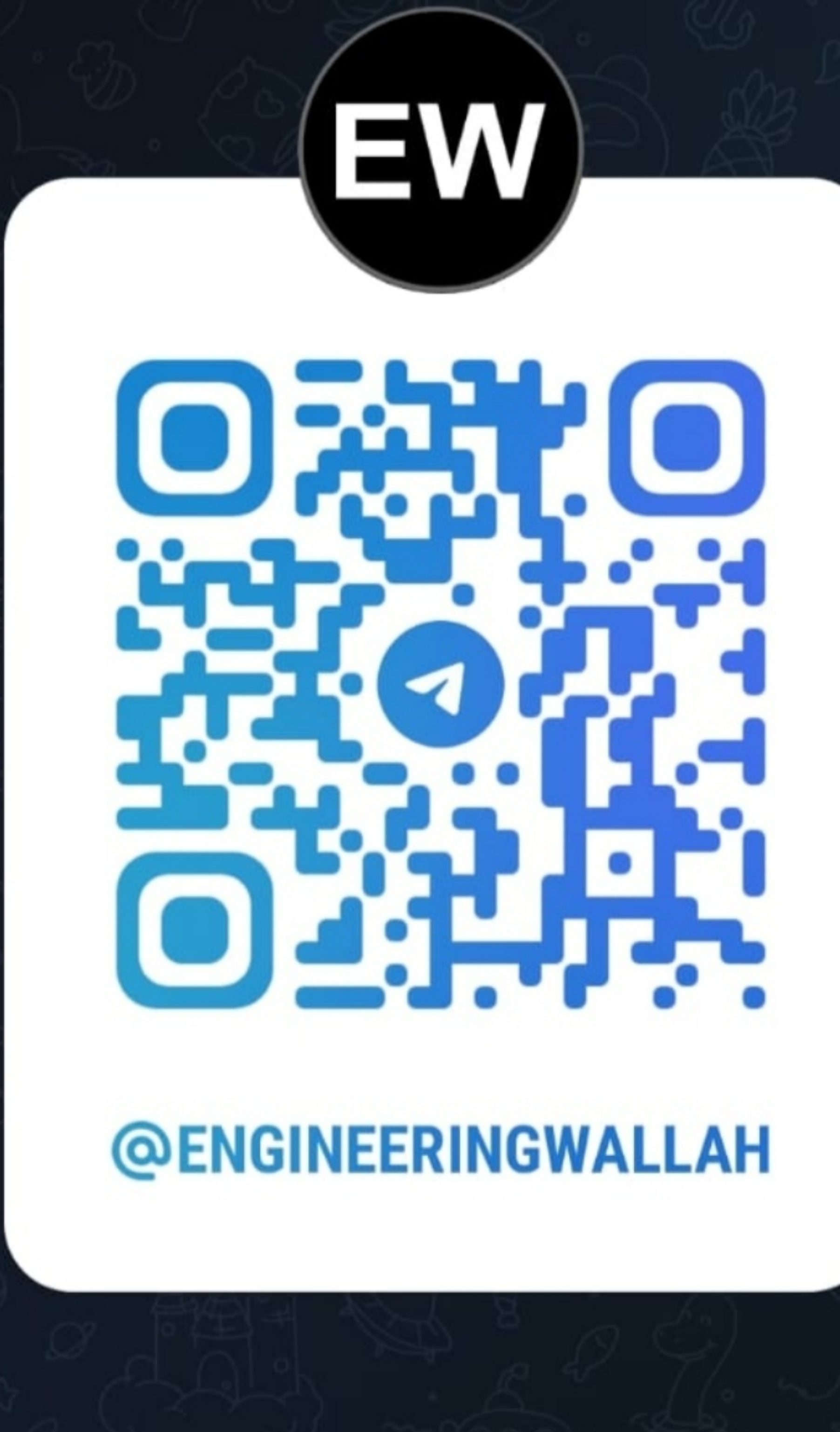
EW
@ENGINEERINGWALLAH

Unit IV

Python Strings Revisited

Strings

Python treats strings as contiguous series of characters delimited by single, double or even triple quotes. Python has a built-in string class named "str" that has many useful features. We can simultaneously declare and define a string by creating a variable of string type. This can be done in several ways which are as follows:

```
name = "India" graduate = 'N' country = name nationality = str("Indian")
```

Indexing: Individual characters in a string are accessed using the subscript ([ ]) operator. The expression in brackets is called an index. The index specifies a member of an ordered set and in this case it specifies the character we want to access from the given set of characters in the string.

The index of the first character is 0 and that of the last character is n-1 where n is the number of characters in the string. If you try to exceed the bounds (below 0 or above n-1), then an error is raised.

Strings

Traversing a String: **A string can be traversed by accessing character(s) from one index to another. For example, the following program uses indexing to traverse a string from first character to the last.**

Example:

```
message = "Hello!"
index = 0
for i in message:
print("message[", index, "] = ", i)
index += 1
```

OUTPUT

```
message[ 0 ] =  H
message[ 1 ] =  e
message[ 2 ] =  l
message[ 3 ] =  l
message[ 4 ] =  o
message[ 5 ] =  !
```

Concatenating, Appending and Multiplying Strings

Examples:

```
str1 = "Hello "
str2 = "World"
str3 = str1 + str2
print("The concatenated string is : ", str3)
```

OUTPUT

```
The concatenated string is :  Hello World
```

```
str = "Hello"
print(str * 3)
```

OUTPUT

```
Hello Hello Hello
```

```
str = "Hello, "
name = input("\n Enter your name : ")
str += name
str += ". Welcome to Python Programming."
print(str)
```

OUTPUT

```
Enter your name : Arnav
Hello, Arnav. Welcome to Python Programming.
```

Strings are Immutable

Python strings are immutable which means that once created they cannot be changed. Whenever you try to modify an existing string variable, a new string is created.

Example:

```
str1 = "Hello"
print("Str1 is : ", str1)
print("ID of str1 is : ", id(str1))

str2 = "World"
print("Str2 is : ", str2)

print("ID of str1 is : ", id(str2))
str1 += str2
print("Str1 after concatenation is : ", str1)
print("ID of str1 is : ", id(str1))
str3 = str1
print("str3 = ", str3)
print("ID of str3 is : ", id(str3))
```

OUTPUT

```
Str1 is :  Hello
ID of str1 is :  45093344
Str2 is :  World
ID of str1 is :  45093312
Str1 after concatenation is :  HelloWorld
ID of str1 is :  43861792
str3 =  HelloWorld
ID of str3 is :  43861792
```

String Formatting Operator

The % operator takes a format string on the left (that has %d, %s, etc) and the corresponding values in a tuple (will be discussed in subsequent chapter) on the right. The format operator, % allow users to construct strings, replacing parts of the strings with the data stored in variables. The syntax for the string formatting operation is:

"<Format>" % (<Values>)

Example:

```
name = "Aarish"
age = 8
print("Name = %s and Age = %d" %(name, age))
print("Name = %s and Age = %d" %("Anika", 6))
```

OUTPUT

```
Name = Aarish and Age = 8
Name = Anika and Age = 6
```

Built-in String Methods and Functions

Function	Usage	Example

`capitalize()`	This function is used to capitalize first letter of the string.	`str = "hello"`
		`print(str.capitalize())`
		OUTPUT
		`Hello`

`center(width, fillchar)`	Returns a string with the original string centered to a total of `width` columns and filled with `fillchar` in columns that do not have characters.	`str = "hello"`
		`print(str.center(10, '*'))`
		OUTPUT
		`**hello***`

`count(str, beg, end)`	Counts number of times str occurs in a string. You can specify beg as 0 and end as the length of the message to search the entire string or use any other value to just search a part of the string.	`str = "he"`
		`message = "helloworldhellohello"`
		`print(message. count (str,0, len (message)))`
		OUTPUT
		`3`

`endswith (suffix, beg, end)`	Checks if string ends with suffix; returns `True` if so and `False` otherwise. You can either set `beg = 0` and `end` equal to the length of the message to search entire string or use any other value to search a part of it.	`message = "She is my best friend"`
		`print(message.endswith("end", 0,len(message)))`
		OUTPUT
		`True`

Built-in String Methods and Functions

`find(str, beg, end)`	Checks if `str` is present in string. If found it returns the position at which `str` occurs in string, otherwise returns `-1`. You can either set `beg = 0` and `end` equal to the length of the message to search entire string or use any other value to search a part of it.	`message = "She is my best friend"`
		`print(message. find("my",0, len (message)))`
		OUTPUT
		`7`

`index(str, beg, end)`	Same as find but raises an exception if `str` is not found.	`message = "She is my best friend"`
		`print(message.index("mine", 0, len(message)))`
		OUTPUT
		`ValueError: substring not found`

`rfind(str, beg, end)`	Same as find but starts searching from the end.	`str = "Is this your bag?"`
		`print(str.rfind("is", 0, len(str)))`
		OUTPUT
		`5`

`rindex(str, beg, end)`	Same as `rindex` but start searching from the end and raises an exception if `str` is not found.	`str = "Is this your bag?"`
		`print(str.rindex("you", 0, len(str)))`
		OUTPUT
		`8`

Built-in String Methods and Functions

`isalnum()`	Returns True if string has at least 1 character and every character is either a number or an alphabet and False otherwise.	`message = "JamesBond007"`
		`print(message.isalnum())`
		OUTPUT
		`True`

`isalpha()`	Returns True if string has at least 1 character and every character is an alphabet and False otherwise.	`message = "JamesBond007"`
		`print(message.isalpha())`
		OUTPUT
		`False`

`isdigit()`	Returns True if string contains only digits and False otherwise.	`message = "007"`
		`print(message.isdigit())`
		OUTPUT
		`True`

`islower()`	Returns True if string has at least 1 character and every character is a lowercase alphabet and False otherwise.	`message = "Hello"`
		`print(message.islower())`
		OUTPUT
		`False`

`isspace()`	Returns True if string contains only whitespace characters and False otherwise.	`message = "  "`
		`print(message.isspace())`
		OUTPUT
		`True`

`isupper()`	Returns True if string has at least 1 character and every character is an upper case alphabet and False otherwise.	`message = "HELLO"`
		`print(message.isupper())`
		OUTPUT
		`True`

`len(string)`	Returns the length of the string.	`str = "Hello"`
		`print(len(str))`
		OUTPUT
		`5`

`ljust(width[, fillchar])`	Returns a string left-justified to a total of `width` columns. Columns without characters are padded with the character specified in the `fillchar` argument.	`str = "Hello"`
		`print(str.ljust(10, '*'))`
		OUTPUT
		`Hello*****`

`rjust(width[, fillchar])`	Returns a string right-justified to a total of `width` columns. Columns without characters are padded with the character specified in the `fillchar` argument.	`str = "Hello"`
		`print(str.rjust(10, '*'))`
		OUTPUT
		`*****Hello`

Slice Operation

A substring of a string is called a *slice*. The slice operation is used to refer to sub-parts of sequences and strings. You can take subset of string from original string by using [ ] operator also known as *slicing operator*.

Examples:

```
str = "PYTHON"
print("str[1:5] = ", str[1:5])   #characters starting at index 1 and extending up
to but not including index 5
print("str[:6] = ", str[:6])     # defaults to the start of the string
print("str[1:] = ", str[1:])      # defaults to the end of the string
print("str[:] = ", str[:])        # defaults to the entire string
print("str[1:20] = ", str[1:20])  # an index that is too big is truncated down to
length of the string
```

OUTPUT

```
str[1:5] =  YTHO
str[:6] =  PYTHON
str[1:] =  YTHON
str[:] =  PYTHON
str[1:20] =  YTHON
```

Programming Tip: Python does not have any separate data type for characters. They are represented as a single character string.

Specifying Stride while Slicing Strings

In the slice operation, you can specify a third argument as the *stride*, which refers to the number of characters to move forward after the first character is retrieved from the string. By default the value of stride is 1, so in all the above examples where he had not specified the stride, it used the value of 1 which means that every character between two index numbers is retrieved.

Examples:

```
str = "Welcome to the world of Python"
print("str[2:10] = ", str[2:10])    # default stride is 1
print("str[2:10:1] = ", str[2:10:1])   # same as stride = 1
print("str[2:10:2] = ", str[2:10:2])   # skips every alternate character
print("str[2:13:4] = ", str[2:13:4])   # skips every fourth character
```

OUTPUT

```
str[2:10] = lcome to
str[2:10:1] = lcome to
str[2:10:2] = loet
str[2:13:4] = le
```

ord() and chr() Functions

ord() function returns the ASCII code of the character and chr() function returns character represented by a ASCII number. Examples:

```
ch = 'R'
print(ord(ch))
```

OUTPUT

```
82
```

```
print(chr(82))
```

OUTPUT

```
R
```

```
print(chr(112))
```

OUTPUT

```
p
```

```
print(ord('p'))
```

OUTPUT

```
112
```

in and not in Operators

in and not in operators can be used with strings to determine whether a string is present in another string. Therefore, the in and not in operator are also known as membership operators.

Examples:

```
str1 = "Welcome to the world of Python
!!!"
str2 = "the"
if str2 in str1:
    print("Found")
else:
    print("Not Found")
```

OUTPUT

```
Found
```

```
str1 = "This is a very good book"
str2 = "best"
if str2 not in str1:
    print("The book is very good but it
may not be the best one.")
else:
    print ("It is the best book.")
```

OUTPUT

```
The book is very good but it may not be
the best one.
```

Comparing Strings

Operator	Description	Example
==	If two strings are equal, it returns True.	>>> "AbC" == "AbC" True

!= or <>	If two strings are not equal, it returns True.	>>> "AbC" != "Abc"
		True
		>>> "abc" <> "ABC"
		True

>	If the first string is greater than the second, it returns True.	>>> "abc" > "Abc" True
<	If the second string is greater than the first, it returns True.	>>> "abC" < "abc" True
>=	If the first string is greater than or equal to the second, it returns True.	>>> "aBC" >= "ABC" True
<=	If the second string is greater than or equal to the first, it returns True.	>>> "ABc" <= "ABc" True

Iterating String

String is a sequence type (sequence of characters).You can iterate through the string using for loop.

Examples:

```
str = "Welcome to Python"
for i in str:
print(i, end=' ')
```

OUTPUT

```
W e l c o m e  t o  P y t h o n
```

```
message = " Welcome to Python "
index = 0
while index < len(message):
    letter = message[index]
    print(letter, end=' ')
    index += 1
```

OUTPUT

```
W e l c o m e  t o  P y t h o n
```

The String Module

The string module consist of a number of useful constants, classes and functions (some of which are deprecated). These functions are used to manipulate strings.

Examples:

```
str = "Welcome to the world of Python"
print("Uppercase - ", str.upper())
print("Lowercase - ", str.lower())
print("Split - ", str.split())
print("Join - ", '-'.join(str.split()))
print("Replace - ", str.replace("Python", "Java"))
print("Count of o - ", str.count('o') )
print("Find of - ", str.find("of"))
```

OUTPUT

```
Uppercase -  WELCOME TO THE WORLD OF PYTHON
Lowercase -  welcome to the world of python
Split -  ['Welcome', 'to', 'the', 'world', 'of', 'Python']
Join -  Welcome-to-the-world-of-Python
Replace -  Welcome to the world of Java
Count of o -  5
Find of -  21
```

Programming Tip: A method is called by appending its name to the variable name using the period as a delimiter.

Working with Constants in String Module

You can use the constants defined in the string module along with the find function to classify characters. For example, if find(lowercase, ch) returns a value except -1, then it means that ch must be a lowercase character. An alternate way to do the same job is to use the in operator or even the comparison operation.

Examples:

```
# First Way
import string
print(string.find(string.
lowercase, 'g') != -1)

OUTPUT
True
```

```
# Second Way
import string
print('g' in string.
lowercase)

OUTPUT
True
```

```
# Third Way
import string
ch = 'g'
print('a' <= ch <= 'z')

OUTPUT
True
```