String DataType

A string is a data structure in Python that represents a sequence of characters. It is an immutable data type, meaning that once you have created a string, you cannot change it. Strings are used widely in many different applications, such as storing and manipulating text data, representing names, addresses, and other types of data that can be represented as text.

Example : 'aditya , "lokhande"

Strings can be created using single qoutes or double qoutes.

Creating Strings

```
# Creating a String
# with single Quotes
String1 = 'Hello World'
print("String with the use of Single Quotes: ")
print(String1)

# output
String with the use of Single Quotes:
Hello World



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

# output
```

Saved successfully!

```
# Creating a String
# with triple Quotes
String1 = '''I'm a Geek and I live in a world of "Geeks"'''
print("\nString with the use of Triple Quotes: ")
print(String1)

# Creating String with triple
# Quotes allows multiple lines
String1 = '''Aditya
                  Neeti
                  Nidhi'''
print("Creating a multiline String: ")
print(String1)

# output
Creating a multiline String:
Aditya
Neeti
Nidhi
```

Accessing characters in Python String

- Using indexing method
- Using slice operator

--

Indexing Method

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.

+ve index : forward direction
-ve index : backward direction

[IMAGE]

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

```
s = "lokhande"

print(s)      #output : lokhande
print(s[0])   #output : l
print(s[-1])  #output : e
print(s[20])  #Error  : IndexError
print(s[1.2]) #Error  : TypeError

i=0
for x in s :
  print("The char at index +ve{} & -ve{} is :{}".format(i,i-len(s),x))
  i+=1
```

Python Slice Operator

To access a range of characters in the String, the method of slicing is used. Slicing in a String is done by using a Slicing operator (colon).

s[beginIndex:endIndex:Step]

```
s = "lokhande" # len(s) = 8

print(s)        #output : lokhande
print(s[0:8])   #output : lokhande
print(s[0:8:1]) #output : lokhande
print(s[::])    #output : lokhande
print(s[::-1])  #output : ednahkol

print(s[0:3])   #output : lok
```



```
print(s[0:8:2]) #output : lkad
print(s[-6:-1]) #output : khand
```

In Forward Direction

- default value for begin : 0
- default value for end : len(string)
- default value for step : 1

In Backward Direction

- default value for begin : -1
- default value for end : -len(string)+1
- default value for step : 1

Mathematical Operators In Strings

We can use + and * operators on string

- '+' : concatenation
  both the opearands should be string type
- '*' : repetition operator
  string * n ; n should be string type

```
s = 'aditya'

print(s+" lokhande") #output : aditya lokhande
print(s*3)           #output : adityaadiyaaditya
```

Some String Functions

```
s = " aditya lokhande "

# len() : finds length of string
```

```
print(len(s)) # output : 18

list = ["Hyderabad","Delhi","Mumbai"]
s    = " Mumbai "

# strip() : removes spaces from string
# lstrip() : removes spaces from the beginning
# rstrip() : removes spaces from the ending

if s in list:
  print("yes")
else:
  print("no")
# output : no

if s.strip() in list:
  print("yes")
else:
  print("no")
# output : yes

AUR BHI BOHOT SARE H PURE NHI LIKH RAHA


SARI RAAT SOYA NHI
SUBHAH MUJHE SONE DO.
KHWAAB PURE HUE NHI
NEEND PURI HONE DO
```

Comparison Of Strings

Ye Wali link se padh le

Saved successfully!

Finding Substring

- find()
- index()
- 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

--

find()

The find() method finds the first occurrence of the specified value.

The find() method returns -1 if the value is not found.

The find() method is almost the same as the index() method, the only difference is that the index() method raises an exception if the value is not found.

Syntax :

string.find(value, start, end)

Parameter Description

- value Required. The value to search for
- start Optional. Where to start the search. Default is 0
- end Optional. Where to end the search. Default is to the end of the string

```
# Where in the text is the first occurrence of the letter "e"?:

txt = "Hello, welcome to my world."
x = txt.find("e")
print(x)

# Where in the text is the first occurrence of the letter "e" when you only search between position 5 and 10?:

txt = "Hello, welcome to my world."
x = txt.find("e", 5, 10)
```

```
print(x)

# If the value is not found, the find() method returns -1, but the index() method will raise an exception:

txt = "Hello, welcome to my world."
print(txt.find("q"))
print(txt.index("q"))
```

--

index()

The index() method finds the first occurrence of the specified value.

The index() method raises an exception if the value is not found.

The index() method is almost the same as the find() method, the only difference is that the find() method returns -1 if the value is not found. (See example below)

Syntax :

string.index(value, start, end)

Parameter Description

- value Required. The value to search for
- start Optional. Where to start the search. Default is 0
- end Optional. Where to end the search. Default is to the end of the string

--

Replacing Substring

String replace() in Python returns a copy of the string where occurrences of a substring are replaced with another substring. In this article, we will see how to replace string Python.

Saved successfully!

string.replace(old, new, count)

Parameters:

- old – old substring you want to replace.
- new – new substring which would replace the old substring.
- count – (Optional) the number of times you want to replace the old substring with the new substring.
- Return Value : It returns a copy of the string where all occurrences of a substring are replaced with another substring.

```
string = "Good Morning"
new_string = string.replace("Good", "Great")
print(new_string)

# Output
Great Morning
```

--

Splitting A String

- split()
- rsplit() : Python String rsplit() method returns a list of strings after breaking the given string from the right side by the specified separator.

Python String split() method in Python split a string into a list of strings after breaking the given string by the specified separator.

Syntax :

str.split(separator, maxsplit)

Parameters :

- separator: This is a delimiter. The string splits at this specified separator. If is not provided then any white space is a separator.
- maxsplit: It is a number, which tells us to split the string into maximum of provided number of times. If it is not provided then the default is -1 that means there is no limit.
- Returns : Returns a list of strings after breaking the given string by the specified separator.

```
string = "one,two,three"
words = string.split(',')
```

```
print(words)

# Output:
['one', 'two', 'three']



word = 'geeks, for, geeks, pawan'

# maxsplit: 0
print(word.split(', ', 0))

# maxsplit: 4
print(word.split(', ', 4))

# maxsplit: 1
print(word.split(', ', 1))

# Output :
['geeks, for, geeks, pawan']
['geeks', 'for', 'geeks', 'pawan']
['geeks', 'for, geeks, pawan']
```

Joining Strings

The string join() method returns a string by joining all the elements of an iterable (list, string, tuple), separated by the given separator.

```
text = ['Python', 'is', 'a', 'fun', 'programming', 'language']

# join elements of text with space
print(' '.join(text))

                                                    language
```

Saved successfully!

Changing Case Of String

- upper() : string.lower()
- lower() : string.upper()
- swapcase() : string.swapcase()
- title() : string.title
- capitalize() : string.capitalize()

```
# Original sentence
sentence = "hello, world!"

# Convert all characters to uppercase
uppercase_sentence = sentence.upper()
print("Uppercase:", uppercase_sentence)  # Output: "HELLO, WORLD!"

# Convert all characters to lowercase
lowercase_sentence = sentence.lower()
print("Lowercase:", lowercase_sentence)  # Output: "hello, world!"

# Swap the case of each character
swapcase_sentence = sentence.swapcase()
print("Swapcase:", swapcase_sentence)  # Output: "HELLO, WORLD!"

# Convert the first character of each word to uppercase (title case)
title_sentence = sentence.title()
print("Title case:", title_sentence)  # Output: "Hello, World!"

# Convert the first character of the string to uppercase
capitalize_sentence = sentence.capitalize()
print("Capitalized:", capitalize_sentence)  # Output: "Hello, world!"
```

Checking Cases Of String

- islower(): Returns True if all characters are lowercase.
- isupper(): Returns True if all characters are uppercase.
- startswith(prefix[, start[, end]]): Returns True if the string starts with the specified prefix.
- endswith(suffix[, start[, end]]): Returns True if the string ends with the specified suffix.
- isalpha(): Returns True if all characters are alphabetic (letters).
- isdigit(): Returns True if all characters are digits.

- isnumeric(): Returns True if all characters are numeric.
- isdecimal(): Returns True if all characters are decimal digits (0-9).
- isalnum(): Returns True if all characters are alphanumeric (letters or digits).
- isspace(): Returns True if all characters are whitespace characters (spaces, tabs, newlines, etc.).
- isidentifier(): Returns True if the string is a valid Python identifier (variable name).
- istitle(): Returns True if the string is in title case (each word starts with an uppercase character).

```
# 1. islower()
text = "hello"
print(text.islower())  # Output: True

# 2. isupper()
text = "HELLO"
print(text.isupper())  # Output: True

# 3. startswith()
sentence = "Hello, world!"
print(sentence.startswith("Hello"))  # Output: True
print(sentence.startswith("hello"))  # Output: False

# 4. endswith()
sentence = "Hello, world!"
print(sentence.endswith("world!"))  # Output: True
print(sentence.endswith("World!"))  # Output: False

# 5. isalpha()
text = "Hello"
print(text.isalpha())  # Output: True

# 6. isdigit()
number = "12345"
print(number.isdigit())  # Output: True

# 7. isnumeric()
```



```
False

# 8. isdecimal()
number = "12345"
print(number.isdecimal())  # Output: True

# 9. isalnum()
mixed_text = "Hello123"
print(mixed_text.isalnum())  # Output: True

# 10. isspace()
text = "  \t  "
print(text.isspace())  # Output: True

# 11. isidentifier()
identifier = "my_variable"
print(identifier.isidentifier())  # Output: True

invalid_identifier = "123variable"
print(invalid_identifier.isidentifier())  # Output: False

# 12. istitle()
text = "This Is Title Case"
print(text.istitle())  # Output: True

text = "This is not Title Case"
print(text.istitle())  # Output: False
```

String Formatting

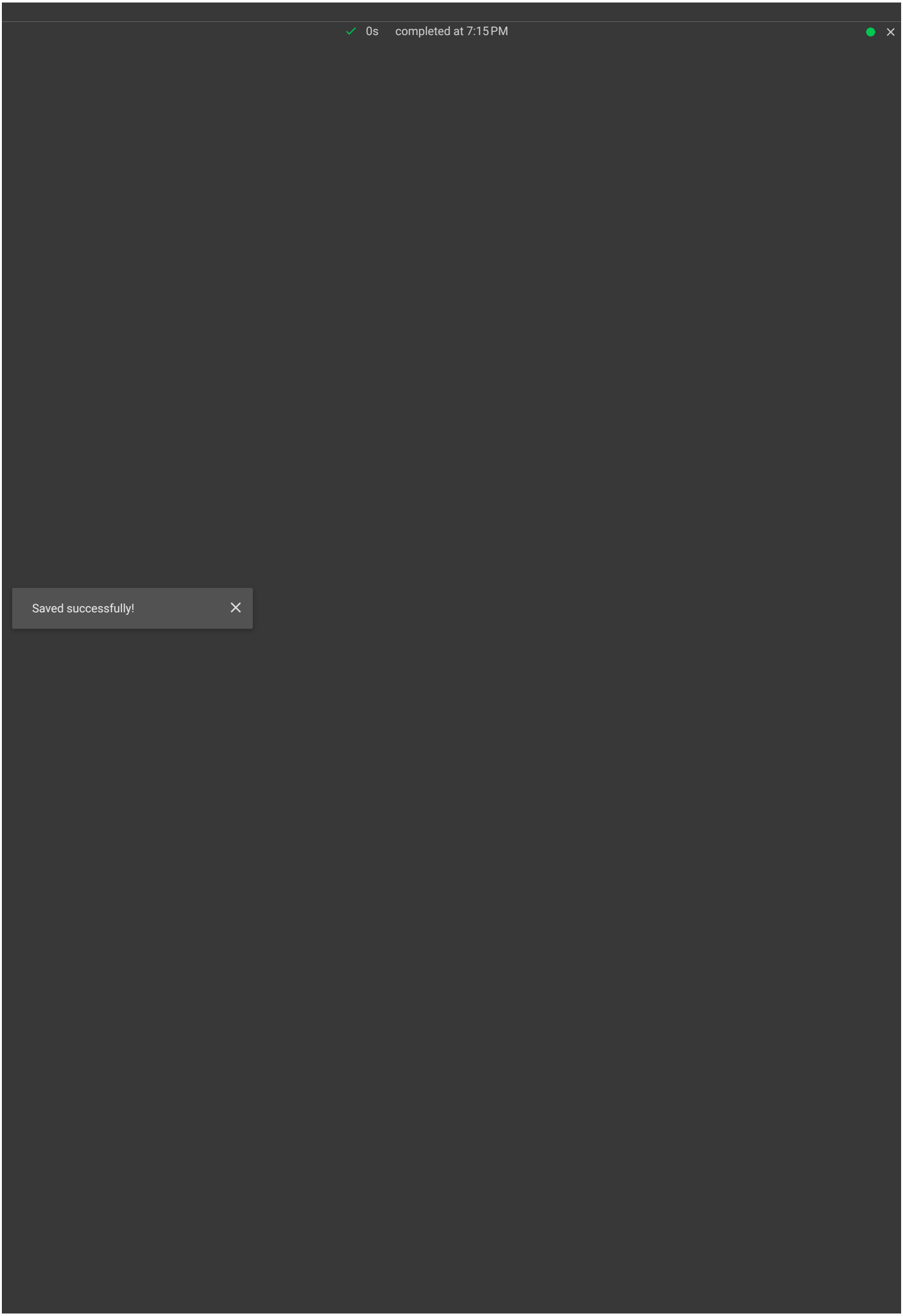
- String Concatenation: You can use the + operator to concatenate strings.
- %-formatting (Old Style): Use % operator with placeholders like %s for strings and %d for integers.
- str.format() (New Style): Use curly braces {} as placeholders for values that you provide using .format() method.
- f-strings (Formatted String Literals) - Python 3.6+: Use f before the string to create formatted strings with variables directly inside curly braces.
- Specifying Precision for Floating-Point Numbers: Control the precision of floating-point numbers using format specifications like "{:.2f}".
- Padding and Alignment: Align and pad strings using format specifications like "{:<10}" for left alignment and "{:>5}" for right alignment.
- Leading Zeros and Sign: Pad numbers with leading zeros using format specifications like "{:06}" and include the sign using "{:+}".

- Comma Separators for Large Numbers: Use commas to format large numbers for improved readability with format specifications like "{:,}".

Remember that the formatting options may vary depending on the chosen method. f-strings (formatted string literals) are recommended for Python 3.6 and above due to their simplicity and readability. However, older-style formatting methods can still be useful in certain situations or when working with older versions of Python. The choice of formatting style depends on your specific use case and the Python version you are working with.

```
# String Concatenation
first_name = "John"
last_name = "Doe"
full_name = first_name + " " + last_name
print("String Concatenation:", full_name)  # Output: "John Doe"

# %-formatting (Old Style)
name = "Alice"
age = 30
message = "My name is %s and I am %d years old." % (name, age)
print("%-formatting (Old Style):", message)  # Output: "My name is Alice and I am 30 years old."

# str.format() (New Style)
name = "Bob"
age = 25
message = "My name is {} and I am {} years old.".format(name, age)
print("str.format() (New Style):", message)  # Output: "My name is Bob and I am 25 years old."

# f-strings (Formatted String Literals) - Python 3.6+
name = "Charlie"
age = 22
message = f"My name is {name} and I am {age} years old."
print("f-strings (Formatted String Literals):", message)  # Output: "My name is Charlie and I am 22 years old."

# Specifying Precision for Floating-Point Numbers
pi = 3.141592653589793
formatted_pi = "{:.2f}".format(pi)
print("Specifying Precision for Floating-Point Numbers:", formatted_pi)  # Output: "3.14"

# Padding and Alignment
name = "John"
formatted_name = "{:<10}".format(name)
print("Padding and Alignment (Left):", "|{}|".format(formatted_name))  # Output: "|John      |"

age = 25
formatted_age = "{:>5}".format(age)
print("Padding and Alignment (Right):", "|{}|".format(formatted_age))  # Output: "|   25|"

# Leading Zeros and Sign
number = 42
formatted_number = "{:06}".format(number)
print("Leading Zeros:", formatted_number)  # Output: "000042"

number = -15
formatted_number = "{:+}".format(number)
print("Sign:", formatted_number)  # Output: "-15"

# Comma Separators for Large Numbers
population = 1000000
formatted_population = "{:,}".format(population)
print("Comma Separators for Large Numbers:", formatted_population)  # Output: "1,000,000"
```

Saved successfully!


0s completed at 7:15 PM
Saved successfully!