- Returns the response from the client to the requestor
- Sends data to create or update a resource
- Carries the request to the client from the requestor



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 2

1/1 point (ungraded)

What does URL stand for?

- Uniform Resource Learning
- Uniform Resource Locator
- Unilateral Resistance Locator
- Uniform Request Location



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 3

1/1 point (ungraded)

What does the file extension "csv" stand for?

Comma Serrated Values

Comma Separation Valuations

Common Separated Variables

Comma Separated Values



Submit

You have used 2 of 2 attempts

✓ Correct (1/1 point)

Question 4

1/1 point (ungraded)

What is webscraping?

The process to display all data within a URL.

The process to request and retrieve information from a client.

The process to describe communication options.

The process to extract data from a particular website.



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Graded Quiz: Working with Numpy Arrays

[Bookmark this page](#)

Graded Quiz due Jan 29, 2022 16:26 +08

Question 1

1/1 point (graded)

What does REST stand for?

Response Expectation Style

Response Situation Transfer

Representational Expectation State Transfer

Representational State Transfer

Response Expectation System Transformation



[Submit](#)

You have used 1 of 2 attempts

Correct (1/1 point)

Question 2

1/1 point (graded)

In what data structure do HTTP responses generally return?

Lists

Tuples

Nested Lists

JSON



[Submit](#)

You have used 1 of 2 attempts

Correct (1/1 point)

Question 3

1/1 point (graded)

What are the 3 parts to a URL?

- Block, post, and route
- Put, route, and get
- Get, post, and scheme
- Scheme, internet address, and route



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 4

1/1 point (graded)

What are the 3 parts to a response message?

- Bookmarks, history, and security
- Encoding, body, and cache
- HTTP headers, blank line, and body
- Start or status line, header, and body



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 5

1/1 point (graded)

What is the purpose of this line of code "**table_row=table.find_all(name='tr')**" used in webscraping?

- It will find all of the data within the table marked with a tag "a"
- It will find all of the data within the table marked with a tag "p"
- It will find all of the data within the table marked with a tag "h1"
- It will find all of the data within the table marked with a tag "tr"



Submit

You have used 1 of 2 attempts

✓ Correct (1/1 point)

Question 1

0/1 point (graded)

What does REST stand for?

Response Expectation Style

Response Situation Transfer

Representational Expectation State Transfer

Representational State Transfer

Response Expectation System Transformation

<https://github-wiki-see.page/m/brendensong/IBM-Data-Science-Professional-Certificate/wiki/4.4.6.Graded-Quiz-Numpy-in-Python>

4.4.6.Graded Quiz Numpy in Python - brendensong/IBM-Data-Science-Professional-Certificate Wiki

LATEST SUBMISSION GRADE 100%

Question 1

What is the result of the following lines of code?

```
a=np.array([1,1,1,1,1])
b=np.array([2,2,2,2,2])
a*b
```

- 0
- array([0, 0, 0, 0, 0])
- array([2, 2, 2, 2, 2])

Correct

Question 2

What is the result of the following lines of code?

```
a=np.array([0,1])
b=np.array([1,0])
np.dot(a,b)
```

- 0
- 1
- array([1,1])

Correct

Question 3

Is the following operation allowed with numpy arrays?

```
a=np.array([1,2,3,4], [5,6,7,8] (/brendensong/IBM-Data-Science-Professional-Certificat
a*[1,2,3,4]
```

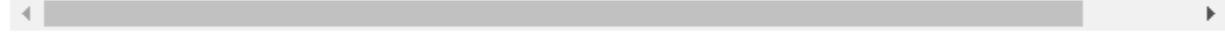
- 
- no, the two elements must be of equal size
 - yes, this is an example of broadcasting

Correct

Question 4

What is the value of Z after the following code is run?

```
X=np.array([1,0],[0,1] (/brendensong/IBM-Data-Science-Professional-Certificate/wiki/1,
Y=np.array([0,1],[1,0] (/brendensong/IBM-Data-Science-Professional-Certificate/wiki/0,
Z=X+Y
```

- 
- array([[1, 1],[1, 1]])
 - array([[1, 0],[0, 1]])
 - array([[0, 1],[1, 1]])

Correct, the '+' corresponds to matrix addition

Question 5

What values does the variable **out** take if the following lines of code are run?

```
X=np.array([1,0,1],[2,2,2]())
out=X[0,1:3]
out
```

- 
- `array([0, 1])`
 - `array([2, 2])`
 - `array([1, 0, 1])`

Correct, the first index corresponds to the rows the second index corresponds to the columns

Question 6

What is the value of **Z** after the following code is run?

```
X=np.array([1,0],[0,1]())
Y=np.array([2,2],[2,2]())
Z=np.dot(X,Y)
```

- 
- `array([[2, 2],[2, 2]])`
 - `array([[2, 0],[0, 2]])`
 - `array([[3, 2],[2, 3]])`

Correct, the dot function corresponds to matrix multiplication

Graded Quiz: REST APIs, Webscraping, and Working with Files

Question 1

What are the 3 parts to a URL?

- Get, post, and scheme
- Block, post, and route
- *Scheme, internet address, and route*
- Put, route, and get

Correct

Question 2

What are the 3 parts to a response message?

- *Start or status line, header, and body*
- Encoding, body, and cache
- Bookmarks, history, and security
- HTTP headers, blank line, and body

Correct

Question 3

What is the 404 status code?

- Unauthorized
- OK
- *Not found*
- Bad request

Correct

Question 4

What is the purpose of this line of code "`table_row=table.find_all(name='tr')`" used in webscraping?

- *It will find all of the data within the table marked with a tag "tr"*
- It will find all of the data within the table marked with a tag "a"
- It will find all of the data within the table marked with a tag "p"
- It will find all of the data within the table marked with a tag "h1"

<https://datai.co/python-interview-questions/dictionary/>

Instructions

 Bookmark this page

Exam Instructions

1. This exam is worth 50% of your entire grade for the course
2. There is no pass/fail for the exam itself, but the grade you get will affect your overall passing grade for the course
3. Time allowed: **1 hour.**
4. Attempts per question:
 - One attempt - For True/False questions
 - Two attempts - For any question other than True/False
5. Clicking the "**Final Check**" button when it appears, means your submission is **FINAL**. You will **NOT** be able to resubmit your answer for that question ever again
6. Check your grades in the course at any time by clicking on the "Progress" tab

IMPORTANT: Do not let the time run out and expect the system to grade you automatically. You must explicitly submit your answers, otherwise they would be marked as incomplete.