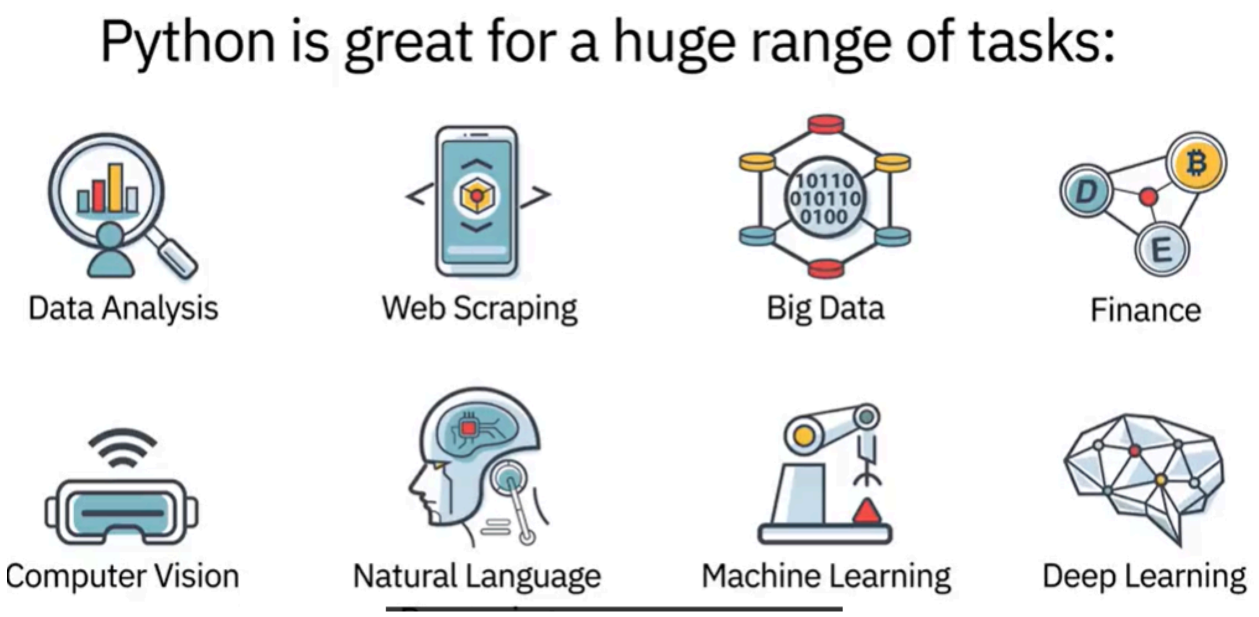
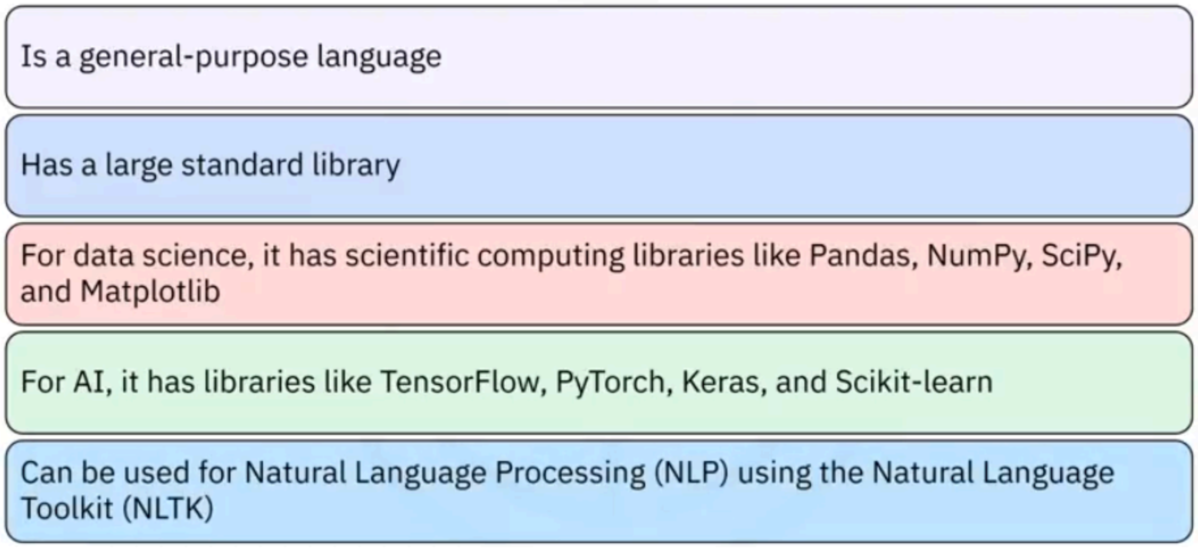
intro and Data Structure Python

Introduction

- The best part is that Python is super easy to learn and is often one of the first languages people turn to when trying to learn to code. Python is very powerful. It has a huge ecosystem of libraries that will help you get the most complex things done with just a few lines of code. Python is great for everything from data analysis, web scraping, to working with big data, finance, computer vision, natural language processing, machine learning, deep learning, and much more. Python can do anything you can throw at it.



What makes Python great


Is a general-purpose language

Has a large standard library

For data science, it has scientific computing libraries like Pandas, NumPy, SciPy, and Matplotlib

For AI, it has libraries like TensorFlow, PyTorch, Keras, and Scikit-learn

Can be used for Natural Language Processing (NLP) using the Natural Language Toolkit (NLTK)

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:

```python
# Check the Python Version

import sys
print(sys.version)
```

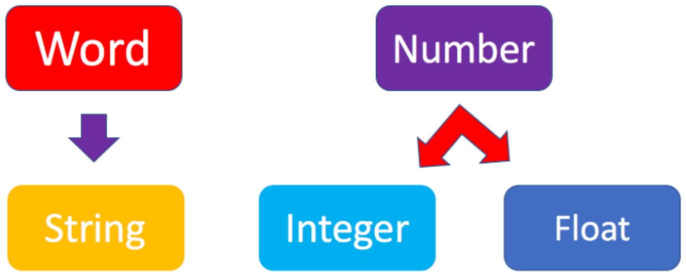
```
3.12.1 (main, Jun  7 2024, 22:40:30) [Clang 19.0.0git (https:/github.com/llvm/llvm-project 0a8cd1ed1f4f35905df318015b
```

[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 explictly `import` it.

Data Types

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).



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`.

```
# Type of 12

type(12)
```

```
# Type of 2.14

type(2.14)
```

```
# Type of "Hello, Python 101!"

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

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:

```
# Verify that this is an integer

type(2)
```

Converting integers to floats

Let's cast integer 2 to float:

```
# Convert 2 to a float

float(2)
```

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()`:

```
# Convert a string into an integer

int('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:

```
# Convert a string into an integer with error

int('1 or 2 people')
```

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

```
# Convert the string "1.2" into a float

float('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?

```
# Convert an integer to a string

str(1)
```

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

```
# Convert a float to a string

str(1.2)
```

Python Operators

- Addition (+): Adds two values together.
- Subtraction (-): Subtracts one value from another.
- Multiplication (*): Multiplies two values.
- Division (/): Divides one value by another, returns a float.
- Floor Division (//): Divides one value by another, returns the quotient as an integer.
- Modulo (%): Returns the remainder after division.

Example:

```
x = 9 y = 4
result_add= x + y # Addition
result_sub= x - y # Subtraction
result_mul= x * y # Multiplication
result_div= x / y # Division
result_fdiv= x // y # Floor Division
result_mod= x % y # Modulo</td>
```

Summary

- Python can distinguish among data types such as integers, floats, strings, and Booleans.
- Integers are whole numbers that can be positive or negative.
- Floats are numbers that have decimal points; they can represent whole or fractional values.
- You can convert integers to floats using typecasting and vice-versa.
- You can convert integers and floats to strings.
- You can convert an integer or float to a Boolean: 0 becomes False, non-zero becomes True.
- Expressions in Python are a combination of values and operations used to produce a single result.
- Expressions perform mathematical operations such as addition, subtraction, multiplication, and so on.
- We can use // to perform integer division, which results in an integer value by discarding the fractional part.
- Python follows the order of operations (BODMAS) to perform operations with multiple expressions.
- Variables store and manipulate data, allowing you to access and modify values throughout your code.
- The assignment operator "=" assigns a value to a variable.
- Assigning another value to the same variable overrides the previous value of that variable.
- You can perform mathematical operations on variables using the same or different variables.
- Modifying the value of one variable will affect other variables only if they reference the same mutable object.
- Python string operations involve manipulating text data using tasks such as indexing, concatenation, slicing, and formatting.
- A string is usually written within double quotes or single quotes, including letters, white space, digits, or special characters.
- A string can be assigned to a variable and is an ordered sequence of characters.
- Characters in a string identify their index numbers, which can be positive or negative.
- Strings are sequences that support operations like indexing and slicing.
- You can input a stride value to perform slicing while operating on a string.
- Operations like concatenation and replication produce new strings, while finding the length of a string returns a number.
- You cannot modify an existing string; they are immutable.
- You can use escape sequences with a backslash (\) to change the layout of a string. (For example, \n for a new line, \t for a tab, and \\ for a backslash, etc.)
- In Python, you perform tasks such as searching, modifying, and formatting text data with its pre-built string methods.
- You apply a method to a string to change its value, resulting in another string.
- You can perform actions such as changing the case of characters in a string, replacing items in a string, finding items in a string, and so on using pre-built string methods.

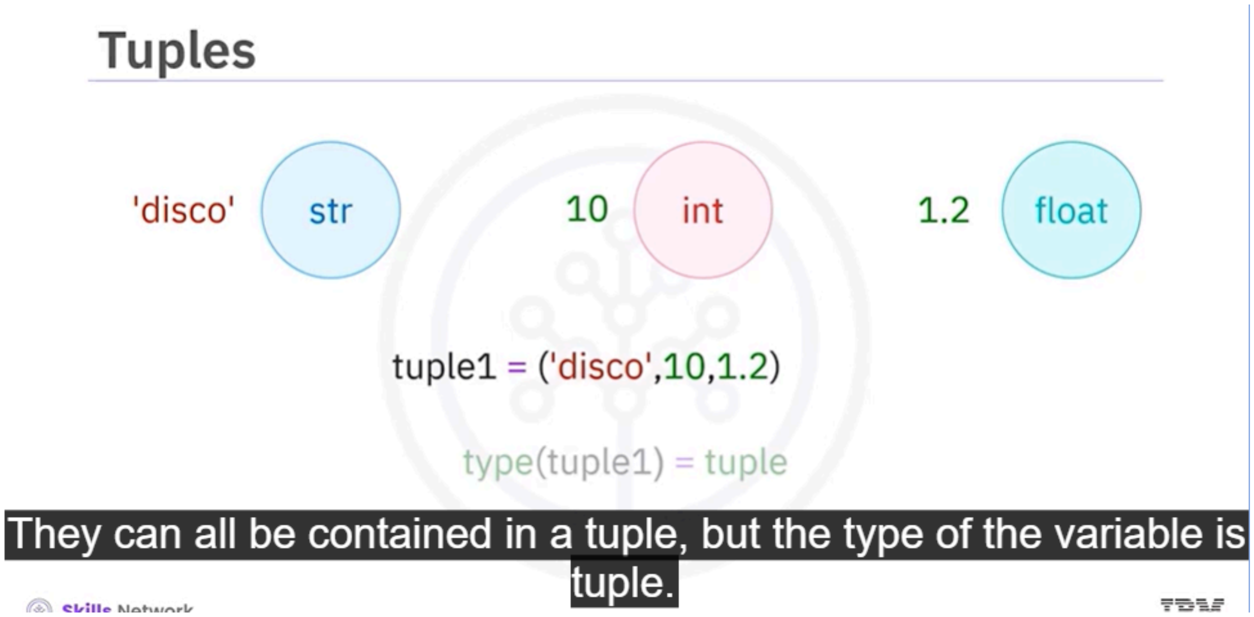
AI	AI (artificial intelligence) is the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings.

Application development	Application development, or app development, is the process of planning, designing, creating, testing, and deploying a software application to perform various business operations.
Arithmetic Operations	Arithmetic operations are the basic calculations we make in everyday life like addition, subtraction, multiplication and division. It is also called as algebraic operations or mathematical operations.
Array of numbers	Set of numbers or objects that follow a pattern presented as an arrangement of rows and columns to explain multiplication.

Slicing in Python	Slicing is used to return a portion from defined list.

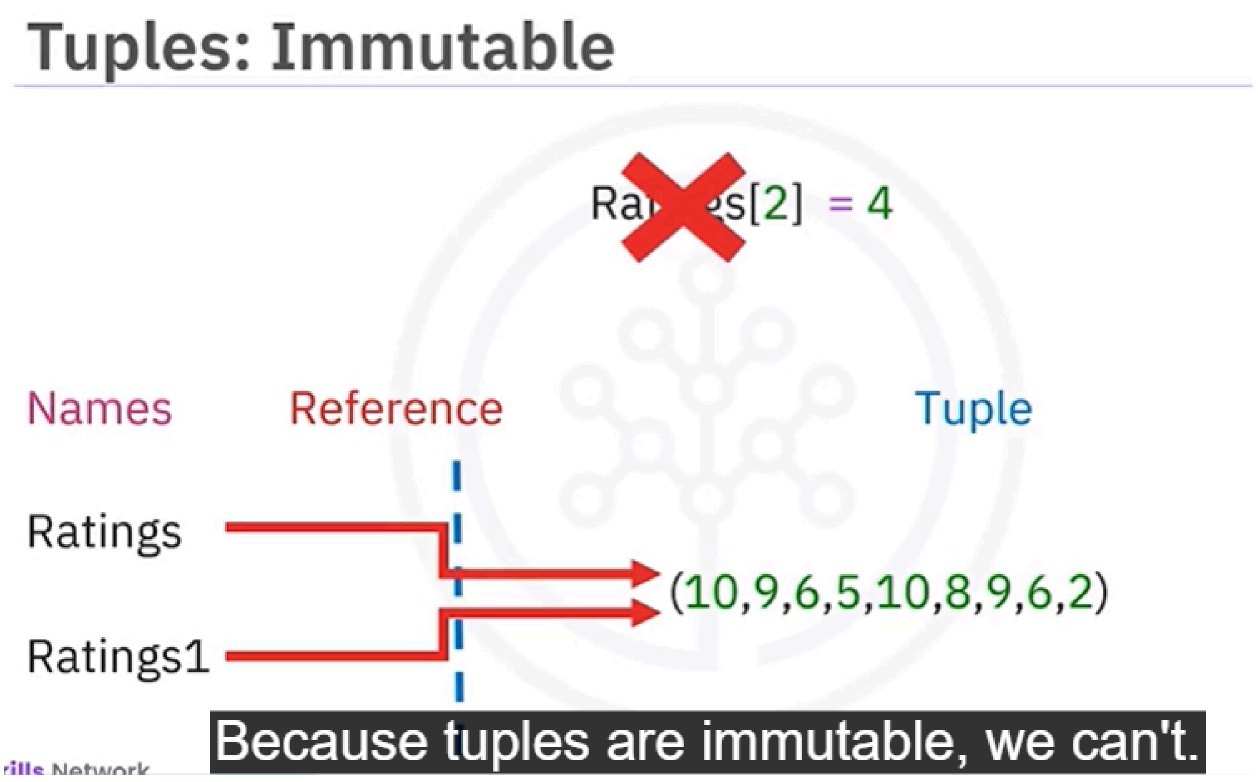
Variables	Variables are containers for storing data values.

Lists and Tuples



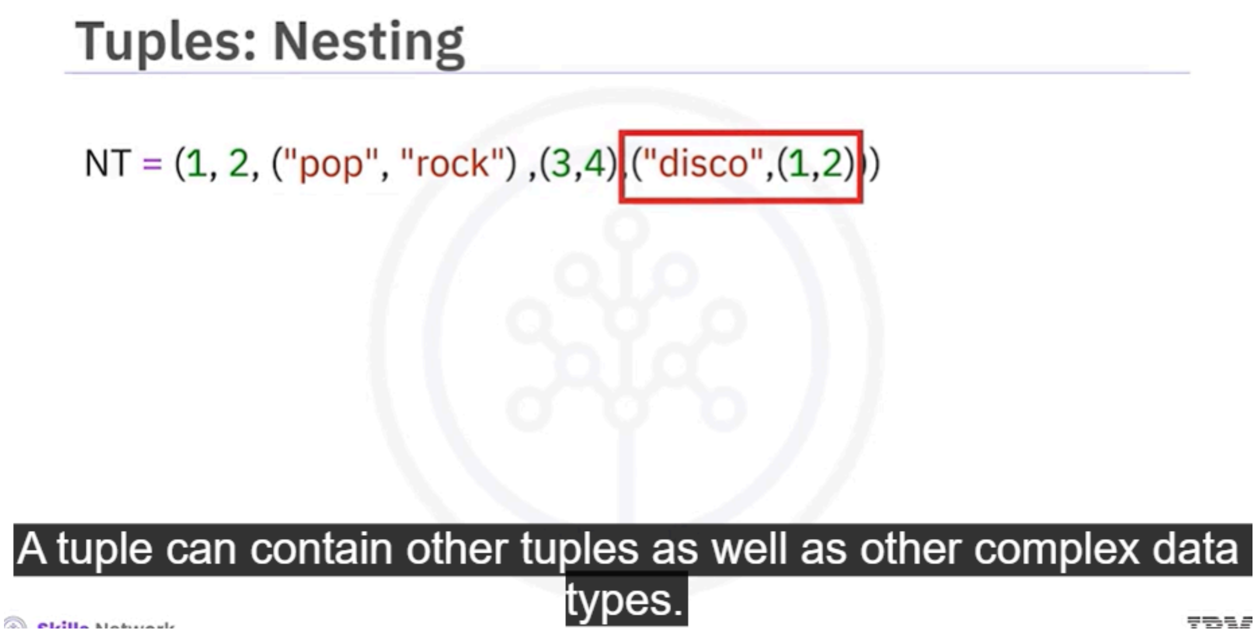
```
tuple1 = ('disco',10,1.2)
type(tuple1) = tuple
```

They can all be contained in a tuple, but the type of the variable is tuple.



Tuples: Immutable

Because tuples are immutable, we can't.

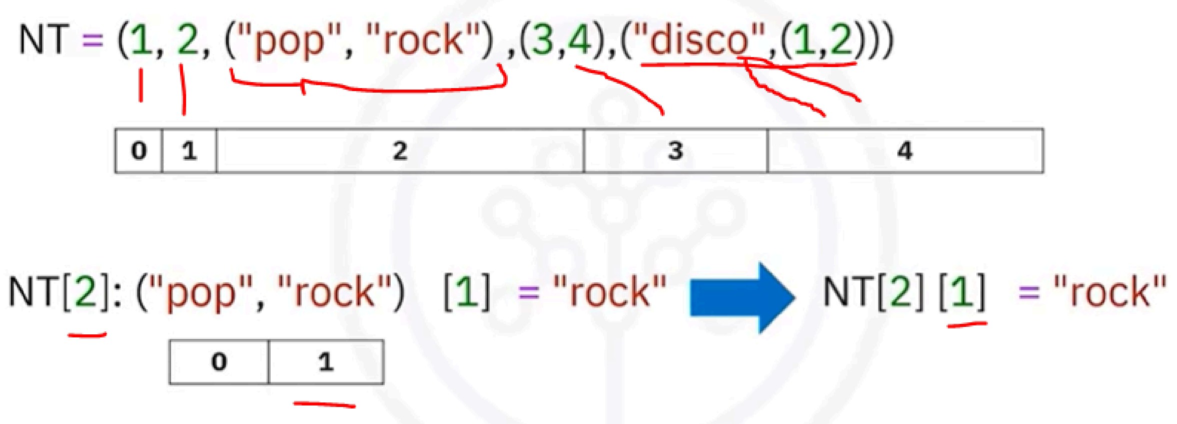


Tuples: Nesting

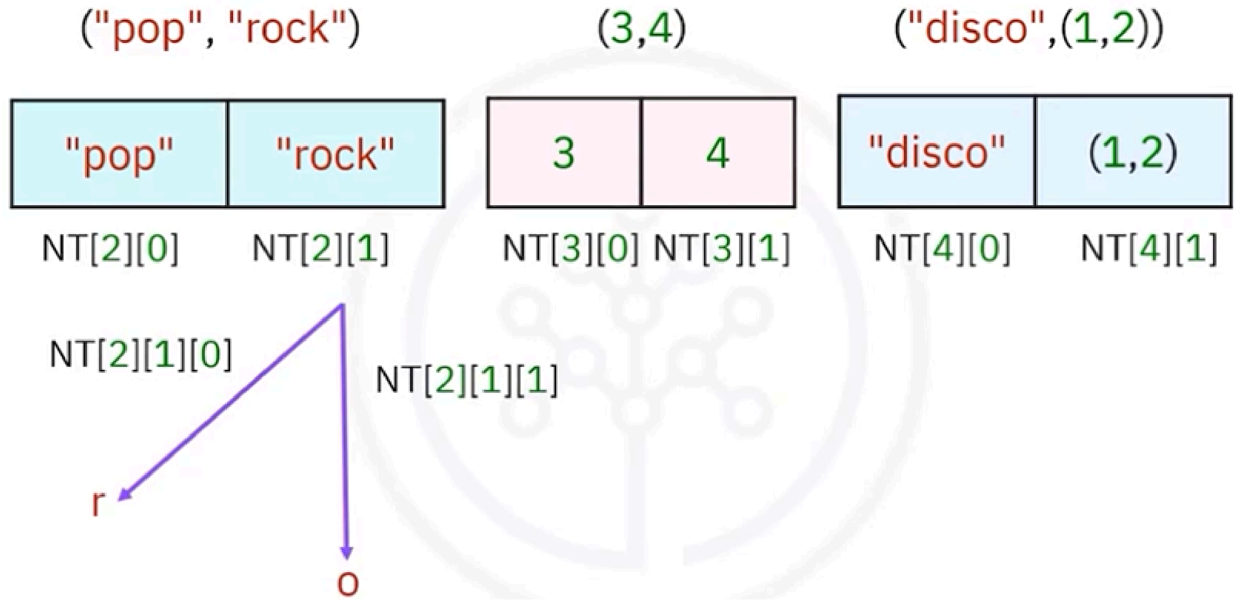
```
NT = (1, 2, ("pop", "rock") ,(3,4),("disco",(1,2)))
```

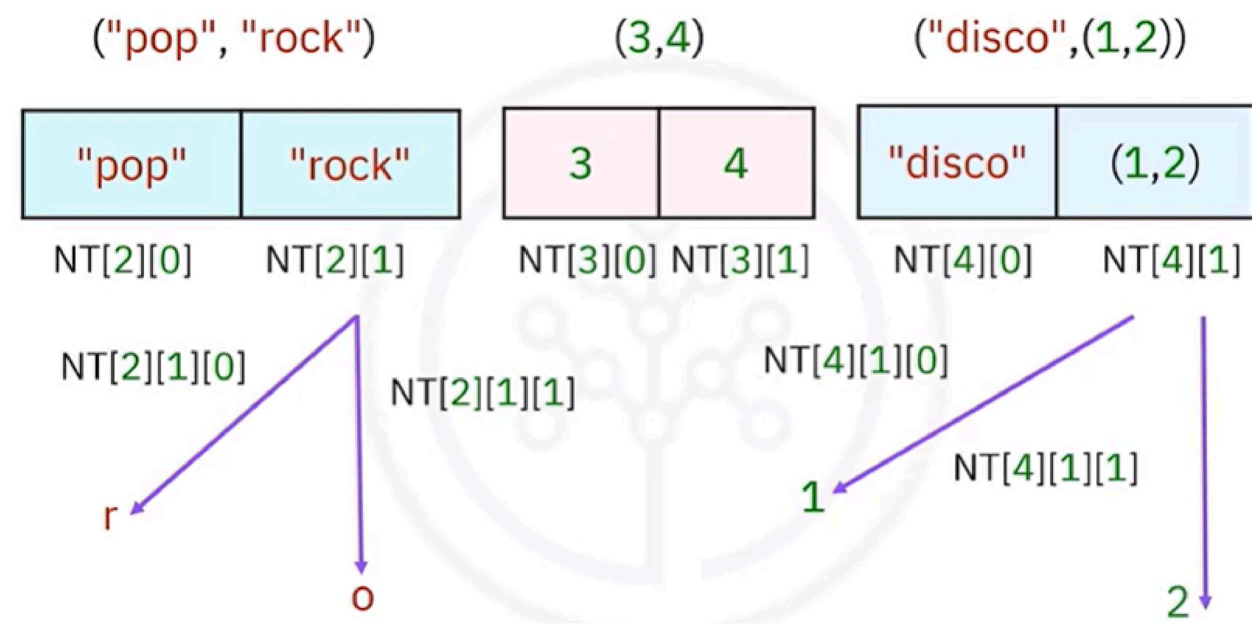
A tuple can contain other tuples as well as other complex data types.

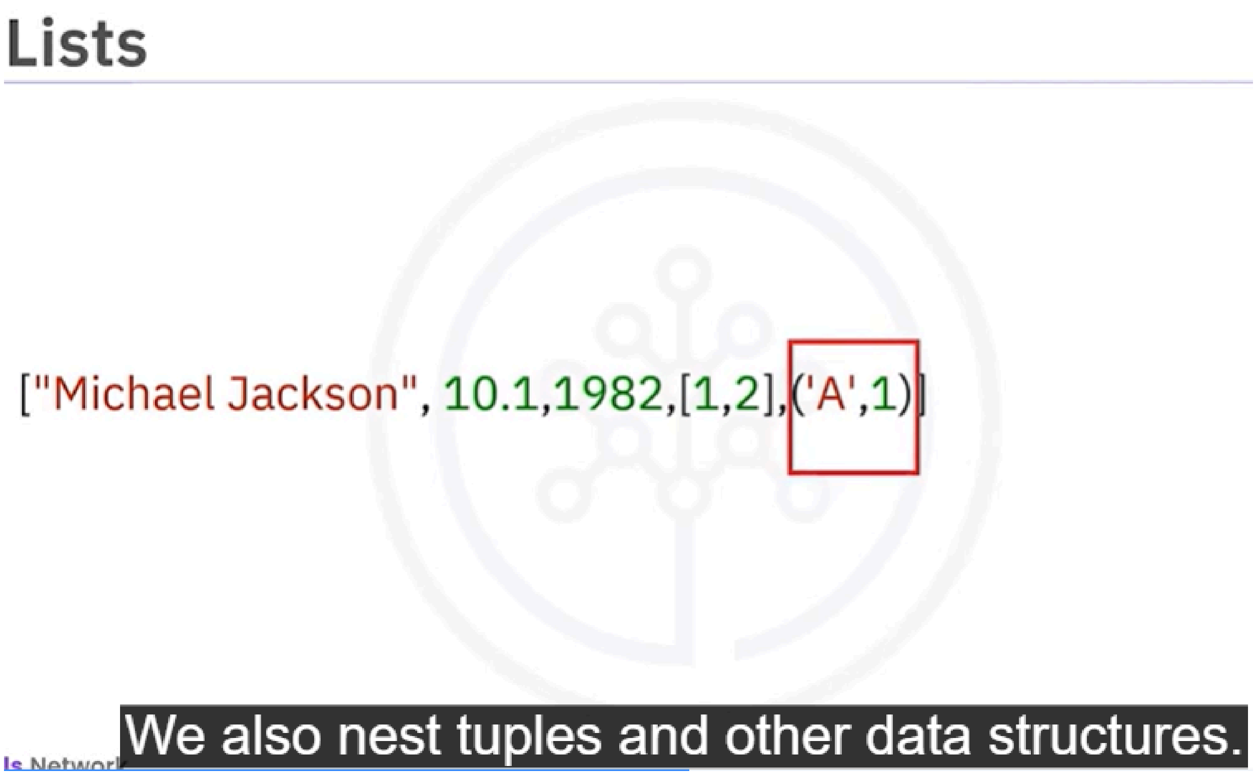
Tuples: Nesting



Ve can apply this indexing directly to the tuple variable nt.







We also nest tuples and other data structures.

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:

```
# Sample List

["The Bodyguard", 7.0, 1992, [1, 2], ("A", 1)]
```

```
['The Bodyguard', 7.0, 1992, [1, 2], ('A', 1)]
```

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:

```
# Split the string by comma

'A,B,C,D'.split(',')
```

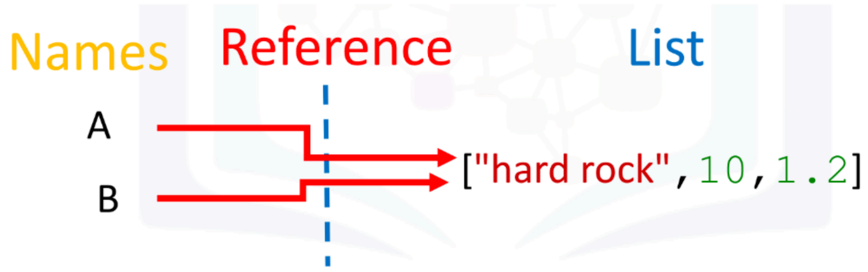
```
['A', 'B', 'C', 'D']
```

Copy and Clone List

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

```
# 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":

```
# Examine the copy by reference

print('B[0]:', B[0])
A[0] = "banana"
print('B[0]:', B[0])
```

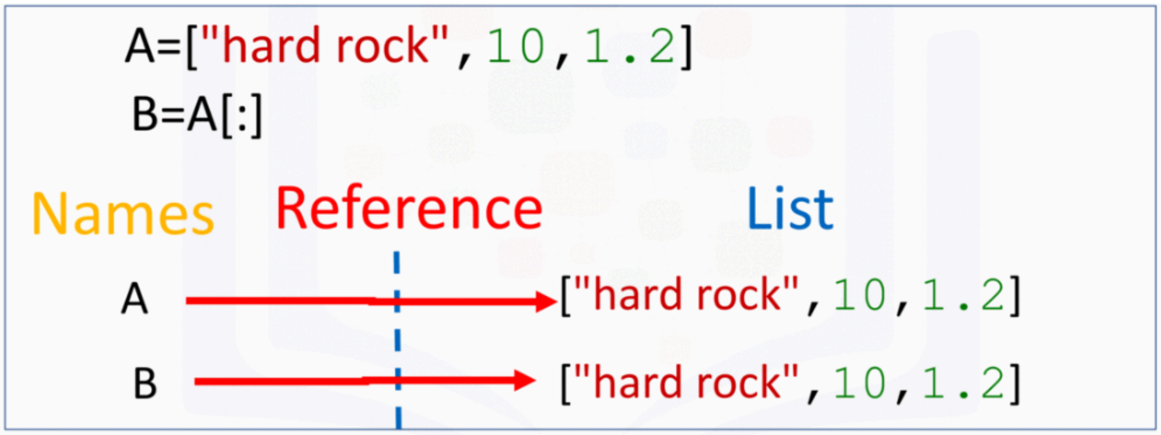
```
B[0]: hard rock
B[0]: banana
```

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

```
# Clone (clone by value) the list A

B = A[:]
B
```

```
['banana', 10, 1.2]
```

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

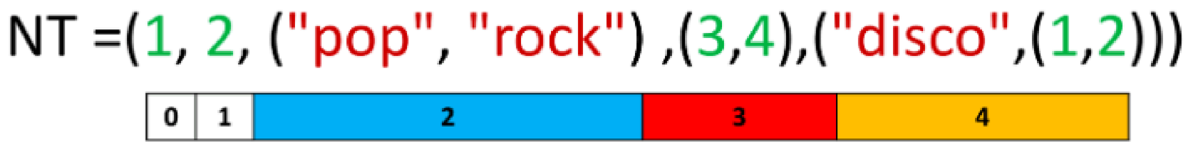


Now if you change **A**, **B** will not change:

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

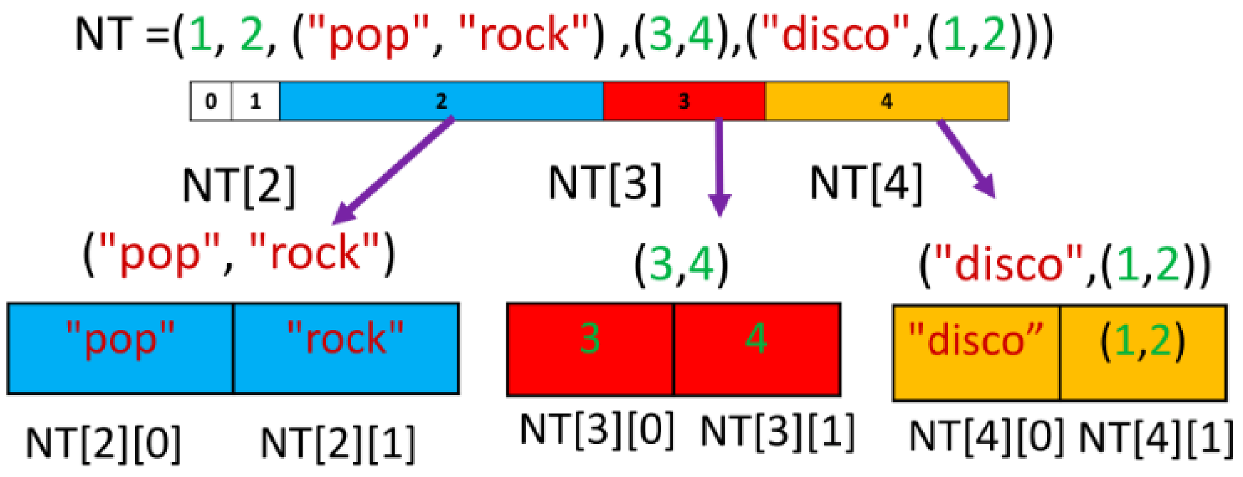
```
B[0]: banana
B[0]: banana
```

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



```
# 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])
```

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



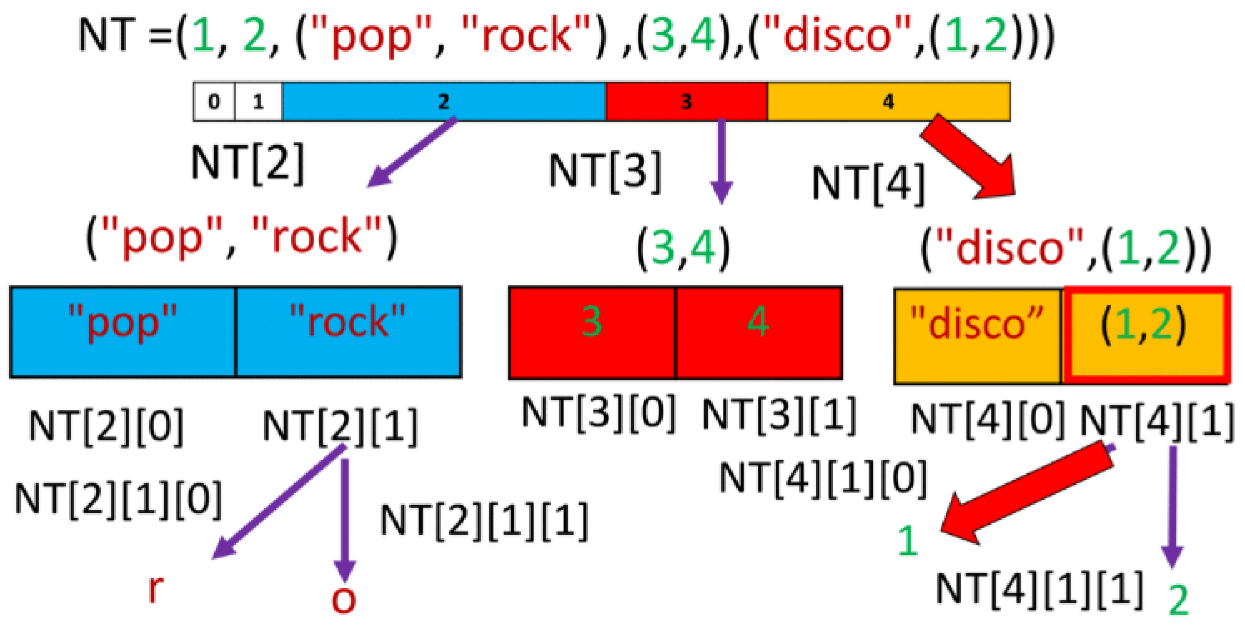
We can access the nested tuples:

```
# 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])
```

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

```
# Print the second element in the second nested tuples

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

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



Sets

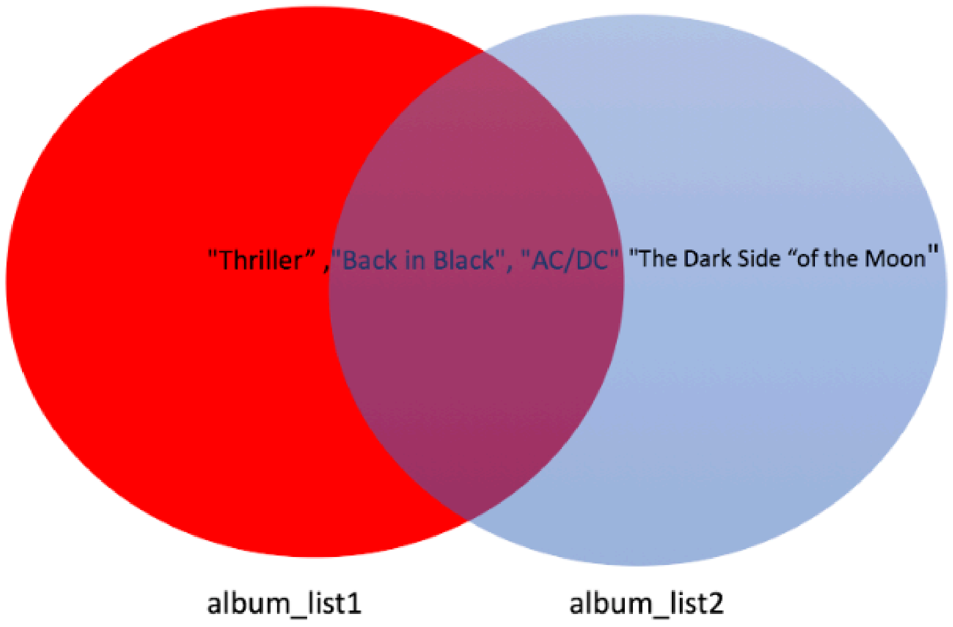
```python
# Convert list to set

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
```

```
{'00:42:19',
 10.0,
 1982,
 '30-Nov-82',
 46.0,
 65,
 'Michael Jackson',
 None,
 'Pop, Rock, R&B',
 'Thriller'}
```

```python
# Print two sets

album_set1, album_set2
```

```
({'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 `&`:

```python
# Find the intersections

intersection = album_set1 & album_set2
intersection
```

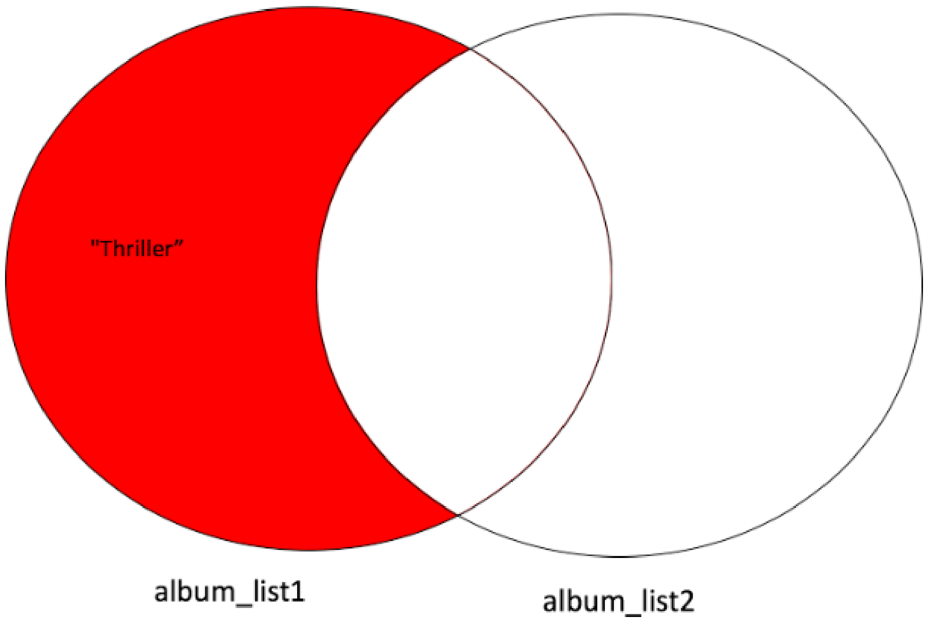
```
{'AC/DC', 'Back in Black'}
```

You can find all the elements that are only contained in `album_set1` using the `difference` method:

```
# Find the difference in set1 but not set2

album_set1.difference(album_set2)
```

```
{'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:

```
album_set2.difference(album_set1)
```

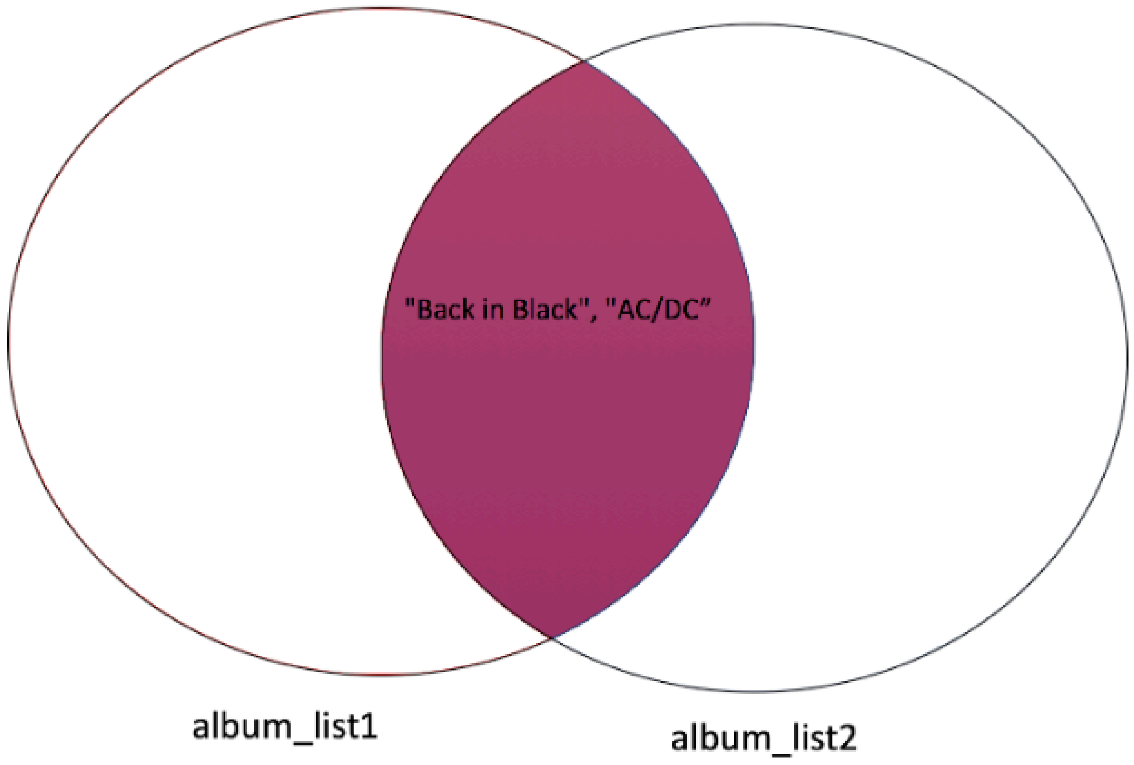
```
{'The Dark Side of the Moon'}
```

You can also find the intersection of `album_list1` and `album_list2`, using the `intersection` method:

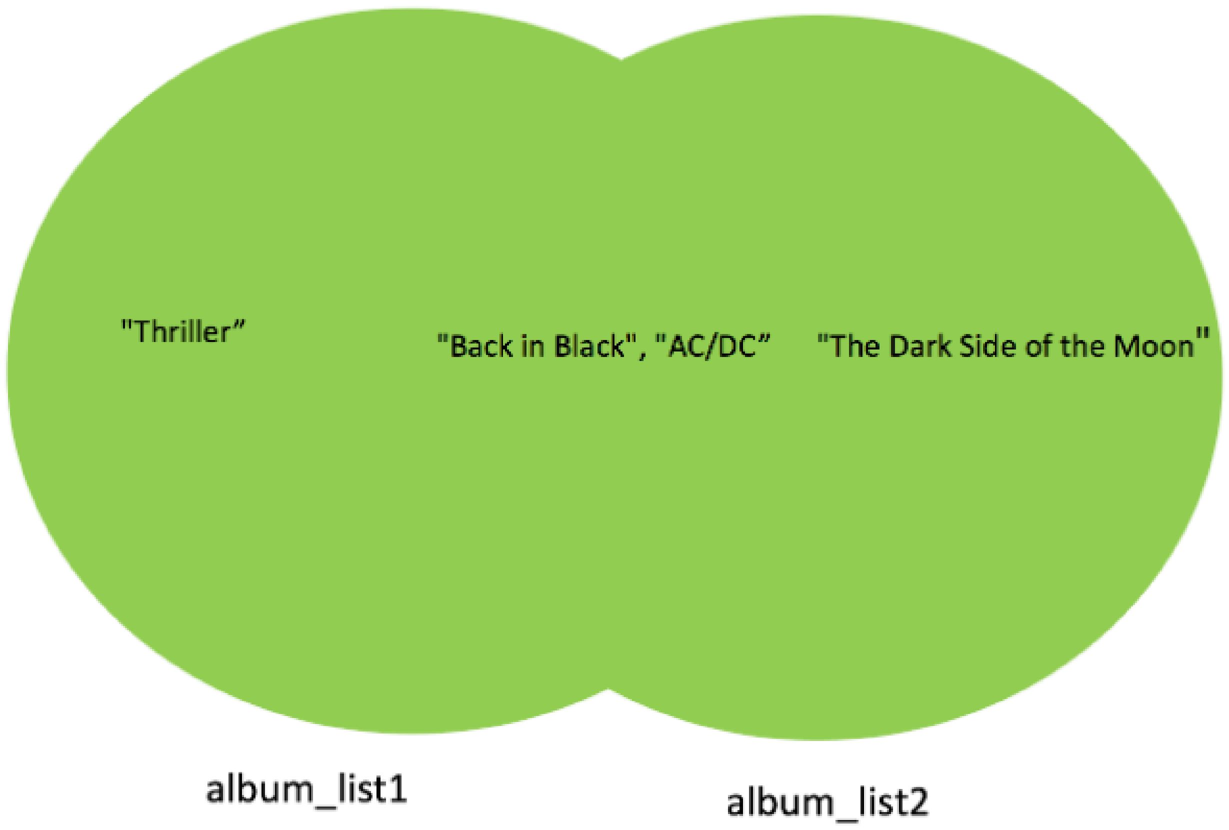
```
# Use intersection method to find the intersection of album_list1 and album_list2

album_set1.intersection(album_set2)
```

```
{'AC/DC', 'Back in Black'}
```

This corresponds to the intersection of the two circles:



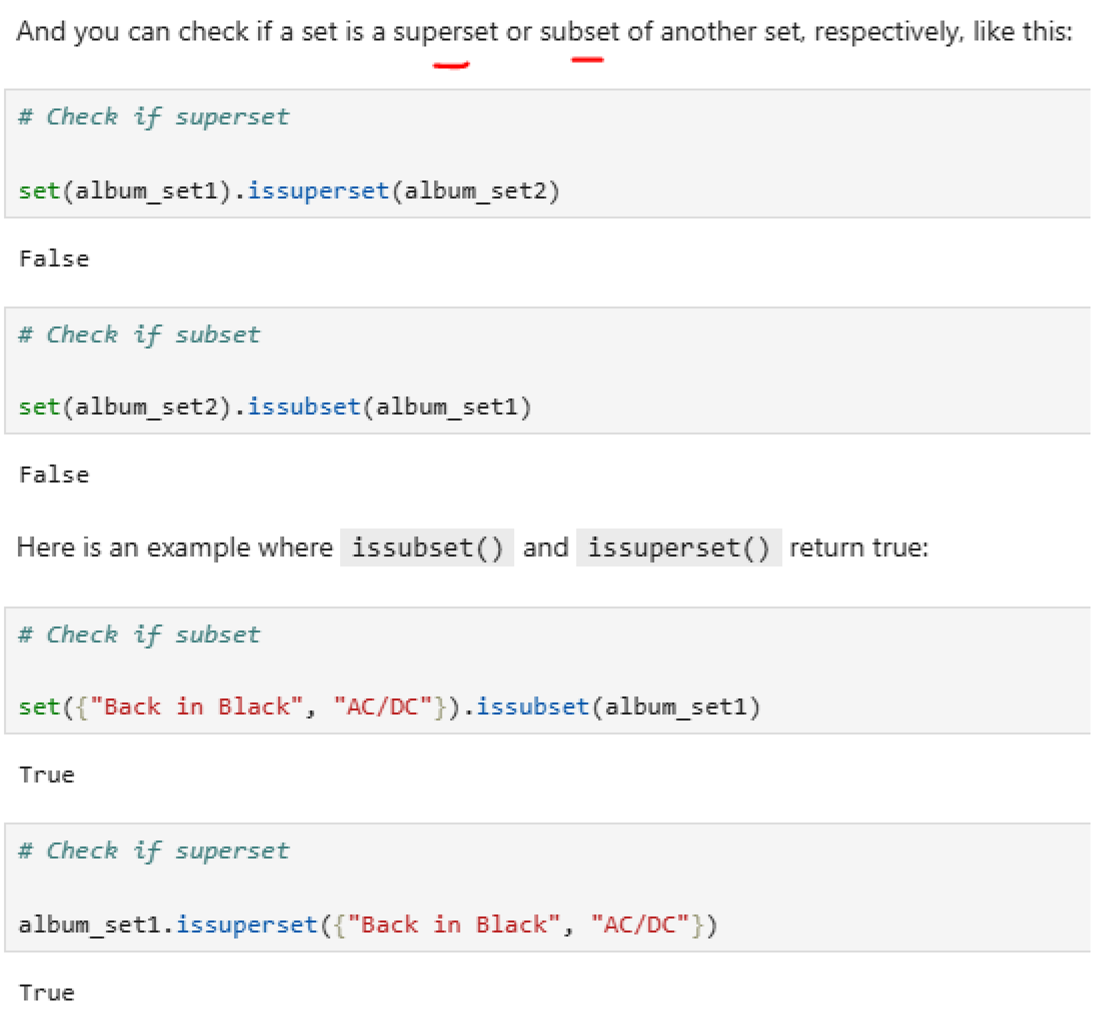
The union corresponds to all the elements in both sets, which is represented by coloring both circles:



The union is given by:

```
# Find the union of two sets

album_set1.union(album_set2)
```

```
{'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:

```
# Check if superset

set(album_set1).issuperset(album_set2)
```

```
False
```

```
# Check if subset

set(album_set2).issubset(album_set1)
```

```
False
```

Here is an example where `issubset()` and `issuperset()` return true:

```
# Check if subset

set({"Back in Black", "AC/DC"}).issubset(album_set1)
```

```
True
```

```
# Check if superset

album_set1.issuperset({"Back in Black", "AC/DC"})
```

```
True
```

Summary: Python Data Structures

- In Python, we often use tuples to group related data together.Tuples refer to ordered and immutable collections of elements.
- Tuples are usually written as comma-separated elements in parentheses “()".
- You can include strings, integers, and floats in tuples and access them using both positive and negative indices.
- You can perform operations such as combining, concatenating, and slicing on tuples.

- Tuples are immutable, so you need to create a new tuple to manipulate it.
- Tuples, termed nesting, can include other tuples of complex data types.
- You can access elements in a nested tuple through indexing.
- Lists in Python contain ordered collections of items that can hold elements of different types and are mutable, allowing for versatile data storage and manipulation.
- A List is an ordered sequence, represented with square brackets "[]".
- Lists possess mutability, rendering them akin to tuples.
- A List can contain strings, integers, and floats; you can nest lists within it.
- You can access each element in a List using both positive and negative indexing.
- Concatenating or appending a List will result in the modification of the same List.
- You can perform operations such as adding, deleting, splitting, and so forth on a List.
- You can separate elements in a List using delimiters.
- Aliasing occurs when multiple names refer to the same object.
- You can also clone a List to create another list.
- Dictionaries in Python are key-value pairs that provide a flexible way to store and retrieve data based on unique keys.
- Dictionaries consist of keys and values, both composed of string elements.
- You denote Dictionaries using curly brackets.
- The keys necessitate immutability and uniqueness.
- The values may be either immutable or mutable, and they allow duplicates.
- You separate each key-value pair with a comma, and you can use color highlighting to make the key more visible.
- You can assign Dictionaries to a variable.
- You use the key as an argument to retrieve the corresponding value.

- You can make additions and deletions to Dictionaries.
- You can perform an operation on a Dictionary to check the key, which results in a true or false output.
- You can apply methods to obtain a list of keys and values in a Dictionary.
- Sets in Python are collections of unique elements, useful for tasks such as removing duplicates and performing set operations like union and intersection. Sets lack order.
- Curly brackets "{}" are helpful for defining elements of a Set.
- Sets do not contain duplicate items.
- A list passed through the Set function generates a set containing unique elements.
- You use “Set Operations” to perform actions such as adding, removing, and verifying elements in a set.
- You can combine Sets using the ampersand "&" operator to obtain the common elements from both sets.
- You can use the Union function to combine two Sets, including both the common and unique elements from both sets.
- The sub-set method is used to determine if two or more sets are subsets.