There are tons of things we could do with **strings** in our scripts. For example, we can check if files are named a certain way by looking at the filename and seeing if they match our criteria, or we can create a list of emails by checking out the users of our system and concatenating our domain.

**String Indexing** = if we have a text that's too long to display and we want to show just a portion of it. Or if we want to make an acronym by taking the first letter of each word in a phrase. We can do that through an operation called string indexing. This operation lets us access the character in a given position or index using square brackets and the number of the position we want.

**A slice** is the portion of a string that can contain more than one character, also sometimes called a substring. We do that by creating a range using a colon as a separator.

**Note :** If you try to access an index that’s larger than the length of your string, you’ll get an IndexError.

String in python is immutabel, for example:

name = “rio”

name[3] = “a”

**it will get error.**

Untuk memperbaikinya, seperti ini :

name = name[0:2]+”a”+name[3:]

**String operations**

* **len(string)** - Returns the length of the string
* **for character in string** - Iterates over each character in the string
* **if substring in string** - Checks whether the substring is part of the string
* **string[i]** - Accesses the character at index i of the string, starting at zero
* **string[i:j]** - Accesses the substring starting at index i, ending at index j minus 1. If i is omitted, its value defaults to 0. If j is omitted, the value will default to len(string).

**String Methods**

* **string.lower()** - Returns a copy of the string with all lowercase characters
* **string.upper()** - Returns a copy of the string with all uppercase characters
* **string.lstrip()** - Returns a copy of the string with the left-side whitespace removed
* **string.rstrip()** - Returns a copy of the string with the right-side whitespace removed
* **string.strip()** - Returns a copy of the string with both the left and right-side whitespace removed
* **string.count(substring)** - Returns the number of times substring is present in the string
* **string.isnumeric()** - Returns True if there are only numeric characters in the string. If not, returns False.
* **string.isalpha()** - Returns True if there are only alphabetic characters in the string. If not, returns False.
* **string.split()** - Returns a list of substrings that were separated by whitespace (whitespace can be a space, tab, or new line)
* **string.split(delimiter)** - Returns a list of substrings that were separated by whitespace or a delimiter
* **string.replace(old, new)** - Returns a new string where all occurrences of old have been replaced by new.
* **delimiter.join(list of strings)** - Returns a new string with all the strings joined by the delimiter

**Lists Defined**

Lists in Python are defined using square brackets, with the elements stored in the list separated by commas: list = ["This", "is", "a", "list"]. You can use the len() function to return the number of elements in a list: len(list) would return 4. You can also use the in keyword to check if a list contains a certain element. If the element is present, it will return a True boolean. If the element is not found in the list, it will return False. For example, "This" in list would return True in our example. Similar to strings, lists can also use indexing to access specific elements in a list based on their position. You can access the first element in a list by doing list[0], which would allow you to access the string "This".

In Python, lists and strings are quite similar. They’re both examples of sequences of data. Sequences have similar properties, like (1) being able to iterate over them using for loops; (2) support indexing; (3) using the len function to find the length of the sequence; (4) using the plus operator + in order to concatenate; and (5) using the in keyword to check if the sequence contains a value. Understanding these concepts allows you to apply them to other sequence types as well.

**Strings** are sequences of characters and are immutable. **Lists** are sequences of elements of any type and are mutable. A third data type that's a sequence and also closely related to lists is the tuple. **Tuples** are sequences of elements of any type that are immutable.

**Iterating Over Lists Using Enumerate**

When we covered for loops, we showed the example of iterating over a list. This lets you iterate over each element in the list, exposing the element to the for loop as a variable. But what if you want to access the elements in a list, along with the index of the element in question? You can do this using the enumerate() function. The enumerate() function takes a list as a parameter and returns a tuple for each element in the list. The first value of the tuple is the index and the second value is the element itself.

But because creating lists based on sequences is such a common task Python provides a technique called **list comprehension**, that lets us do it in just one line**. List comprehensions** let us create new lists based on sequences or ranges.

**Dictionaries** : Like lists, dictionaries are used to organize elements into collections. Unlike lists, you don't access elements inside dictionaries using their position. Instead, the data inside dictionaries take the form of pairs of keys and values. To get a dictionary value we use its corresponding key. Another way these two vary is while in a list the index must be a number, in a dictionary you can use a bunch of different data types as keys, like strings, integers, floats, tuples, and more. The name dictionaries comes from how they work in a similar way to human language dictionaries. In an English language dictionary the word comes with a definition.

When is it best to use a **list** and when is the **dictionary** the way to go? Think about the kind of information you can represent in each data structure. If you've got a list of information you'd like to collect and use in your script then the **list** is probably the right approach. If you had a list of host names and their corresponding IP addresses, you might want to pair them as key values in a **dictionary**. Because of the way **dictionaries** work, it's super easy and fast to search for an element in them.

You want to use **dictionaries** when you plan on searching for a specific element.

One of these data types is a **set** which is a bit like a cross between a list and a dictionary. A **set** is used when you want to store a bunch of elements and be certain that there are only present once. Elements of a set must also be immutable.