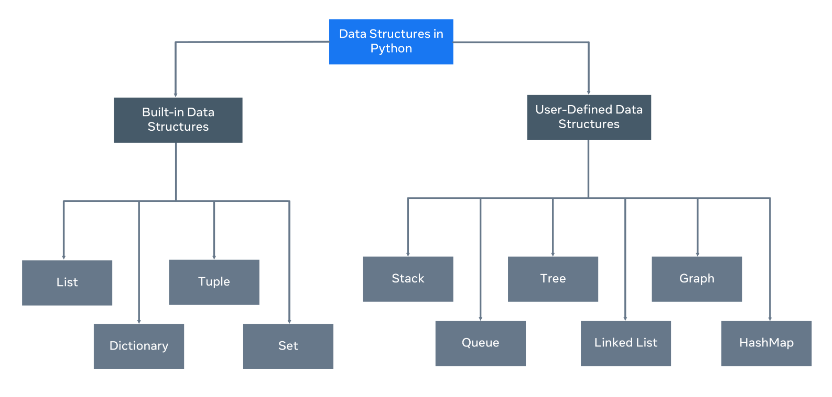
1. Data structures

So far, you have only stored small bits of data in a variable. This was either an integer, Boolean or a string.

But what happens if you need to work with more complex information, such as a collection of data like a list of people or a list of companies? Data structures are designed for this very purpose.



A data structure allows you to organize and arrange your data to perform operations on them. Python has the following built-in data structures: **List**, **dictionary**, **tuple** and **set**. These are all considered **non-primitive** data structures, meaning they are classed as objects, we will explore this more.

Along with the built-in data structures, Python allows users to create their own. Data structures such as Stacks, Queues and Trees can all be created by the user.

Each data structure can be designed to solve a particular problem or optimize a current solution to make it much more performant.

**Mutability and Immutability**

Data Structures can be mutable or immutable.

**Mutability** refers to data inside the data structure that can be modified. For example, you can either change, update, or delete the data when needed. A list is an example of a mutable data structure.

An **immutable** data structure will not allow modification once the data has been set. The tuple is an example of an immutable data structure.

2. Lists and tuples

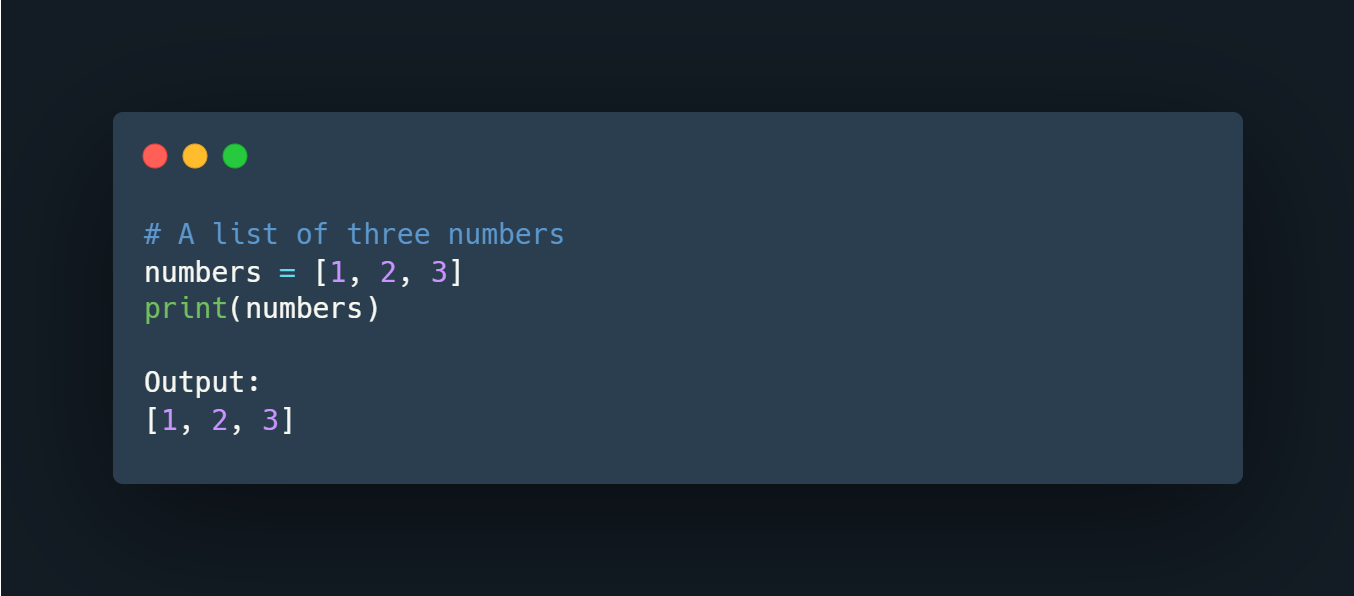
In Python, lists are used to store multiple data at once. For example,

Suppose we need to record the ages of **5** students. Instead of creating **5** separate variables, we can simply create a list:

Lists Elements

**Create a Python List**

A list is created in Python by placing items inside [], separated by commas. For example,



Here, we have created a list named numbers with **3** integer items.

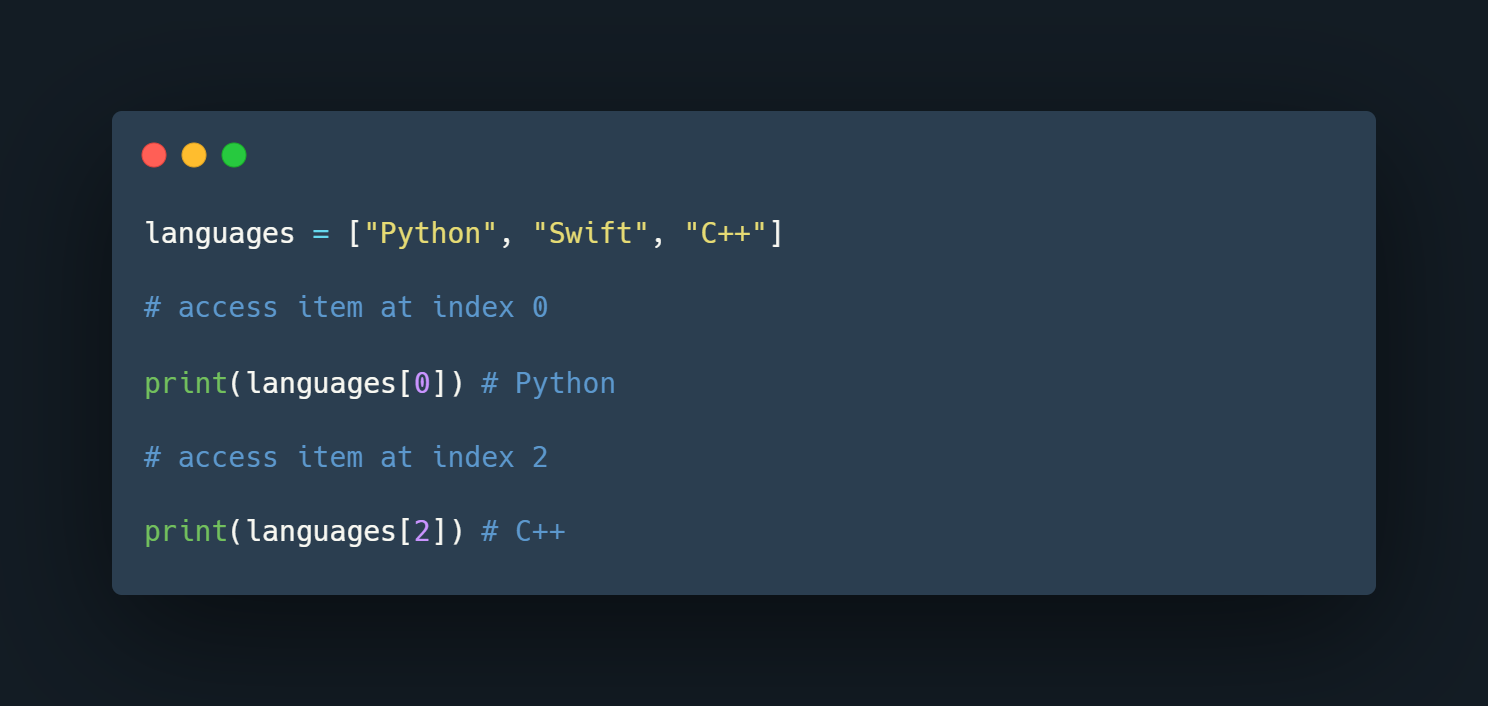
A list can have any number of items and they may be of different types (integer, float, string, etc.). For example,



**Access Python List Elements**

In Python, each item in a list is associated with a number. The number is known as a list index.

We can access elements of an array using the index number **(0, 1, 2 …)**. For example,



In the above example, we have created a list named languages.



List Indexing in Python

Here, we can see each list item is associated with the index number. And, we have used the index number to access the items.

**Note:** The list index always starts with **0**. Hence, the first element of a list is present at index **0**, not **1**.

**Slicing of a Python List**

In Python, it is possible to access a section of items from the list using the slicing operator :, not just a single item. For example,



Here,

* my\_list[2:5] returns a list with items from index **2** to index **4**.
* my\_list[5:] returns a list with items from index **1** to the end.
* my\_list[:] returns all list items

**Note**: When we slice lists, the start index is inclusive but the end index is exclusive.

**Add Elements to a Python List**

Python List provides different methods to add items to a list.

**1. Using append()**

The append() method adds an item at the end of the list.

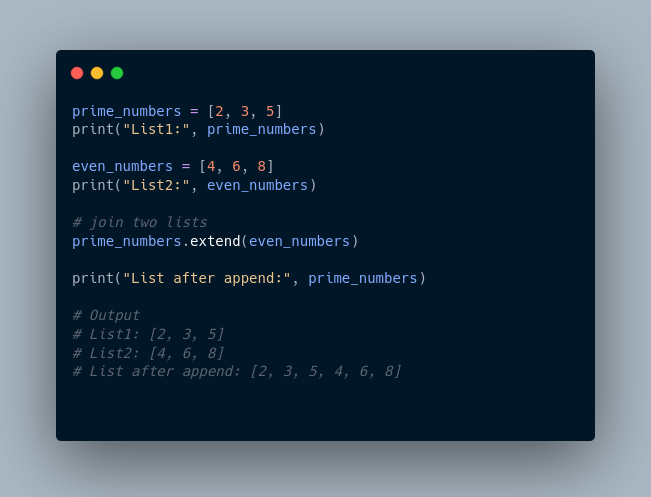
For example,



Here, append() adds **32** at the end of the array.

**2. Using extend()**

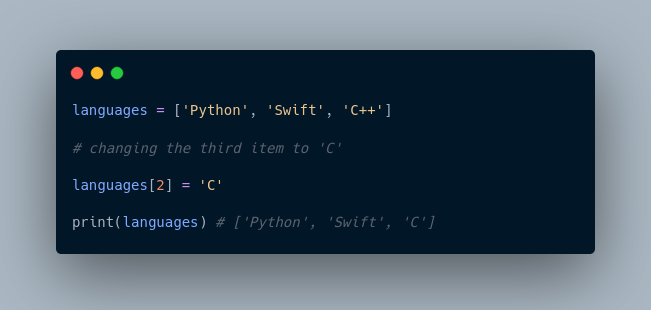
We use the extend() method to add all items of one list to another. For example,



In the above example, we have two lists named prime\_numbers and even\_numbers. Notice the statement,

**Change List Items**

Python lists are mutable. Meaning lists are changeable. And, we can change items of a list by assigning new values using = operator. For example,



**Remove an Item From a List**

**1. Using del()**

In Python, we can use the del statement to remove one or more items from a list. For example,



**2. Using remove()**

We can also use the remove() method to delete a list item.

For example:



Here, languages.remove('Python') removes 'Python' from the languages list.

**Python List Methods**

Python has many useful list methods that make it really easy to work with lists.



**Iterating through a List**

We can use the for loop to iterate over the elements of a list. For example,

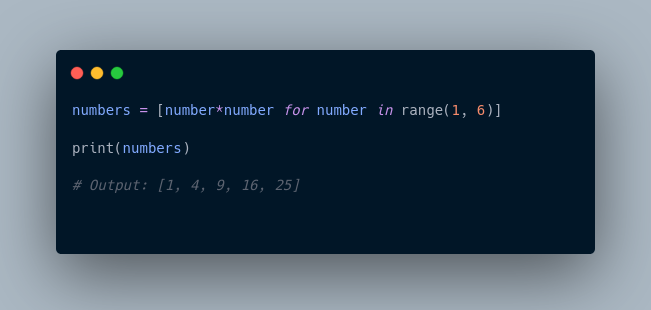


**Python List Comprehension**

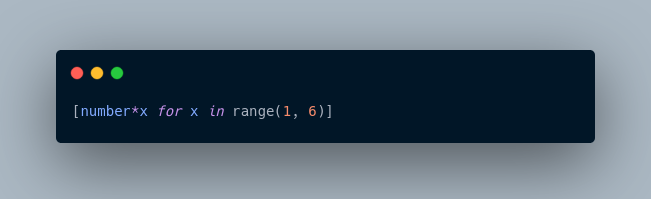
List comprehension is a concise and elegant way to create lists.

A list comprehension consists of an expression followed by the for statement inside square brackets.

Here is an example to make a list with each item being increasing by power of **2.**

****

In the above example, we have used the list comprehension to make a list with each item being increased by power of **2.** Notice the code,



**Tuples**

A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

**Creating a Tuple**

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

**Create a Python Tuple With one Element**

In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.

We can use the type() function to know which class a variable or a value belongs to.



Here,

* ("hello") is a string so type() returns str as class of var1 i.e. <class 'str'>
* ("hello",) and "hello", both are tuples so type() returns tuple as class of var1 i.e. <class 'tuple'>

**Access Python Tuple Elements**

Like a list, each element of a tuple is represented by index numbers **(0, 1, ...)** where the first element is at index **0**.

We use the index number to access tuple elements. For example,

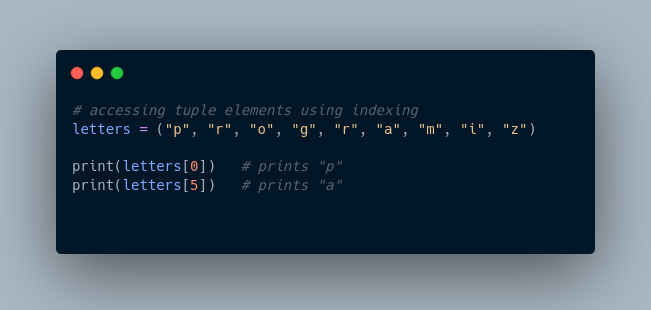
**1. Indexing**

We can use the index operator [] to access an item in a tuple, where the index starts from 0.

So, a tuple having **6** elements will have indices from **0** to **5**. Trying to access an index outside of the tuple index range( **6,7,...** in this example) will raise an IndexError.

The index must be an integer, so we cannot use float or other types. This will result in TypeError.

Likewise, nested tuples are accessed using nested indexing, as shown in the example below.



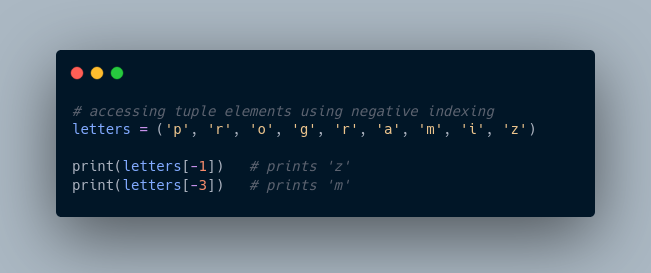
In the above example,

* letters[0] - accesses the first element
* letters[5] - accesses the sixth element

**2. Negative Indexing**

Python allows negative indexing for its sequences.

The index of **-1** refers to the last item, **-2** to the second last item and so on. For example,



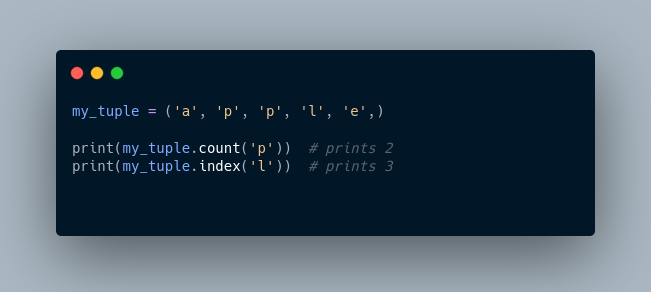
In the above example,

* letters[-1] - access last element
* letters[-3] - access third last element

**Python Tuple Methods**

In Python, methods that add items or remove items are not available with tuples. Only the following two methods are available.

Some examples of Python tuple methods:



Here,

* my\_tuple.count('p') - counts total number of 'p' in my\_tuple
* my\_tuple.index('l') - returns the first occurrence of 'l' in my\_tuple

**More Resources:**

1. <https://www.knowledgehut.com/tutorials/python-tutorial/python-lists-tuples>
2. <https://www.freecodecamp.org/news/python-tuple-vs-list-what-is-the-difference/>
3. <https://builtin.com/software-engineering-perspectives/python-tuples-vs-lists>
4. <https://realpython.com/python-lists-tuples/>

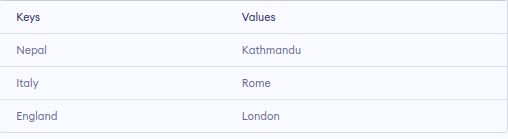
3. Dictionaries

<https://youtu.be/Z17sMiY_Cqg>

Python dictionary is an ordered collection (starting from **Python 3.7**) of items. It stores elements in **key/value** pairs. Here, **keys** are unique identifiers that are associated with each **value**.

Let's see an example,

If we want to store information about countries and their capitals, we can create a dictionary with country names as **keys** and capitals as **values**.



**Create a dictionary in Python**

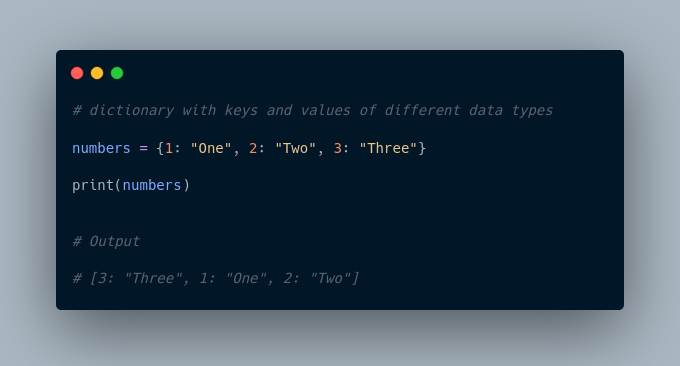
Here's how we can create a dictionary in Python.

In the above example, we have created a dictionary named capital\_city. Here,

1. **Keys** are "Nepal", "Italy", "England"
2. **Values** are "Kathmandu", "Rome", "London"

**Note**: Here, **keys** and **values** both are of string type. We can also have **keys** and **values** of different data types.

**Example 1: Python Dictionary**



In the above example, we have created a dictionary named numbers. Here, **keys** are of integer type and **values** are of string type.

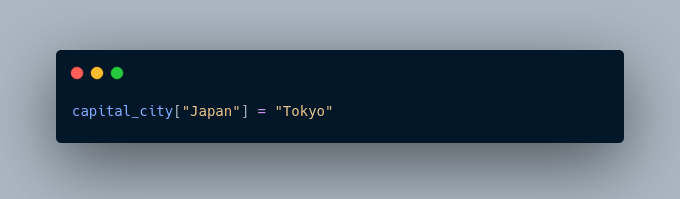
**Add Elements to a Python Dictionary**

We can add elements to a dictionary using the name of the dictionary with [].

For example,



In the above example, we have created a dictionary named capital\_city. Notice the line,



Here, we have added a new element to capital\_city with **key**: Japan and **value**: Tokyo.

**Change Value of Dictionary**

We can also use [] to change the value associated with a particular key. For example,



In the above example, we have created a dictionary named student\_id. Initially, the value associated with the key 112 is "Kyle". Now, notice the line,



Here, we have changed the value associated with the key 112 to "Stan".

**Accessing Elements from Dictionary**

In Python, we use the keys to access their corresponding values. For example,



Here, we have used the keys to access their corresponding values.

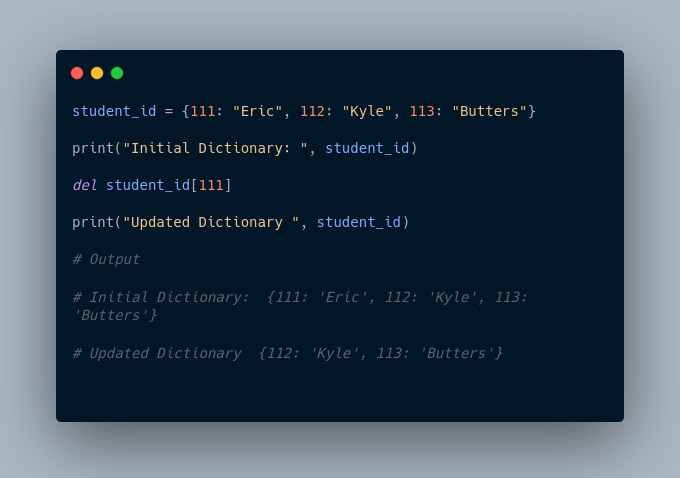
If we try to access the value of a key that doesn't exist, we'll get an error.

For example,

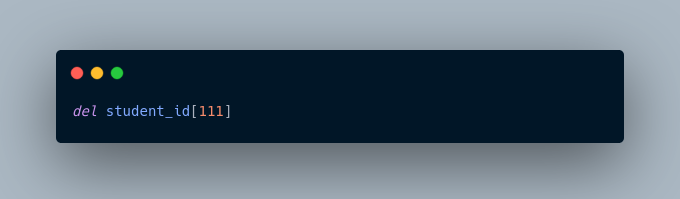


**Removing elements from Dictionary**

We use the del statement to remove an element from the dictionary. For example,

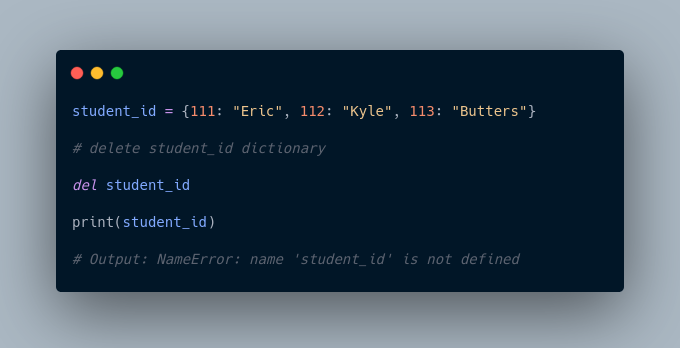


Here, we have created a dictionary named student\_id. Notice the code,



The del statement removes the element associated with the key 111.

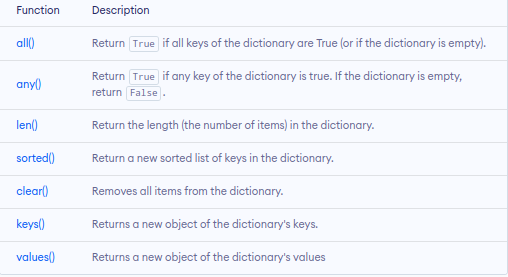
We can also delete the whole dictionary using the del statement,



We are getting an error message because we have deleted the student\_id dictionary and student\_id doesn't exist anymore.

**Python Dictionary Methods**

Methods that are available with a dictionary are tabulated below. Some of them have already been used in the above examples.



**Dictionary Membership Test**

We can test if a key is in a dictionary or not using the keyword in. Notice that the membership test is only for the keys and not for the values.



**Iterating Through a Dictionary**

We can iterate through each key in a dictionary using a loop.



Here, we have iterated through each **key** in the squares dictionary using the for loop.

More Resources:

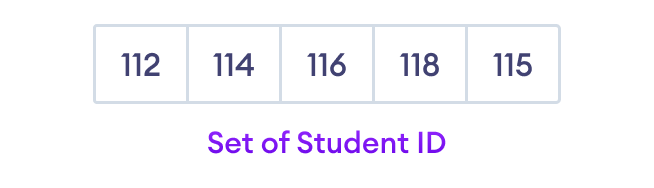
1. <https://www.w3schools.com/python/python_dictionaries.asp>
2. <https://realpython.com/python-dicts/>

4. Sets

A set is a collection of unique data. That is, elements of a set cannot be duplicated.

For example,

Suppose we want to store information about **student IDs**. Since **student IDs** cannot be duplicated, we can use a set.



Python Set Elements

**Create a Set in Python**

In Python, we create sets by placing all the elements inside curly braces {}, separated by comma.

A set can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

Let's see an example,



In the above example, we have created different types of sets by placing all the elements inside the curly braces {}.

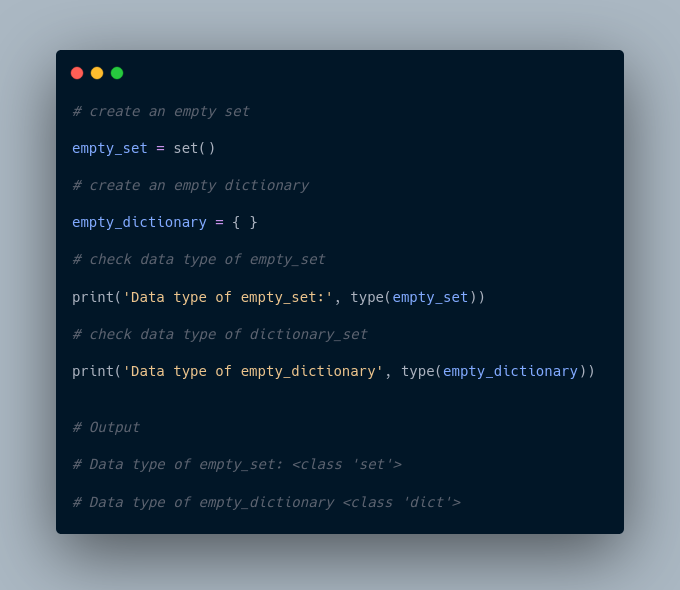
**Note:** When you run this code, you might get output in a different order. This is because the set has no particular order.

**Create an Empty Set in Python**

Creating an empty set is a bit tricky. Empty curly braces {} will make an empty dictionary in Python.

To make a set without any elements, we use the set() function without any argument.

For example,



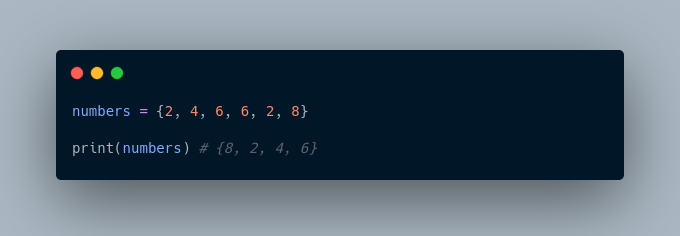
Here,

* empty\_set - an empty set created using set()
* empty\_dictionary - an empty dictionary created using {}

Finally we have used the type() function to know which class empty\_set and empty\_dictionary belong to.

**Duplicate Items in a Set**

Let's see what will happen if we try to include duplicate items in a set.



Here, we can see there are no duplicate items in the set as a set cannot contain duplicates.

**Add and Update Set Items in Python**

Sets are mutable. However, since they are unordered, indexing has no meaning.

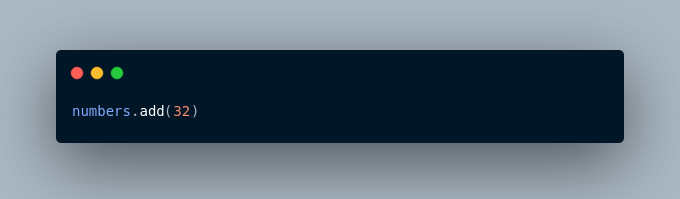
We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.

**Add Items to a Set in Python**

In Python, we use the add() method to add an item to a set. For example,



In the above example, we have created a set named numbers. Notice the line,

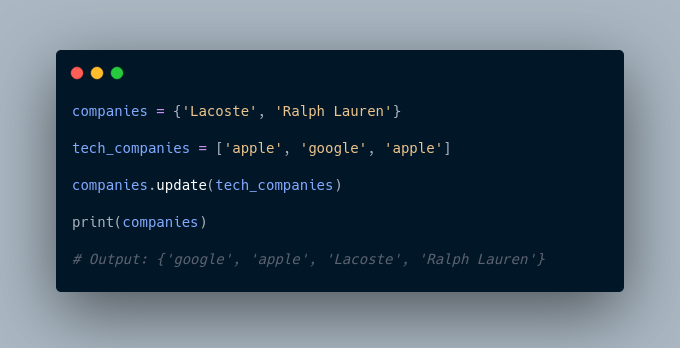


Here, add() adds **32** to our set.

**Update Python Set**

The update() method is used to update the set with items other collection types (lists, tuples, sets, etc).

For example,



Here, all the unique elements of tech\_companies are added to the company's set.

**Remove an Element from a Set**

We use the discard() method to remove the specified element from a set.

For example,



Here, we have used the discard() method to remove 'Java' from the languages set.

**Built-in Functions with Set**

Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with sets to perform different tasks.



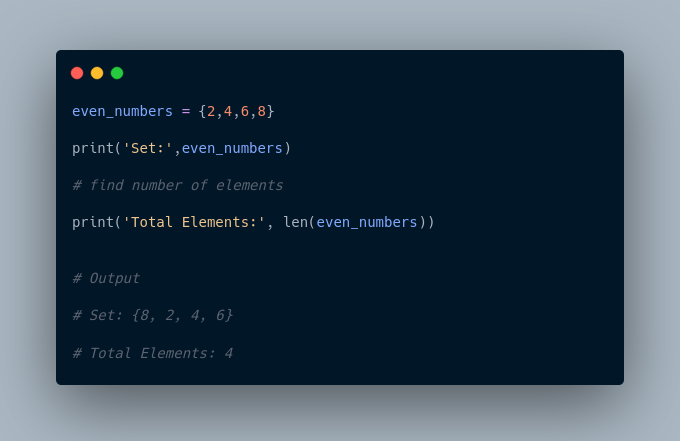
**Iterate Over a Set in Python**

****

**Find Number of Set Elements**

We can use the len() method to find the number of elements present in a Set.

For example,



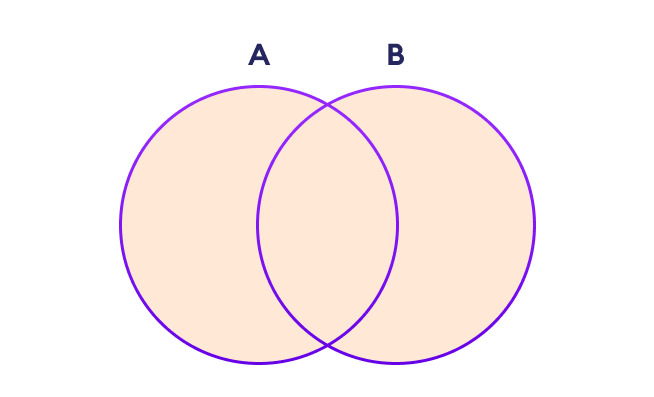
Here, we have used the len() method to find the number of elements present in a Set.

**Python Set Operations**

Python Set provides different built-in methods to perform mathematical set operations like union, intersection, subtraction, and symmetric difference.

**Union of Two Sets**

The union of two sets **A** and **B** include all the elements of set **A** and **B**.



Set Union in Python

We use the | operator or the union() method to perform the set union operation.

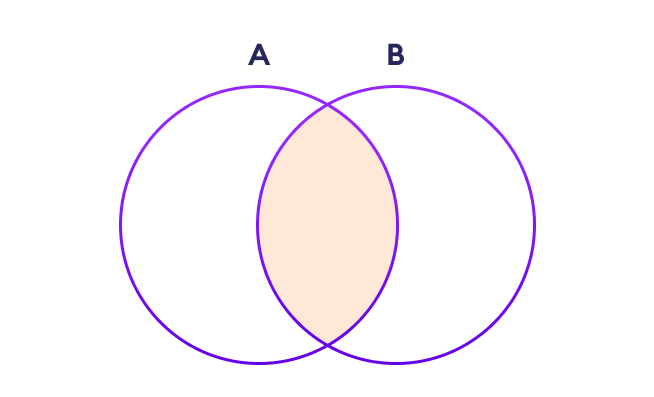
For example,



**Note**: A|B and union() is equivalent to A ⋃ B set operation.

**Set Intersection**

The intersection of two sets **A** and **B** include the common elements between set **A** and **B**.



Set Intersection in Python

In Python, we use the & operator or the intersection() method to perform the set intersection operation.

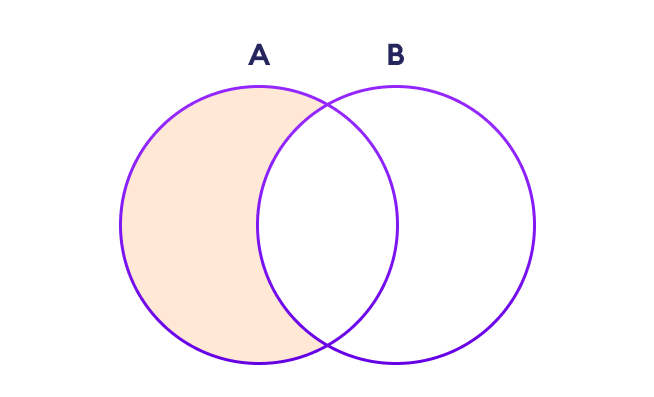
For example,



**Note**: A&B and intersection() is equivalent to A ⋂ B set operation.

**Difference between Two Sets**

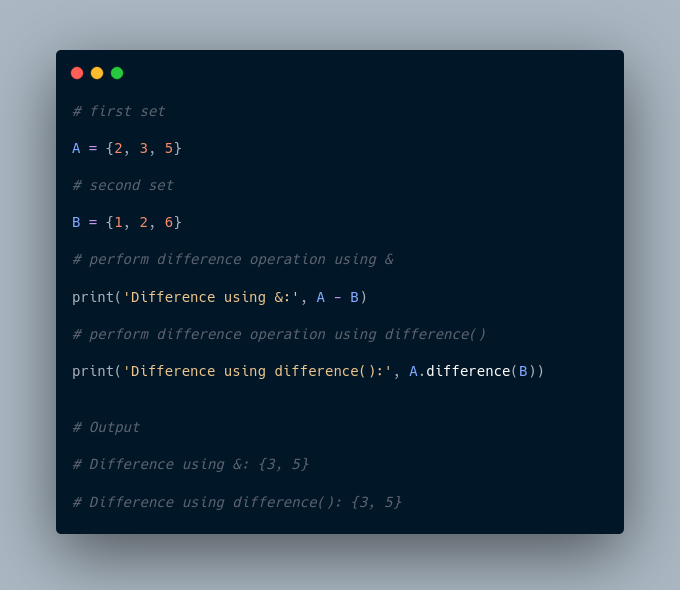
The difference between two sets **A** and **B** include elements of set **A** that are not present on set **B**.



Set Difference in Python

We use the - operator or the difference() method to perform the difference between two sets.

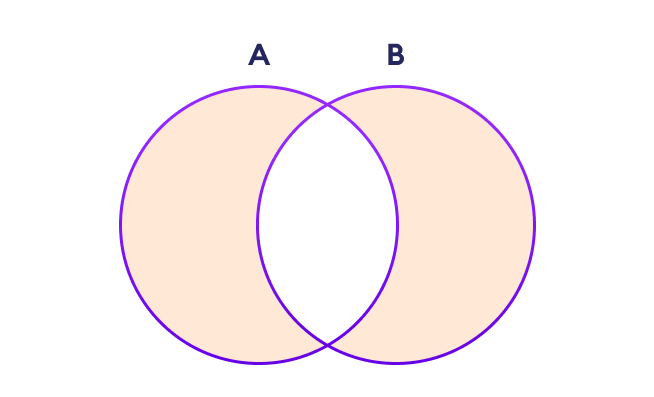
For example,



**Note**: A - B and A.difference(B) is equivalent to A - B set operation.

**Set Symmetric Difference**

The symmetric difference between two sets **A** and **B** includes all elements of **A** and **B** without the common elements.



Set Symmetric Difference in Python

In Python, we use the ^ operator or the symmetric\_difference() method to perform symmetric difference between two sets.

For example,



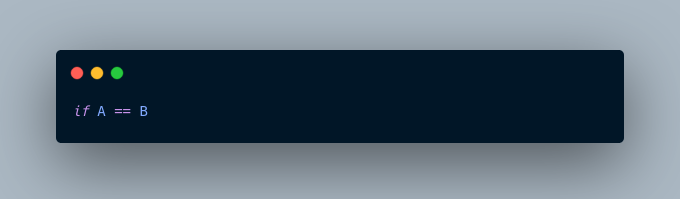
**Check if two sets are equal**

We can use the == operator to check whether two sets are equal or not.

For example,



In the above example, A and B have the same elements, so the condition

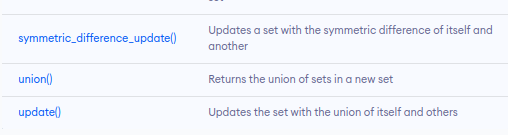


evaluates to True. Hence, the statement print('Set A and Set B are equal') inside the if is executed.

**Other Python Set Methods**

There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with the set objects:





More Resources:

1. <https://www.geeksforgeeks.org/sets-in-python/>
2. <https://realpython.com/python-sets/>

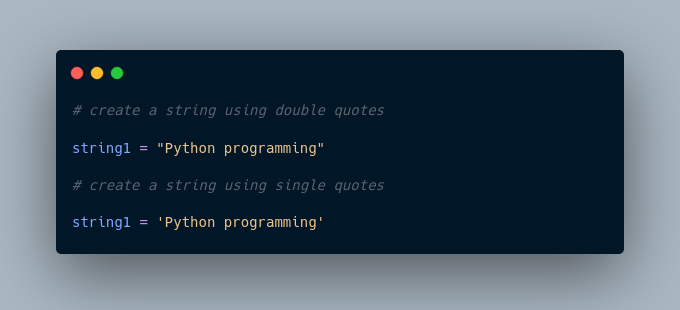
5. Strings

In computer programming, a string is a sequence of characters.

For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.

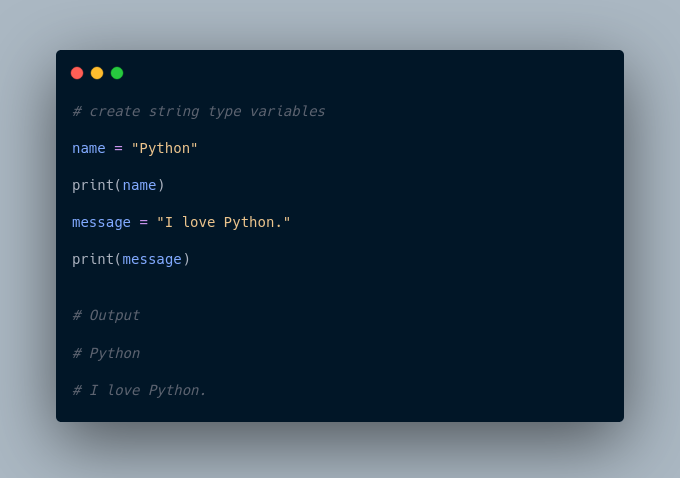
We use single quotes or double quotes to represent a string in Python.

For example,



Here, we have created a string variable named string1. The variable is initialized with the string Python Programming.

**Example: Python String**



In the above example, we have created string-type variables: name and message with values "Python" and "I love Python" respectively.

Here, we have used double quotes to represent strings but we can use single quotes too.

**Access String Characters in Python**

We can access the characters in a string in three ways.

* **Indexing:** One way is to treat strings as a list and use index values.

For example,



* **Negative Indexing:** Similar to a list, Python allows negative indexing for its strings.

For example,



* **Slicing:** Access a range of characters in a string by using the slicing operator colon :

For example,



**Note**: If we try to access an index out of the range or use numbers other than an integer, we will get errors.

**Python Strings are immutable**

In Python, strings are immutable. That means the characters of a string cannot be changed.

For example,



However, we can assign the variable name to a new string.

For example,



**Python Multiline String**

We can also create a multiline string in Python. For this, we use triple double quotes """ or triple single quotes '''.

For example,



In the above example, anything inside the enclosing triple quotes is one multiline string.

**Python String Operations**

There are many operations that can be performed with strings which makes it one of the most used data types in Python.

**1. Compare Two Strings**

We use the == operator to compare two strings. If two strings are equal, the operator returns True. Otherwise, it returns False.

For example,



In the above example,

* str1 and str2 are not equal. Hence, the result is False.
* str1 and str3 are equal. Hence, the result is True.

**2. Join Two or More Strings**

In Python, we can join (concatenate) two or more strings using the + operator.



In the above example, we have used the + operator to join two strings: greet and name.

**Iterate Through a Python String**

We can iterate through a string using a for loop.

For example,



**Python String Length**

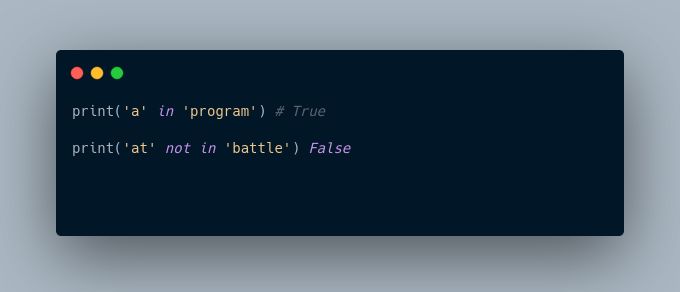
In Python, we use the len() method to find the length of a string.

For example,



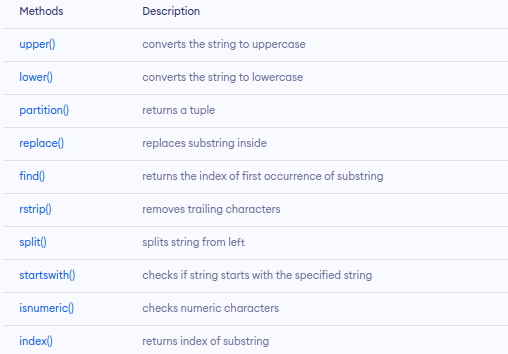
**String Membership Test**

We can test if a substring exists within a string or not, using the keyword in.



**Methods of Python String**

Besides those mentioned above, there are various string methods present in Python. Here are some of those methods:



**Escape Sequences in Python**

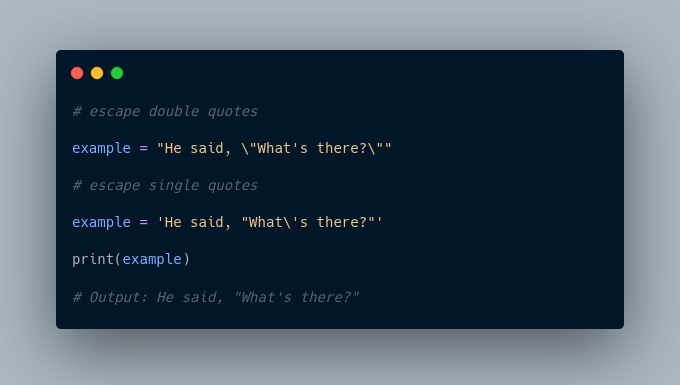
The escape sequence is used to escape some of the characters present inside a string.

Suppose we need to include both double quote and single quote inside a string,

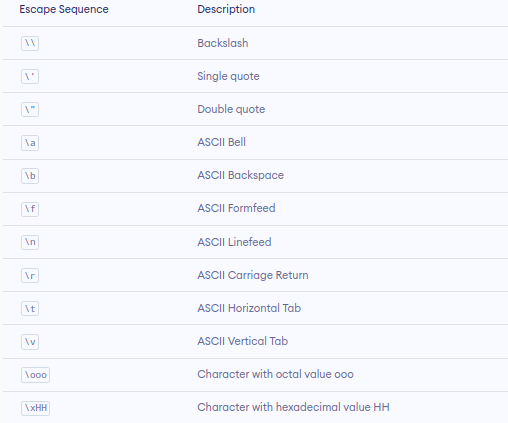


Since strings are represented by single or double quotes, the compiler will treat "He said, " as the string. Hence, the above code will cause an error.

To solve this issue, we use the escape character \ in Python.



Here is a list of all the escape sequences supported by Python.



**Python String Formatting (f-Strings)**

Python **f-Strings** make it really easy to print values and variables.

For example,



Here, f'{name} is from {country}' is an **f-string**.

1. This new formatting syntax is powerful and easy to use. From now on, we will use f-Strings to print strings and variables.

More Resources:

1. <https://www.simplilearn.com/tutorials/python-tutorial/python-strings>
2. <https://www.geeksforgeeks.org/python-string/>
3. <https://www.tutorialspoint.com/python/python_strings.htm>