## Strings

A string in Python is a data structure that represents a sequence of characters. It is immutable, meaning that once created, it cannot be altered. Strings are widely used in Python for various purposes, such as storing and manipulating text, representing names, addresses, and other types of textual data.

**Syntax:**

```
string_var = 'Learning Python is fun'
print(string_var)
print(type(string_var))
```

**Output:**

```
Learning Python is fun
<class 'str'>
```

## Creating String in Python

```
# Creating a String
# with single Quotes
String1 = 'I love my country'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String
# with double Quotes
String1 = "I'm a citizen"
print("\nString with the use of Double Quotes: ")
print(String1)

# Creating a String
# with triple Quotes
```

```
String1 = '''I'm a citizen and I say "I love my country"'''
print("\nString with the use of Triple Quotes: ")
print(String1)

# Creating String with triple
# Quotes allows multiple lines
String1 = '''I love
            my
            country'''
print("\nCreating a multiline String: ")
print(String1)
```

**Output:**

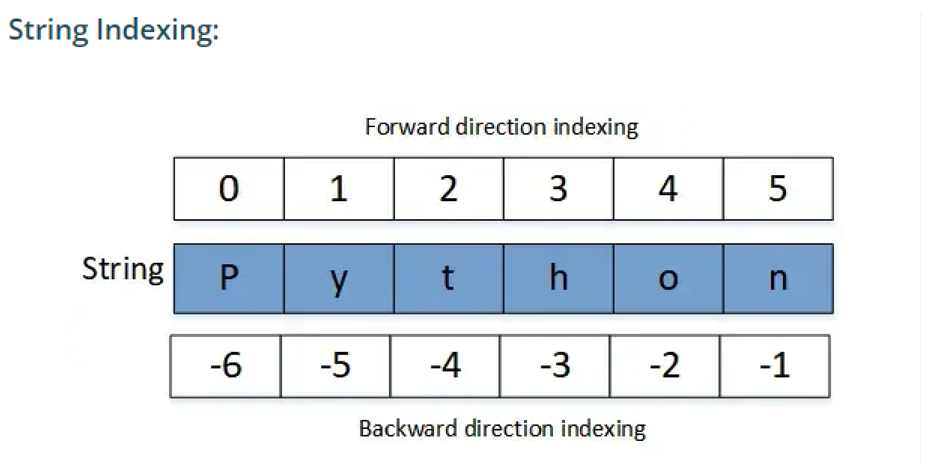
```
String with the use of Single Quotes:
I love my country

String with the use of Double Quotes:
I'm a citizen

String with the use of Triple Quotes:
I'm a citizen and I say "I love my country"

Creating a multiline String:
I love
 my
country
```

## Accessing the characters in python:

There are two methods of accessing the characters of string in python:

- Indexing
- Slicing

Strings are sequences of characters, and indexing allows us to access individual characters in the string. Indexing is done using square brackets [], and each character in the string is assigned a specific position, or index. Python supports both positive and negative indexing.



**Positive Indexing**

String: H e l l o

Index: 0 1 2 3 4

**Negative Indexing**

String:  H  e  l  l  o

Index:  -5 -4 -3 -2

**Some Examples:**

```
mystring = 'I am Nepali Citizen.'
print(mystring[3])
print(mystring[-2])
print(mystring[4000])
```

**Output:**

```
m
n
Traceback (most recent call last):
  File "g:\INTERN\06-Day6\indexing.py", line 4, in <module>
    print(mystring[4000])
          ~~~~~~~~^^^^^^
IndexError: string index out of range
```

**Note: While accessing an index out of the range will cause an IndexError as seen above. Only Integers are allowed to be passed as an index, float or other types that will cause a TypeError.**

**Program to print both positive and negative index of every character in string:**

```
my_string = "My country's name is Nepal"
i=0
print("---Index of every characters in both positive and
negative---")
for x in my_string:
     print(f"The character {x} is present at positive {i} index and
negative index {i - len(my_string)}")
    i += 1
```

**Output:**

```
---Index of every characters in both positive and negative---
The character M is present at positive 0 index and negative index
```

```
-26
The character y is present at positive 1 index and negative index
-25
The character   is present at positive 2 index and negative index
-24
The character c is present at positive 3 index and negative index
-23
The character o is present at positive 4 index and negative index
-22
The character u is present at positive 5 index and negative index
-21
The character n is present at positive 6 index and negative index
-20
The character t is present at positive 7 index and negative index
-19
The character r is present at positive 8 index and negative index
-18
The character y is present at positive 9 index and negative index
-17
The character ' is present at positive 10 index and negative index
-16
The character s is present at positive 11 index and negative index
-15
The character   is present at positive 12 index and negative index
-14
The character n is present at positive 13 index and negative index
-13
The character a is present at positive 14 index and negative index
-12
The character m is present at positive 15 index and negative index
-11
The character e is present at positive 16 index and negative index
```

```
-10
The character   is present at positive 17 index and negative index
-9
The character i is present at positive 18 index and negative index
-8
The character s is present at positive 19 index and negative index
-7
The character   is present at positive 20 index and negative index
-6
The character N is present at positive 21 index and negative index
-5
The character e is present at positive 22 index and negative index
-4
The character p is present at positive 23 index and negative index
-3
The character a is present at positive 24 index and negative index
-2
The character l is present at positive 25 index and negative index
-1
```

**Slicing:**

String slicing allows us to extract a portion (or "slice") of a string by specifying a start index, an end index, and an optional step value.

Example:

```
my_string = "Learning python is fun!"

print(my_string[0:5])
print(my_string[8:])
print(my_string[:7])
```

**Output:**

```
Learn
python is fun!
Learning
```

**Using negative Indexing:**

```
print(my_string[-4:])
print(my_string[-9:-5])
```

**Output:**

```
fun!
n is
```

**Note: If we give out of range, it returns an empty string.**

## Mathematical Operations for String

Two operators i.e. +, * can be used in the string in python, + is known as concatenation and * is known as string repetition.

**Concatenation:**
String concatenation refers to joining two or more strings together to form a single string. In Python, we use the + operator to concatenate strings.

**Both operands of the + operator must be strings. If we attempt to concatenate a string with a non-string type (like an integer), Python will raise a TypeError. We can convert other types to strings using str().**

Examples:

```
string1 = "Rajesh "
string2 = "Hamal"

print(string1 + string2)
print("Rajesh " + "Hamal")
```

Output:

```
Rajesh Hamal
Rajesh Hamal
```

**String Repetition in Python**

String repetition allows us to repeat a string multiple times. We use the * operator to achieve this. It takes a string and an integer as operands, and repeats the string the number of times specified by the integer.

```
str = 'hello '
print(str*5)
```

**Output:**

```
hello hello hello hello hello
```

## Membership Operators in Strings

Membership operators check whether a substring or character is present in a given string.

**in Operator:** Returns True if the specified character or substring exists in the string, otherwise returns False.

**not in Operator:** Returns True if the specified character or substring does not exist in the string, otherwise returns False.

Example:

```
string = "I love Python programming"

# Using 'in' operator
print("love" in string)
print("Java" in string)

# Using 'not in' operator
print("Python" not in string)
print("Java" not in string)
```

Output:

```
True
False
False
True
```

## Comparison Operators in Strings

Comparison operators compare two strings lexicographically (based on the ASCII values of characters). They check for equality, inequality, and the order of the strings.

- ==: Checks if two strings are equal.
- !=: Checks if two strings are not equal.
- >: Checks if the first string is lexicographically greater than the second.
- <: Checks if the first string is lexicographically smaller than the second.
- >=: Checks if the first string is greater than or equal to the second.
- <=: Checks if the first string is smaller than or equal to the second.

Example:

```
str1 = "apple"
str2 = "banana"
str3 = "apple"

# Equality and inequality
print(str1 == str3)   # Output: True
print(str1 != str2)   # Output: True

# Lexicographical comparison
print(str1  <  str2)       #  Output:  True  ("apple"  comes  before
"banana")
print(str1 > str2)    # Output: False
print(str1 >= str3)   # Output: True
print(str1 <= str3)   # Output: True
```

Output:

```
True
True
True
False
True
True
```

To remove spaces in python from string we have different methods like:

- strip(): Removes spaces from both the start and end of the string.

- lstrip(): Removes spaces from the start (left) only.
- rstrip(): Removes spaces from the end (right) only.
- replace(" ", ""): Removes all spaces from the string.
- split() and join(): Removes extra spaces and ensures there's only one space between words.

Example:

```
# Original string with leading and trailing spaces
text = "   Hello World   "

# Removing spaces from the left (beginning) only
left_trimmed = text.lstrip()
print("After lstrip(): '", left_trimmed, "'", sep='')

# Removing spaces from the right (end) only
right_trimmed = text.rstrip()
print("After rstrip(): '", right_trimmed, "'", sep='')

# Removing spaces from both sides (start and end)
fully_trimmed = text.strip()
print("After strip(): '", fully_trimmed, "'", sep='')
```

Output:

```
After lstrip(): 'Hello World   '
After rstrip(): '   Hello World'
After strip(): 'Hello World'
```

## Finding Substring in Python

There are 4 methods to find substring from a string in python.

They are:

- **find(substring):** Returns the lowest index of the substring or -1 if not found.
- **index(substring):** Returns the lowest index of the substring or raises ValueError if not found.
- **rfind(substring):** Returns the highest index of the substring or -1 if not found.
- **rindex(substring):** Returns the highest index of the substring or raises ValueError if not found.

**Note: They all return positive index of the substring.**

**Syntax:**

```
string.method(substring, start, end)


In place of method, write one of the above functions.
```

Example:

```
text = "Python programming is fun and programming is great"

# Using find()
find_position = text.find("programming")
print("Using find():", find_position)  # Output: 7
print("Position of 'Java':", text.find("Java"))  # Output: -1

# Using index()
index_position = text.index("programming")
print("\nUsing index():", index_position)  # Output: 7
# print(text.index("Java"))  # This line would raise ValueError if
uncommented

# Using rfind()
rfind_position = text.rfind("programming")
print("\nUsing rfind():", rfind_position)  # Output: 30
print("Last position of 'Java':", text.rfind("Java"))  # Output:
-1

# Using rindex()
rindex_position = text.rindex("programming")
print("\nUsing rindex():", rindex_position)  # Output: 30
# print(text.rindex("Java"))  # This line would raise ValueError
if uncommented
```

**Output:**

```
Using find(): 7
```

```
Position of 'Java': -1

Using index(): 7

Using rfind(): 30
Last position of 'Java': -1

Using rindex(): 30
```

## count() and replace() in python

### count()

The count() method in Python is used to determine how many times a specified substring appears within a string. It is a straightforward way to analyze the content of a string and can take optional arguments to specify the range for the search.

Syntax:

```
string.count(substring, start=..., end=...)
```

Example:

```
s = "learning python is fun"
print(s.count('i'))
```

Output: 2

### replace()

The replace() method is used to replace all occurrences of a specified substring with another substring. **Since strings are immutable in Python, this method does not modify the original string; instead, it returns a new string with the replacements made.**

Example:

```
text = "I love oranges and oranges are my favorite."
new_text = text.replace("oranges", "mangoes")
print(new_text)

print(id(text))
print(id(new_text))
```

Output:

```
I love mangoes and mangoes are my favorite.
2290542176912
2290542170480
```

## split() and join()

**split()**
The split() method is used to divide a string into a **list of substrings** based on a specified delimiter (separator). By default, it splits the string at whitespace.

**Example:**

```
text = "I love my country Nepal"
words = text.split()
print(words)
```

Output:
['I', 'love', 'my', 'country', 'Nepal']

## join()

The join() method is used to concatenate a list (or any iterable) of strings into a single string, with a specified separator between each element.

Example:

```
words = ['Python', 'is', 'beautiful']
```

```python
joined_text = " ".join(words)
print(joined_text)
```

Output: Python is beautiful

## Changing case in python

In Python, we can easily change the case of strings using several built-in methods. These methods allow us to convert strings to lowercase, uppercase, title case, and more. Here's an overview of the commonly used methods for case manipulation in strings:

- lower(): Converts the entire string to lowercase.
- upper(): Converts the entire string to uppercase.
- title(): Converts the first character of each word to uppercase.
- capitalize(): Converts the first character of the string to uppercase and the rest to lowercase.
- swapcase(): Swaps the case of all characters in the string.

Example:

```python
text = "Learning python is fun and interesting"

# Changing to lowercase
lowercase_text = text.lower()
print("Lowercase:", lowercase_text)

# Changing to uppercase
uppercase_text = text.upper()
print("Uppercase:", uppercase_text)

# Changing to title case
title_text = text.title()
print("Title Case:", title_text)

# Changing to capitalized
capitalized_text = text.capitalize()
print("Capitalized:", capitalized_text)
# Changing to swapped case
swapped_text = text.swapcase()
print("Swapped Case:", swapped_text)
```

Output:

```
Lowercase: learning python is fun and interesting
Uppercase: LEARNING PYTHON IS FUN AND INTERESTING
Title Case: Learning Python Is Fun And Interesting
Capitalized: Learning python is fun and interesting
Swapped Case: lEARNING PYTHON IS FUN AND INTERESTING
```

## Checking starting and ending part of string

We can check the starting and ending parts of a string using the **startswith()** and **endswith()** methods. These methods are useful for validating or filtering strings based on their prefixes and suffixes.

They return the boolean value i.e. True and False.

```
text = "Python programming is fun."
print(text.startswith("Python"))
print(text.startswith("Java"))
print(text.endswith("fun."))
print(text.endswith("Java"))
```

Output:

```
True
False
True
False
```