A string is a data structure in Python that represents a sequence of characters.

Strings can be created by enclosing a sequence of characters in either single or double quotes.

For example, the following are all valid strings:

- "This is a string."
- 'This is also a string.'

In [1]:
```
# With Single quotes

a = 'This is also a string.'
print(a)
```

```
This is also a string.
```

In [2]:
```
# With Double quotes

a = "Hello Pakistan"
print(a)
```

```
Hello Pakistan
```

In [3]:
```
# You can use three double quotes:


a = """Data science is the study of data to extract meaningful insights
for business. It is a multidisciplinary approach that combines principles
and practices from the fields of mathematics, statistics, artificial inte
and computer engineering to analyze large amounts of data."""

print(a)
```

```
"Data science is the study of data to extract meaningful insights
for business. It is a multidisciplinary approach that combines principles
and practices from the fields of mathematics, statistics, artificial inte
lligence,
and computer engineering to analyze large amounts of data.
```

In [4]:
```
 # Three single quotes:


a = '''Data science is the study of data to extract meaningful insights
for business. It is a multidisciplinary approach that combines principles
and practices from the fields of mathematics, statistics, artificial inte
and computer engineering to analyze large amounts of data.'''

print(a)
```

```
Data science is the study of data to extract meaningful insights
for business. It is a multidisciplinary approach that combines principles
and practices from the fields of mathematics, statistics, artificial inte
lligence,
and computer engineering to analyze large amounts of data.
```

strings are arrays in Python. In Python, a string is a sequence of characters, and an array is a data structure that can store a sequence of elements.

- Square brackets can be used to access elements of the string.[]

strings in Python can be indexed. Indexing allows you to access individual characters in a string by their position. The position of a character in a string is called its index. Inedex starts from 0.

- Get the character at position 1 (remember that the first character has the position 0):

In [5]:
```
a = "Hello, World!"
print(a[1])
```

```
e
```

In [6]:
```
a = "Hello, World!"
print(a[0])
```

```
H
```

In [7]:
```
string = "This is a string."

first_character = string[0]

print(first_character)
```

```
T
```

In [8]:
```
a = "Hello, World!"
print(a[7])
```

```
W
```

String Length

- To get the length of a string, use the len() function.

In [9]:
```
# The len() function returns the length of a string:
```

```
a = "NED Academy"
print(len(a))
```

```
11
```

In [10]:
```
string = "Python is a programming language."

length = len(string)

print(length)
```

```
33
```

Check String

- To check if a certain phrase or character is present in a string, we can use the keyword in.

In [11]:
```
txt = "The best things in life are free!"
print("free" in txt)
```

```
True
```

In [12]:
```
a = "The best things in life are free!"
if "free" in a:
    print("Yes, 'free' is present.")
```

```
Yes, 'free' is present.
```

In [13]:
```
a = "The best things in life are free!"
if "busy" in a:
    print("Yes, 'free' is present.")
else:
    print("NO", "Busy is not present")
```

```
NO Busy is not present
```

In [14]:
```
string = "Hello, world!"

character = "!"

is_present = character in string

print(is_present)
```

```
True
```

In [15]:
```
string = "Python is a programming language."

phrase = "Graphics"

is_present = phrase in string

print(is_present)
```

```
False
```

Check NOT

- To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

In [16]:
```
string = "Hello, world!"
print("NED Academy" not in string )
```

```
True
```

In [17]:
```
string = "Hello, world NED Academy"
print("NED Academy" not in string )
```

```
False
```

Slicing Strings

The slice() method takes three arguments: start, stop, and step. The start argument specifies the starting index of the substring,Slicing is the process of extracting a substring from a string. In Python, you can slice a string by using the slice() method or the [:] syntax.

In [18]:
```
# Get the characters from position 2 to position 5 (not included):

b = "Hello, World!"
print(b[2:5])
```

```
llo
```

In [19]:
```
# Slice From the Start

# By leaving out the start index, the range will start at the first chara

b = "Hello, World!"
print(b[:5])
```

```
Hello
```

In [20]:
```
string = "This is a string."

substring = slice(2, 6)

new_string = string[substring]

print(new_string)
```

```
is i
```

In [21]:
```
string = "This is a string."

# Extract the substring from the beginning to the second character
substring = slice(0, 2)

new_string = string[substring]
```

```
print(new_string)

# Extract the substring from the third to the end of the string
substring = slice(3, None)

new_string = string[substring]

print(new_string)

# Extract the substring from the beginning, skipping every other characte
substring = slice(0, None, 2)

new_string = string[substring]

print(new_string)
```

```
Th
s is a string.
Ti sasrn.
```

In [22]:
```
# Slice To the End

# Get the characters from position 2, and all the way to the end:

b = "Hello, World!"
print(b[2:])
```

```
llo, World!
```

In [23]:
```
string = "This is a string."

new_string = string[2:5]

print(new_string)
```

```
is
```

In [24]:
```
string = "This is a string."

new_string = string[2:6]

print(new_string)
```

```
is i
```

In [25]:
```
string = "This is a string."

# Extract the substring from the beginning to the second character
new_string = string[:2]

print(new_string)

# Extract the substring from the third to the end of the string
new_string = string[3:]

print(new_string)

# Extract the substring from the beginning, skipping every other characte
new_string = string[::2]

print(new_string)
```

```
Th
s is a string.
```

```
Ti sasrn.
```

Negative Indexing

Use negative indexes to start the slice from the end of the string:

In [26]:
```
string = "This is a string."

last_character = string[-1]

print(last_character)
```

```
.
```

In [27]:
```
string = "This is a string"

last_character = string[-1]

print(last_character)
```

```
g
```

In [28]:
```
string = "This is a string."

last_character = string[:-1]

print(last_character)
```

```
This is a string
```

In [29]:
```
string = "This is a string."

last_character = string[::-1]

print(last_character)
```

```
.gnirts a si sihT
```

In [30]:
```
string = "This is a string."

# Extract the second-last character
second_last_character = string[-2]

print(second_last_character)

# Extract the substring from the third to the last character
substring = string[2:-1]

print(substring)
```

```
g
is is a string
```

In [31]:
```
b = "Hello, World!"
print(b[-5:-2])
```

```
orl
```

This code will print the value None, because the string "This is a string." only has 17 characters, and the index -100 is outside the range of the string.

In [32]:
```
string = "This is a string."

last_character = string[-100]

print(last_character)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[32], line 3
      1 string = "This is a string."
----> 3 last_character = string[-100]
      5 print(last_character)

IndexError: string index out of range
```

Modify Strings

Upper()

- # The upper() method returns the string in upper case:

In [33]:
```
string = "ned academy"

new_string = string.upper()

print(new_string)
```

```
NED ACADEMY
```

In [34]:
```
a = "Hello, World!"
print(a.upper())
```

```
HELLO, WORLD!
```

In [35]:
```
original_string = "pakistan zindabad"
new_string = original_string.upper()
print(new_string)
```

```
PAKISTAN ZINDABAD
```

Lower Case

The lower() method returns the string in lower case:

In [36]:
```
string = "NED ACADEMY"

new_string = string.lower()

print(new_string)
```

```
ned academy
```

In [37]:
```
a = "HELLO WORLD!"
print(a.lower())
```

```
hello world!
```

In [38]:
```
original_string = "PAKISTAN ZINDABAD"
new_string = original_string.lower()
print(new_string)
```

```
pakistan zindabad
```

Remove Whitespace

There are two ways to remove whitespace in Python strings:

- Using the strip() method
- Using the replace() method

The strip() method removes whitespace from the beginning and end of a string. The replace() method replaces all whitespace characters in a string with a single character.

For example, the following code uses the strip() method to remove whitespace from the beginning and end of the string " This is a string. ":.

In [39]:
```
a = " Hello, World! "
print(a,"\n") # this is for printing actuall  line with spaces

print(a.strip()) # this line is after removing white spaces
```

```
 Hello, World!

Hello, World!
```

In [40]:
```
string = "   This is a string.   "
```

```
print(string,"\n")

new_string = string.strip()


print(new_string)
```

```
   This is a string.

This is a string.
```

The following code uses the replace() method to replace all whitespace characters in the string " This is a string. " with the character ' ':

In [41]:
```
string = "   This is a string.   "

print(string)

new_string = string.replace(" ", "'")

print(new_string)
```

```
   This is a string.
''''''This''is''a''string.''''''
```

Replace String

The replace() method replaces a string with another string:

In [42]:
```
a = "Hello, World!"
print(a.replace("H", "J"))
```

```
Jello, World!
```

The replace() method can be used to replace characters, substrings, or regular expressions. For example, the following code replaces all occurrences of the letter "a" with the letter "b" in the string "This is a string." and then replaces all occurrences of the substring "is" with the substring "was":

In [43]:
```
string = "This is a string."

new_string = string.replace("a", "b").replace("is", "was")

print(new_string)
```

```
Thwas was b string.
```

In [45]:
```
string = "This is a string."

new_string = string.replace("a", "b")

print(new_string)
```

```
This is b string.
```

Split String

The split() method returns a list where the text between the specified separator becomes the list items.

In [46]:
```
a = "Hello World!"
print(a.split()) # returns ['Hello', ' World!']
```

```
['Hello', 'World!']
```

In [47]:
```
string = "This is a string."

print(string.split())
```

```
['This', 'is', 'a', 'string.']
```

In [48]:
```
string = "This is a string."

substrings = string.split(" ")

print(substrings)
```

```
['This', 'is', 'a', 'string.']
```

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

In [49]:
```
# Merge variable a with variable b into variable c:

a = "Hello"
b = "World"
c = a + b
print(c)
```

```
HelloWorld
```

In [50]:
```
# Spaces between words


a = "NED"
b = "Academy"
c = a+" "+b
print(c)
```

```
NED Academy
```

In [51]:
```
string1 = "This is "
string2 = "a long string."

print(string1 + string2)
```

```
This is a long string.
```

In [52]:
```
string1 = "This is "
string2 = "a long string."

new_string = string1 + string2

print(new_string)
```

```
This is a long string.
```

In [53]:
```
string1 = "The number is "
string2 = "123"

new_string = string1 + string2

print(new_string)
```

```
The number is 123
```

In [54]:
```
# Integer and string data types can not be merged. sor for the concatenat


string1 = "The number is "
string2 = 123

new_string = string1 + string2

print(new_string)
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
Cell In[54], line 7
      4 string1 = "The number is "
      5 string2 = 123
----> 7 new_string = string1 + string2
      9 print(new_string)

TypeError: can only concatenate str (not "int") to str
```

In [55]:
```
# converting datatype here

string1 = "The number is "
string2 = str(123)

new_string = string1 + string2

print(new_string)
```

```
The number is 123
```

Count String

In [56]:
```
string = "This is a string."

substrings = string.count("a")

print(substrings)
```

```
1
```

In [57]:
```
string = "banana"

substring = "a"

number_of_occurrences = string.count(substring)

print(number_of_occurrences)
```

```
3
```

In [58]:
```
# Escape Characters

txt = "We are the so-called \"Vikings\" from the north."
finda =txt.find("are")
print(finda)
```

```
3
```

The format() method in Python is used to format strings. It takes a string as an argument and returns a new string with the specified formatting applied.

The formatting syntax is as follows:

"{placeholder} {placeholder} {placeholder}".format(value1, value2, value3)

In [59]:
```
# can only concatenate str (not "int") to str

age = 36
a = "My name is Ali, I am " + age
print(a)
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
```

```
Cell In[59], line 4
      1 # can only concatenate str (not "int") to str
      3 age = 36
----> 4 a = "My name is Ali, I am " + age
      5 print(a)

TypeError: can only concatenate str (not "int") to str
```

In [60]:
```
# The format() method takes the passed arguments, formats them, and place

age = 27
a = "My name is Ali, and I am {}"
print(a.format(age))
```

```
My name is Ali, and I am 27
```

In [61]:
```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

In [62]:
```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

```
I want to pay 49.95 dollars for 3 pieces of item 567.
```

In [63]:
```
name = "Ali"
age = 27

print("My name is {name} and I am {age} years old.".format(name=name,age=


#The {name} placeholder is replaced with the value of the name variable,
#and the {age} placeholder is replaced with the value of the age variable
```

```
My name is Ali and I am 27 years old.
```

In [64]:
```
# Formatting a string with numbers

print("The number is {number:05d}.".format(number=123456))

# Formatting a string with text

print("The name is {name:<10s}.".format(name="Ali"))

# Formatting a string with both numbers and text

print("The number is {number:05d} and the name is {name:<10s}.".format(nu
```

```
The number is 123456.
The name is Ali       .
The number is 12345 and the name is Ali       .
```

There are so many methods in string

Method Description

- capitalize() Converts the first character to upper case
- casefold() Converts string into lower case
- center() Returns a centered string
- count() Returns the number of times a specified value occurs in a string
- encode() Returns an encoded version of the string
- endswith() Returns true if the string ends with the specified value
- expandtabs() Sets the tab size of the string
- find() Searches the string for a specified value and returns the position of where it was found
- format() Formats specified values in a string
- format_map() Formats specified values in a string
- index() Searches the string for a specified value and returns the position of where it was found
- isalnum() Returns True if all characters in the string are alphanumeric
- isalpha() Returns True if all characters in the string are in the alphabet
- isascii() Returns True if all characters in the string are ascii characters
- isdecimal() Returns True if all characters in the string are decimals
- isdigit() Returns True if all characters in the string are digits
- isidentifier() Returns True if the string is an identifier
- islower() Returns True if all characters in the string are lower case
- isnumeric() Returns True if all characters in the string are numeric
- isprintable() Returns True if all characters in the string are printable
- isspace() Returns True if all characters in the string are whitespaces
- istitle() Returns True if the string follows the rules of a title
- isupper() Returns True if all characters in the string are upper case
- join() Joins the elements of an iterable to the end of the string
- ljust() Returns a left justified version of the string
- lower() Converts a string into lower case
- lstrip() Returns a left trim version of the string
- maketrans() Returns a translation table to be used in translations
- partition() Returns a tuple where the string is parted into three parts
- replace() Returns a string where a specified value is replaced with a specified value
- rfind() Searches the string for a specified value and returns the last position of where it was found
- rindex() Searches the string for a specified value and returns the last position of where it was found
- rjust() Returns a right justified version of the string
- rpartition() Returns a tuple where the string is parted into three parts
- rsplit() Splits the string at the specified separator, and returns a list
- rstrip() Returns a right trim version of the string
- split() Splits the string at the specified separator, and returns a list
- splitlines() Splits the string at line breaks and returns a list
- startswith() Returns true if the string starts with the specified value
- strip() Returns a trimmed version of the string
- swapcase() Swaps cases, lower case becomes upper case and vice versa
- title() Converts the first character of each word to upper case

- translate() Returns a translated string
- upper() Converts a string into upper case
- zfill() Fills the string with a specified number of 0 values at the beginning