Python for Data Science

Learning Objectives

The objectives of this course are to get you started with Python as a programming language and to give you a taste of how to start working with data in Python.

In this course you will learn about:

- What is Python and why it is useful
- The application of Python
- How to define **variables**
- **Sets** and **conditional statements** in Python
- The purpose of having **functions** in Python
- How to operate on files to **read and write data** in Python
- How to use **pandas**, a must-have package for anyone attempting data analysis in Python

Syllabus

Module 1 - Python Basics

- Your first program
- Types
- Expressions and Variables
- String Operations

Module 2 - Python Data Structures

- Lists and Tuples
- Sets
- Dictionaries

Module 3 - Python Programming Fundamentals

- Conditions and Branching
- Loops
- Functions
- Objects and Classes

Module 4 - Working with Data in Python

- Reading files with open
- Writing files with open
- Loading data with Pandas
- Working with and Saving data with Pandas

Module 5 - Working with Numpy Arrays & Simple APIs

- Numpy 1D Arrays
- Numpy 2D Arrays
- Simple APIs
- API Setup

Grading Scheme

1. The minimum passing mark for the **course** is 70% with the following weights:
    - 50% - All Review Questions
    - 50% - The Final Exam
2. Though Review Questions and the Final Exam have a passing mark of 50% respectively, the only grade that matters is the overall grade for the **course**.
3. Review Questions have no time limit. You are encouraged to review the course material to find the answers. Please remember that the Review Questions are worth 50% of your final mark.
4. The final exam has a 1 hour time limit.
5. Attempts are per **question** in both, the Review Questions and the Final Exam:
    - One attempt - For True/False questions
    - Two attempts - For any question other than True/False
6. There are no penalties for incorrect attempts.
7. 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.

8. Check your grades in the course at any time by clicking on the "Progress" tab.

Learning Objectives

After completing this section you will:

- Understand the available data types in Python
- Be familiar with Python expressions and variables
- Be able to perform basic operations on strings

Most importantly, you will write your first bits of Python code using Jupyter notebook, the same rich environment used by thousands of data scientists, data analysts and data engineers.

Your First Program

Let's try your first program in Python. **A statement or expression is an instruction the computer will run or execute**. Perhaps the simplest program you can write is a print statement. When you run the **print statement, Python will simply display the value in the parentheses**. The value in the **parentheses is called the argument.** If you are using a Jupiter notebook in this course, you will see a small rectangle with the statement. **This is called a cell.** If you select this cell with your mouse, then click the run cell button. The statement will execute. The result will be displayed beneath the cell. We will follow this paradigm in the videos. **It's customary to comment your code**. This tells other people what your code does. **You simply put a hash symbol proceeding your comment.** When you run the code, Python will ignore the comment. **A Syntactic error is when Python does not understand your code**. For example, if you spell print "frint", you will get an error message. **A Semantic error is when your logic is wrong.** For example, if you enter Python 102 instead of Python 101, you don't get an error message, but your code is wrong.

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 type casting. You can convert an int to a float. For example, you can convert or cast the integer 2 to a float 2. 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 zero. 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.

Expressions and Variables

In this video we'll cover expressions and variables.

Expressions describe a type of operation that computers perform.

Expressions are operations that 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 maths symbols, in this case addition, are called

operators. We can perform operations such as subtraction

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 – and *.

We can also perform division with the forward slash.

25 divided by 5 is 5. 25 divided by 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 1920. 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 equals 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 do 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’s say we would like to convert the number

of minutes in the highlighted examples to number of hours in the following music dataset.

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 can also use a capital

letter. We call the variable that contains the total

number of hours total_hr. 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.

String Operations (3:53)

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 6.

Moreover, we can access the 13th index. We can also use negative indexing with strings.

The last element is given by the index -1. The first element can be obtained by index

-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 2 indicates we 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 concatenated with a new string.

The result is a new string that changes from Michael Jackson to “Michael Jackson is the

best.” Strings are immutable.

Backslashes 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 lower case characters to upper case 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 the segment changed.

The method find, finds sub-strings. The argument is the sub-string you would like

to find. The output is the first index of the sequence.

We can find the sub-string Jack. If the sub-string is not in the string, the

output is negative one. Check the labs for more examples.

Quiz:

Q.What is the result of the following line of code:

“0123456”.find(‘1’)

Answer: 1

Q. Numbers = “0123456”

How would you obtain teh even elements?

A. Numbers[::2]

Review Questions

Instructions for Review Questions

How much time do I have to complete these questions?

Unlimited. You can take as long you want to answer these questions.

Can I go back to the videos to check something, then come back to these Review Questions?

Yes, absolutely! These questions are for you to review what you've learned so far. Take your time.

Do these Review Questions count towards my final grade?

Yes, all of the review questions, combined together, are worth 50% of your total mark.

How many chances do I get to answer these questions?

It depends:

- For True/False questions, you only get one (1) chance.

- For any other question (that is not True/False), you get two (2) chances.

How can I check my overall course grade?

You can check your grades by clicking on "**Progress**" in the top menu.

Quiz:

What is the result of the following operation in Python:

3 + 2 * 2

Answer: 7

Q. In Python, if you executed name = 'Lizz', what would be the output of print(name[0:2])?

A. Li

Q. In Python, if you executed var = '01234567', what would be the result of print(var[::2])?

A. 0246

Q. In Python, what is the result of the following operation '1'+'2'?

A. ‘12’

Q. Given myvar = 'hello', how would you convert myvar into uppercase?

A. myvar.upper()

Learning Objectives

In this lesson you will learn about:

- Lists and Tuples
- Sets
- Dictionaries

Lists and Tuples (8:46)

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 1 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 1 larger than the length of the tuple.
We can use the “len” command to obtain the length of a tuple.
As there are 5 elements, the result is five.
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 tuple.
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 1 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 parenthesis.
The argument inside the parenthesis 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 lists 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 of 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 last 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.

Module 3 - Python Programming
Learning Objectives

In this lesson you will learn about:

- **Conditions and Branching**

- **Loops**
- **Functions**
- **Objects and Classes**

Video Conditions and Branching

In this video you will learn about Conditions and Branching.
Comparison operations compare some value or operand, then, based on some condition they produce a Boolean.
Let's say we assign "a" value of “a” to 6.
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 6.
In this case, as six is not equal to 7, the result is false.
If we performed an equality test for the value 6, 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, 5, the condition becomes true, or else we get a false.
Let's display some values for "i" on the left.
Let’s see the values greater than 5 in green and the rest in red.
If we set "i" equal to 6, we see that 6 is larger than 5, 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, 5, then the condition becomes true.
In this case, we include the value of 5 in the number line and the color changes to green accordingly.
If we set the value of "i" equal to 5, the operand will produce a true.
If we set the value of “i” to 2, we would get a false because 2 is less than 5.
We can change the inequality, if the value of the left operand, in this case, “i” is less than the value of right operand, in this case, 6, 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 2, the result is a true, as 2 is less than 6.
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 2, the operator is true as 2 is not equal to 6.
We compare strings as well.
Comparing “AC/DC” 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 labs 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 the 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 proceeding 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 concert.
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 they are 18 years, they attend Pink Floyd.
After seeing Pink Floyd, they move on.
The syntax of the “elseif statement” is similar.
We simply add the statement elseif 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 else if 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 labs 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 70's 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 true.
Examining the second condition, we see that 1990 is greater than 1989, so the 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.
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.

LAB

Comparison Operations

Find the value of i that produces a True:

```
i=
i!=0
 File "/tmp/ipykernel_64/2139093573.py", line 1
    i=
      ^
SyntaxError: invalid syntax
```

Click here for the solution

```
i = 1

# any value other than 0 will produce output as True
```

Branching

Find the value of x that prints the statement "this is a":

```
x=
if(x=='a'):
   print("this is a")
else:
   print("this is  not a")
 File "/tmp/ipykernel_64/2649428824.py", line 1
    x=
      ^
SyntaxError: invalid syntax
```

Click here for the solution

```
x = 'a'
```

Logic Operators

Find the value of y that produces a True statement:

```
y=
x=1
x>0 and y<10
 File "/tmp/ipykernel_64/3931790973.py", line 1
    y=
      ^
SyntaxError: invalid syntax
```

Click here for the solution

```
y = 0

# any value less than 10 will produce output as True
```

Quiz

QUESTION 1

1/1 point (ungraded)
Select the values of i that produces a True for the following:

i!=0

Answer:
1
-1
100000000

QUESTION 2

1/1 point (ungraded)
What is the output of the following:

```
x='a'

if(x!='a'):

print("This is not a.")

else:

print("This is a.")
```

Answer:
"This is a."

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

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:

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

Introduction of Pandas

```
# Dependency needed to install file

!pip install xlrd
Requirement already satisfied: xlrd in
/opt/conda/envs/Python-3.8-main/lib/python3.8/site-packages (2.0.1)
```

```
# 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.

# Read data from CSV file

csv_path =
'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%204/data/TopSellingAlbums.csv'
df = pd.read_csv(csv_path)

We can use the method head() to examine the first five rows of a dataframe:

# Print first five rows of the dataframe

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

We use the path of the excel file and the function read_excel. The result is a data frame as before:

```
# 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()
```

	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

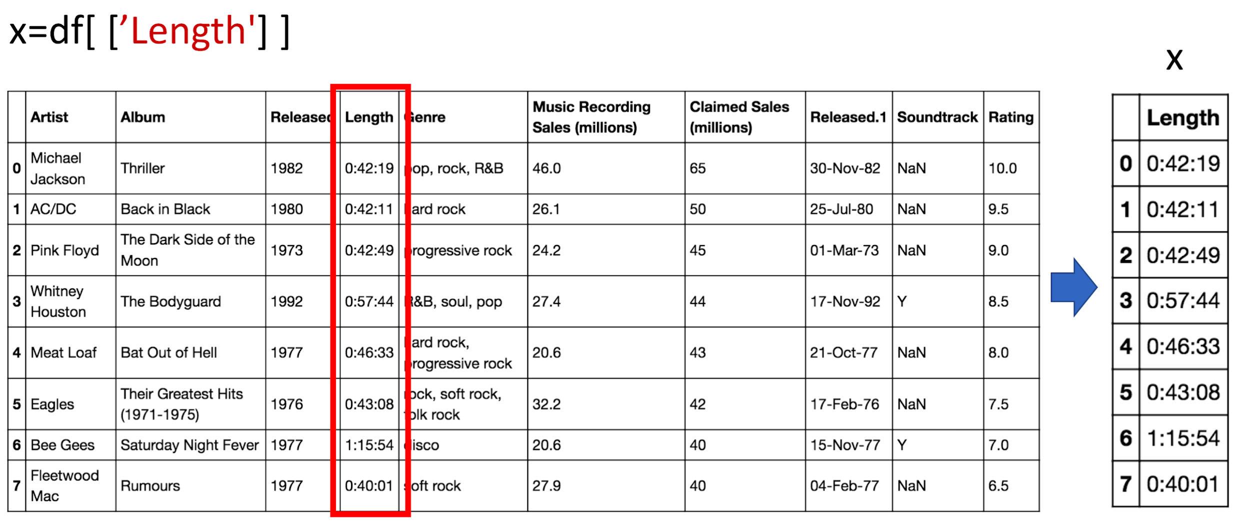
We can access the column **Length** and assign it a new dataframe **x**:

Access to the column Length

```
x = df[['Length']]
x
```

	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:



	Artist	Album	Released	Length	enre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	Rating
0	Michael Jackson	Thriller	1982	0:42:19	op, rock, R&B	46.0	65	30-Nov-82	NaN	10.0
1	AC/DC	Back in Black	1980	0:42:11	ard rock	26.1	50	25-Jul-80	NaN	9.5
2	Pink Floyd	The Dark Side of the Moon	1973	0:42:49	rogressive rock	24.2	45	01-Mar-73	NaN	9.0
3	Whitney Houston	The Bodyguard	1992	0:57:44	&B, soul, pop	27.4	44	17-Nov-92	Y	8.5
4	Meat Loaf	Bat Out of Hell	1977	0:46:33	ard rock, rogressive rock	20.6	43	21-Oct-77	NaN	8.0
5	Eagles	Their Greatest Hits (1971-1975)	1976	0:43:08	ock, soft rock, olk rock	32.2	42	17-Feb-76	NaN	7.5
6	Bee Gees	Saturday Night Fever	1977	1:15:54	isco	20.6	40	15-Nov-77	Y	7.0
7	Fleetwood Mac	Rumours	1977	0:40:01	oft rock	27.9	40	04-Feb-77	NaN	6.5

	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:

```
# Get the column as a series

x = df['Length']
x
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**:

```
# Get the column as a dataframe

x = df[['Artist']]
type(x)
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:

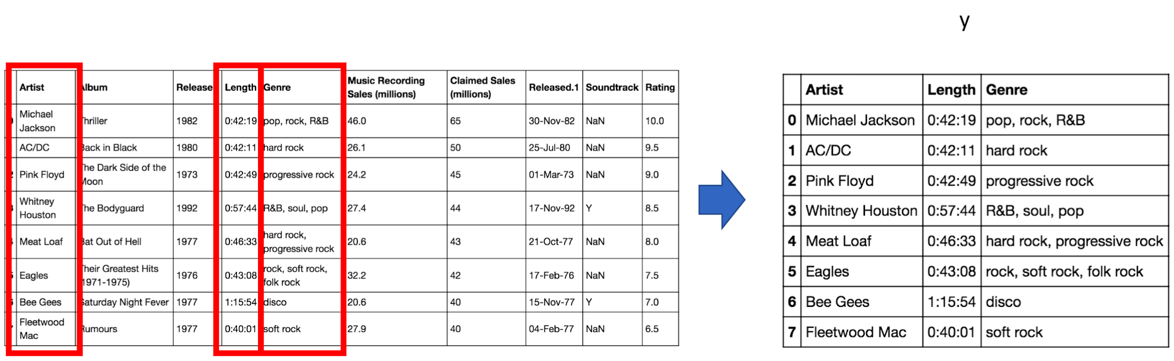
```
# Access to multiple columns

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

The process is shown in the figure:

```
y=df[ ['Artist' ,'Length', 'Genre '] ]
```



Artist	lbum	Release	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	Rating
Michael Jackson	hriller	1982	0:42:19	pop, rock, R&B	46.0	65	30-Nov-82	NaN	10.0
AC/DC	ack in Black	1980	0:42:11	hard rock	26.1	50	25-Jul-80	NaN	9.5
Pink Floyd	he Dark Side of the loon	1973	0:42:49	progressive rock	24.2	45	01-Mar-73	NaN	9.0
Whitney Houston	he Bodyguard	1992	0:57:44	R&B, soul, pop	27.4	44	17-Nov-92	Y	8.5
Meat Loaf	at Out of Hell	1977	0:46:33	hard rock, progressive rock	20.6	43	21-Oct-77	NaN	8.0
Eagles	heir Greatest Hits 1971-1975)	1976	0:43:08	rock, soft rock, folk rock	32.2	42	17-Feb-76	NaN	7.5
Bee Gees	aturday Night Fever	1977	1:15:54	disco	20.6	40	15-Nov-77	Y	7.0
Fleetwood Mac	umours	1977	0:40:01	soft rock	27.9	40	04-Feb-77	NaN	6.5

y

	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:

# Access the value on the first row and the first column

df.iloc[0, 0]
'Michael Jackson'

You can access the 2nd row and the 1st column as follows:

# Access the value on the second row and the first column

df.iloc[1,0]
'AC/DC'

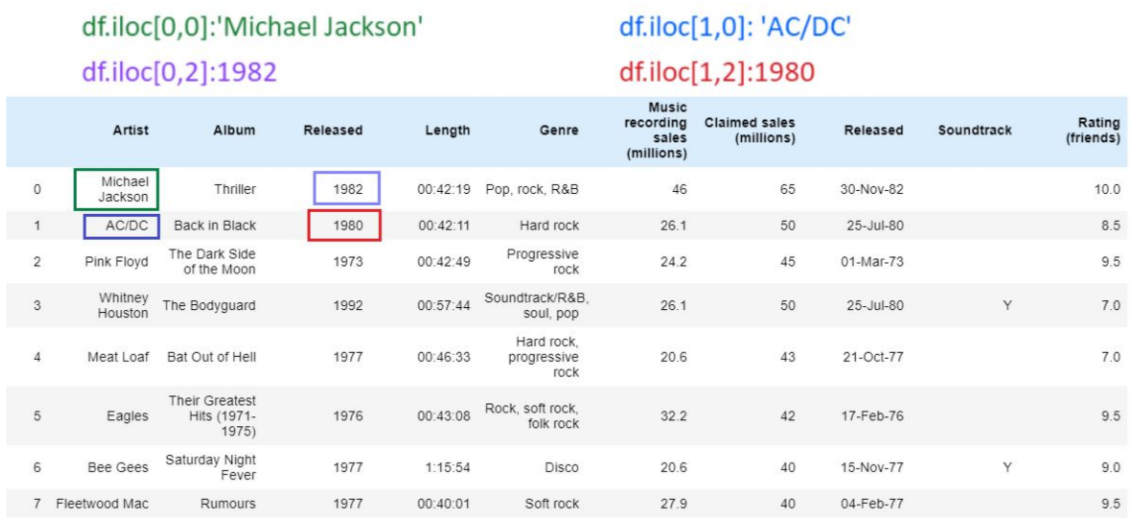
You can access the 1st row and the 3rd column as follows:

# Access the value on the first row and the third column

df.iloc[0,2]
1982
# Access the value on the second row and the third column
df.iloc[1,2]

1980

This is shown in the following image



df.iloc[0,0]:'Michael Jackson'

df.iloc[1,0]: 'AC/DC'

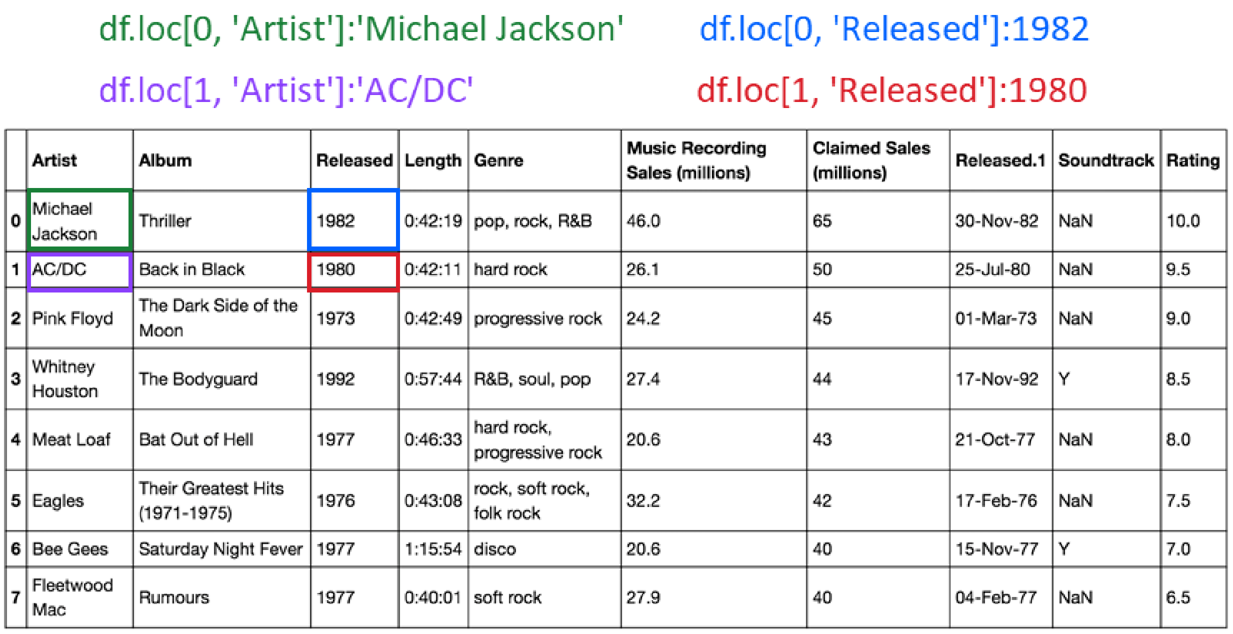
df.iloc[0,2]:1982

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

You can access the column using the name as well, the following are the same as above:

```
# Access the column using the name

df.loc[1, 'Artist']
'AC/DC'
# Access the column using the name

df.loc[1, 'Artist']
'AC/DC'
# Access the column using the name

df.loc[0, 'Released']
1982
# Access the column using the name

df.loc[1, 'Released']
1980
```


df.loc[0, 'Artist']:'Michael Jackson' df.loc[0, 'Released']:1982

df.loc[1, 'Artist']:'AC/DC' 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

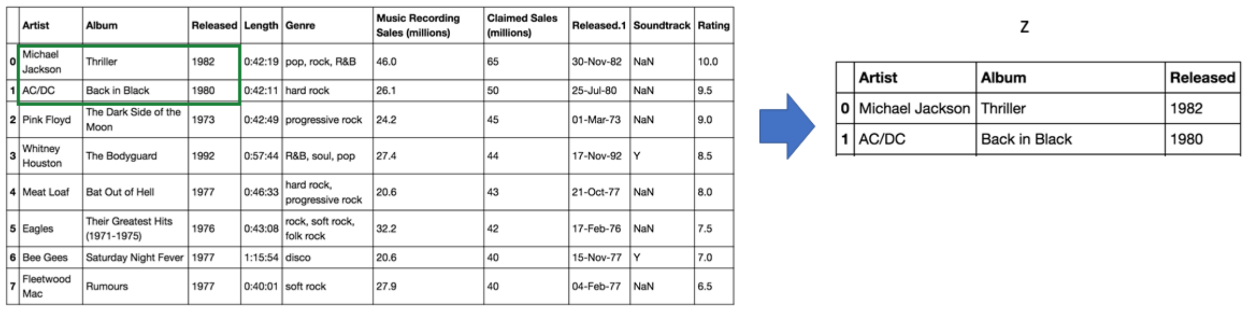
You can perform slicing using both the index and the name of the column:

# Slicing the dataframe

df.iloc[0:2, 0:3]

	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

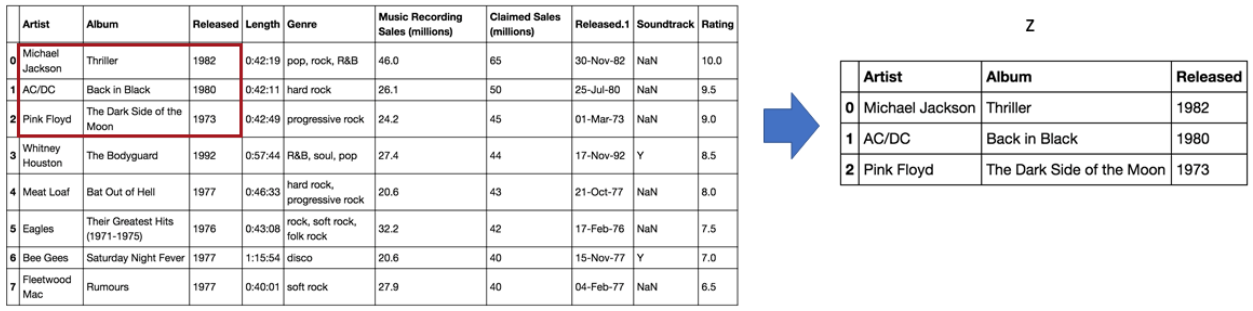
z

	Artist	Album	Released
0	Michael Jackson	Thriller	1982
1	AC/DC	Back in Black	1980

# Slicing the dataframe using name

df.loc[0:2, 'Artist':'Released']

	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.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
2	Pink Floyd	The Dark Side of the Moon	1973

Quiz on DataFrame

Use a variable q to store the column **Rating** as a dataframe

# Write your code below and press Shift+Enter to execute

Click here for the solution

```
q = df[['Rating']]
q
```

Assign the variable q to the dataframe that is made up of the column **Released** and **Artist**:

# Write your code below and press Shift+Enter to execute

Click here for the solution

```
q = df[['Released', 'Artist']]
q
```

Access the 2nd row and the 3rd column of df:

# Write your code below and press Shift+Enter to execute

Click here for the solution

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'

new_index=['a','b','c','d','e','f','g','h']

Click here for the solution
```
df_new=df
df_new.index=new_index
df_new.loc['a', 'Artist']
df_new.loc['a':'d', 'Artist']
```

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python.

Numpy 2D Arrays

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 list 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 axes.
The number of lists the list contains does not have to do with the dimension but the
shape of the list.
As with the 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 3.
The second element corresponds to the size of each of the nested lists or the number
of columns in the rectangular array 0.
The convention is to label this axis 0 and this axis 1 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 9.
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 list
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 can 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 last 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 coloured differently.
Consider the matrix Y, similarly each element is coloured differently.
We can add the matrices.
This corresponds to adding the elements in the same position i.e., adding elements contained
in the same colour 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 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 scaler is identical to multiplying a matrix by a scaler.
Consider the matrix Y, if we multiply the matrix by the scaler 2 we simply multiply
every element in the matrix by 2.
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 scaler 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 products 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 colour.
Also, consider the matrix "B", where each column is a different colour.
In linear algebra, before we can multiply matrix "A" by matrix "B" we must make sure
that the number of columns in matrix "A", in this case 3, is equal to the number of
rows in matrix "B", in this case 3.
For matrix multiplication to obtain the i-th row and j-th column of the new matrix we take
the dot product of the i-throw of "A" with the j-th columns of "B".
For the 1st column 1st row, we take the dot product of the 1st row of "A" with the first
column of "B" as follows.
The result is 0.
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 2.
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 0.
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 2.
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.
Check out numpy.org.
Thanks for watching this video.

Lab

Get to Know a Numpy Array

You will use the numpy array A for the following questions.

```
import numpy as np
A=np.array([[11,12],[21,22],[31,32]])
```

1. Find the type of x using the function type().

Click here for the solution
```
type(A)
```

2. Find the shape of the array:

Click here for the solution
```
A.shape
```

3. Find the type of data in the array:

Click here for the solution
```
A.dtype
```

4. Find the second row of the numpy array A:

Click here for the solution
```
A[1]
```

Two Types of Multiplication

You will use the following numpy arrays for the next questions:

```
A=np.array([[11,12],[21,22]])
B=np.array([[1, 0],[0,1]])
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_404/2025978733.py in <module>
----> 1 A=np.array([[11,12],[21,22]])
      2 B=np.array([[1, 0],[0,1]])

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

1. Multiply array A and B.

Click here for the solution

```
C = A * B
C
```

2. Perform matrix multiplication on array A and B (order will not matter in this case).

Click here for the solution

```
Z = np.dot(A,B)
Z
```

Quiz:

QUESTION 1

1/1 point (ungraded)

How many rows are in the following numpy array?

A = np.array([[1,2],[3,4],[5,6],[7,8]])

Answer: 4

QUESTION 2

1/1 point (ungraded)

Is it possible to perform the following operation?

```
A = np.array([[1,2],[3,4],[5,6],[7,8]])

B = np.array([[1,2,3],[4,5,6],[7,8,9]])

np.dot(A,B)
```

Answer: no