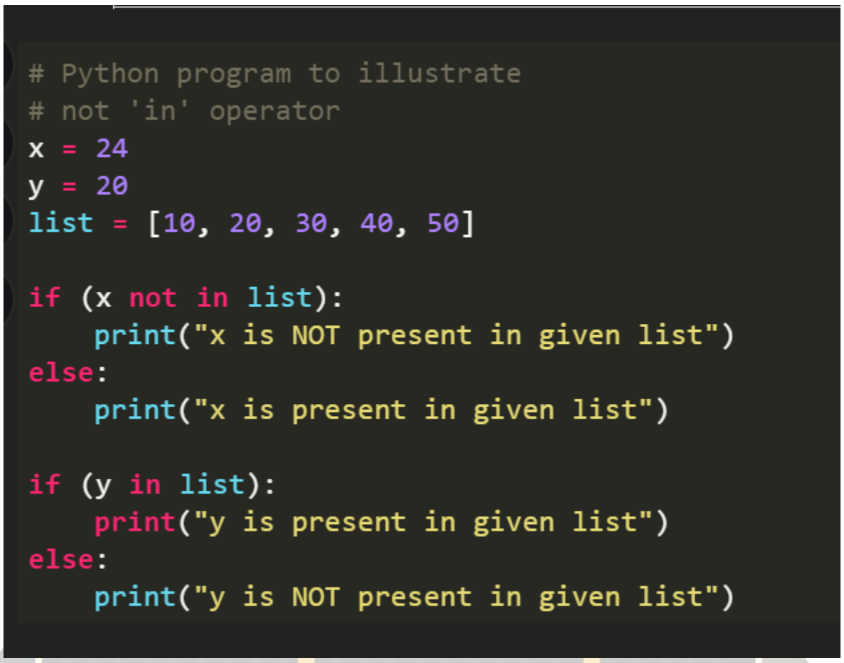
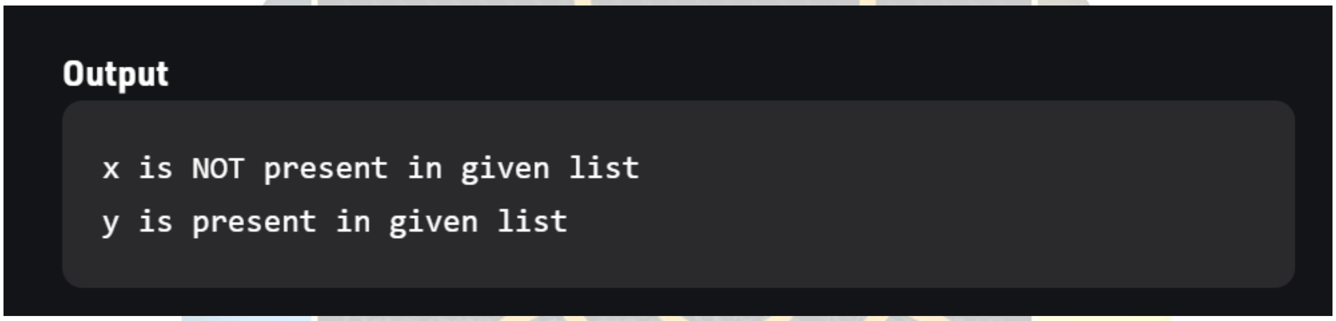
# String

- Traversing a string
  - Individual characters of a string are accessible through the unique index of each character.
  - Traversing refers to iterating through the elements of a string, one character at a time.
  - When you give a negative index, Python adds length of string to gets its forward index e.g., for a 5-lettered string 5, S[-1] will give S[-1+5] i.e., S[4] letter;
- String operators
  - Basic operators
    - String concatenation operator +
      - The + operator creates a new string by joining the two operand strings, e.g., "ice" + "cream" will result into "icecream".
      - We can add numbers using this operator but we cannot add a string and a number let's see how:
        '2' + '3' will result into '23'
        2 + 3 will result into 5
        but when we write '2' + 3 then it'll be error as we cannot add a string and a integer value together.
      - This shows that + operator has to have both operands of the same type either of number types or of string type. It can not work with one operand as string and one as a number.
    - String replication Operator *
      - The * operator when used with numbers (i.e., when both operands are numbers), it performs multiplication and returns the product of the two number operands.
      - To use a * operator with strings you need two types oof operands a string and a number I.e., as number*string or string*number.
      - 2 * 3 will result into 6
        'a' * 5 will result into 'aaaaa' (5 times a)
        6 * 'python' will result into 'pythonpythonpythonpythonpythonpython'
        but when we write 'a' * 'b' then it'll give error as we cannot replicate strings together.
      - The * operator has to either have both operands of the number types (for multiplication) or one string type and one number type (for replication).It cannot work with both operands of string types.
  - Membership Operators
    - **in** True if value is found in the sequence
    - **not in** True if value is not found in the sequence



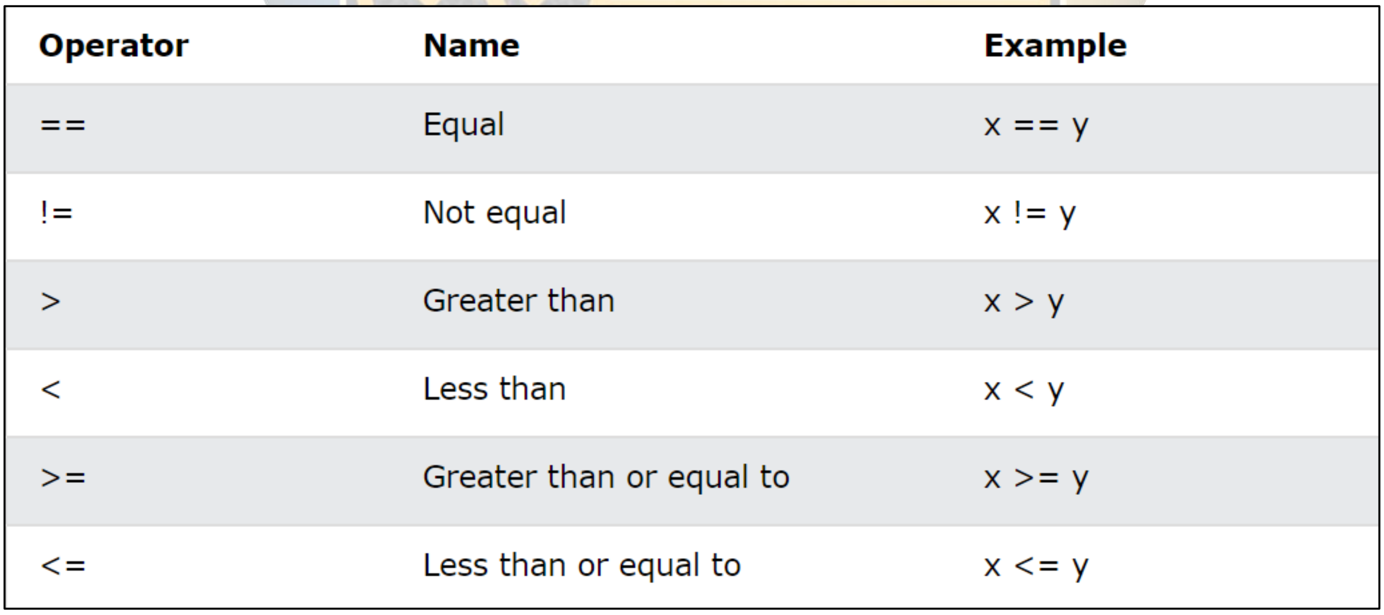
```python
# Python program to illustrate
# not 'in' operator
x = 24
y = 20
list = [10, 20, 30, 40, 50]

if (x not in list):
    print("x is NOT present in given list")
else:
    print("x is present in given list")

if (y in list):
    print("y is present in given list")
else:
    print("y is NOT present in given list")
```



**Output**

```
x is NOT present in given list
y is present in given list
```

- Comparison operator
  - It includes all relational operators
  - The comparison using these operators are based on the standard character-by-character comparison rules for Unicode (i.e., dictionary order)
  - To get Unicode character of a single character



| Operator | Name | Example |
|---|---|---|
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

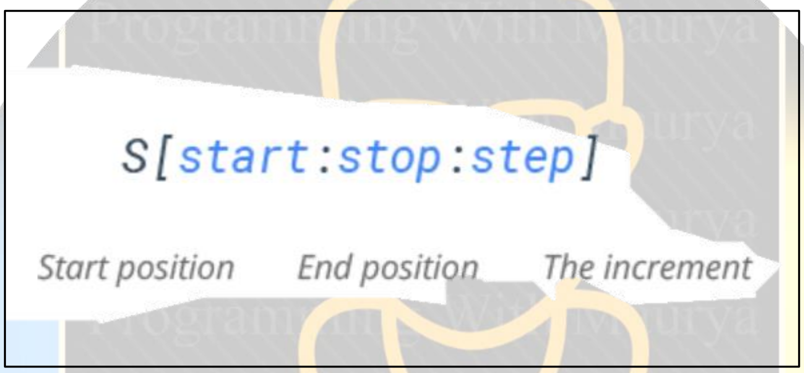
    ord(<single-character>)
    ord('A') will give 65

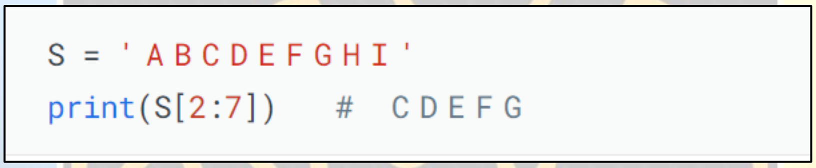
  - To get the ordinal value of the of a character

    chr(<int>)
    chr(65) will give 'A'

- String Slices
  - 'slice' means- 'a part of'

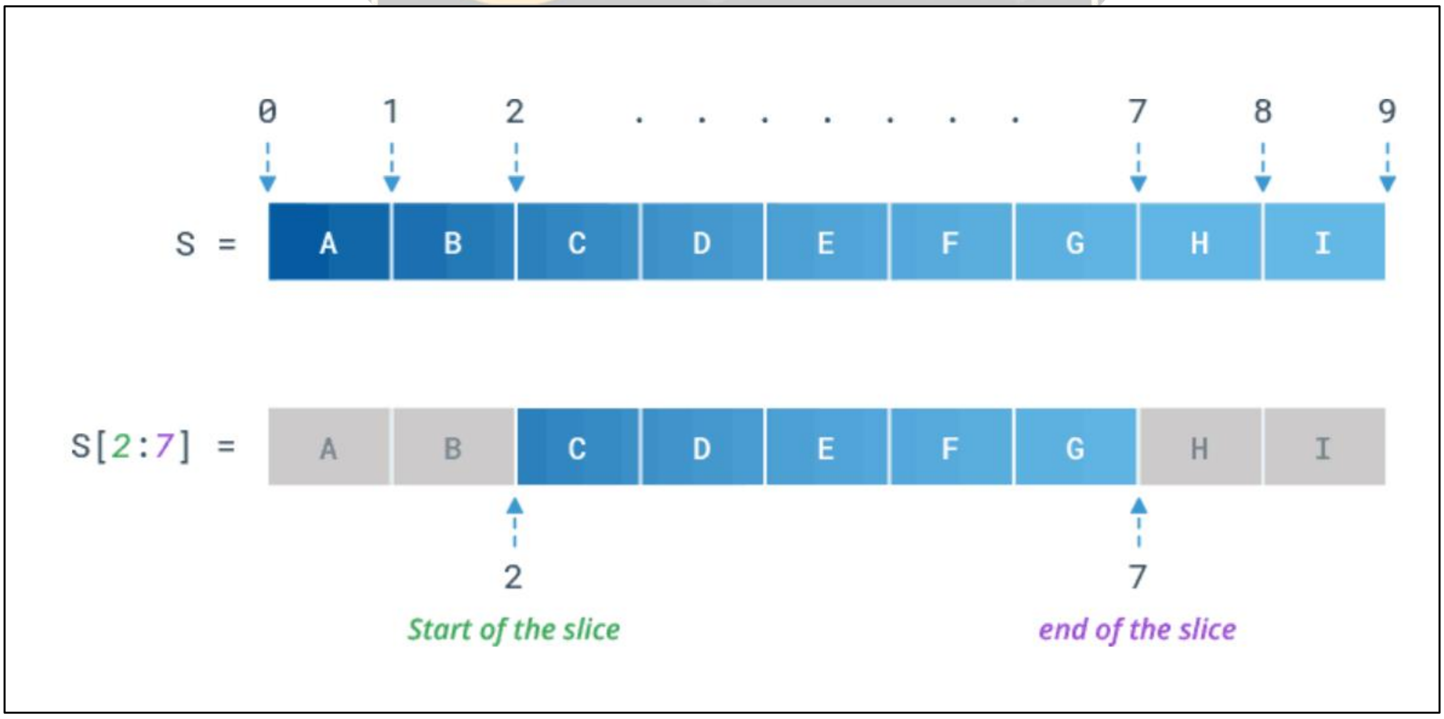
- Part of string containing some contiguous character from the string
- To access a range of characters in a string, you need to slice a string. One way to do this is to use the simple slicing operator :
- With this operator you can specify where to start the slicing, where to end and specify the step.
- Slicing a String
  - If S is a string, the expression S [ start : stop : step ] returns the portion of the string from index start to index stop, at a step size step.





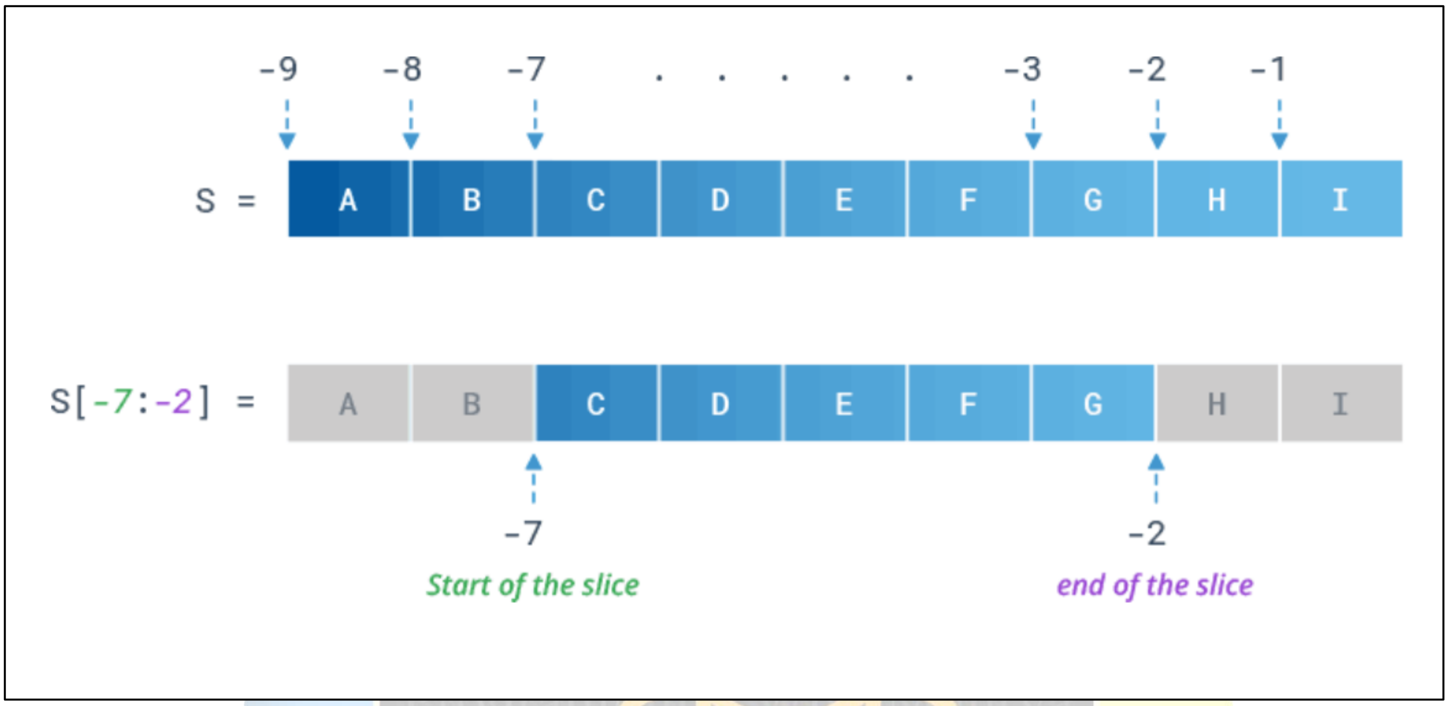
```
S = 'ABCDEFGHI'
print(S[2:7])    # CDEFG
```

- Examples



ⓘ Note that the item at index 7 'H' is not included.

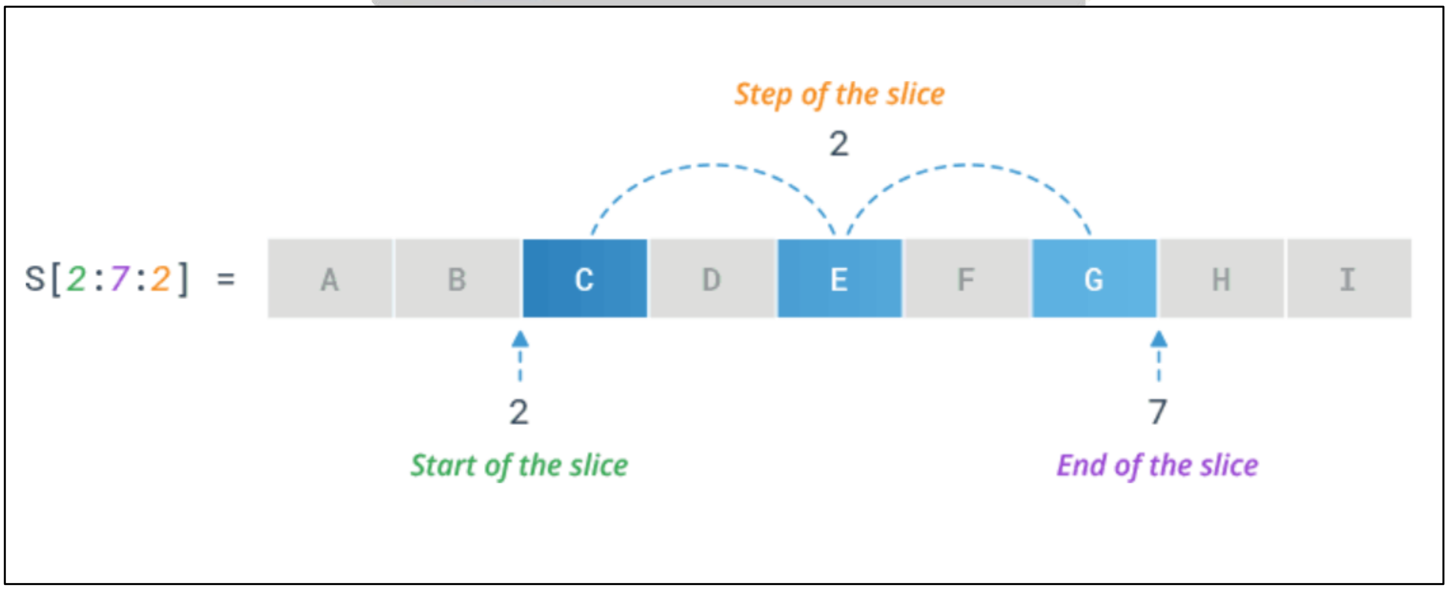
```
S = 'ABCDEFGHI'
print(S[-7:-2]) #  CDEFG
```



- Specify Step of the Slicing

  You can specify the step of the slicing using step parameter. The step parameter is optional and by defaul

```
# Return every 2nd item between position 2 to 7
S = 'ABCDEFGHI'
print(S[2:7:2]) #  CEG
```



- Slice at Beginning & End
  - Omitting the start index starts the slice from the index 0. Meaning, S[:stop] is equivalent to S[0:stop]

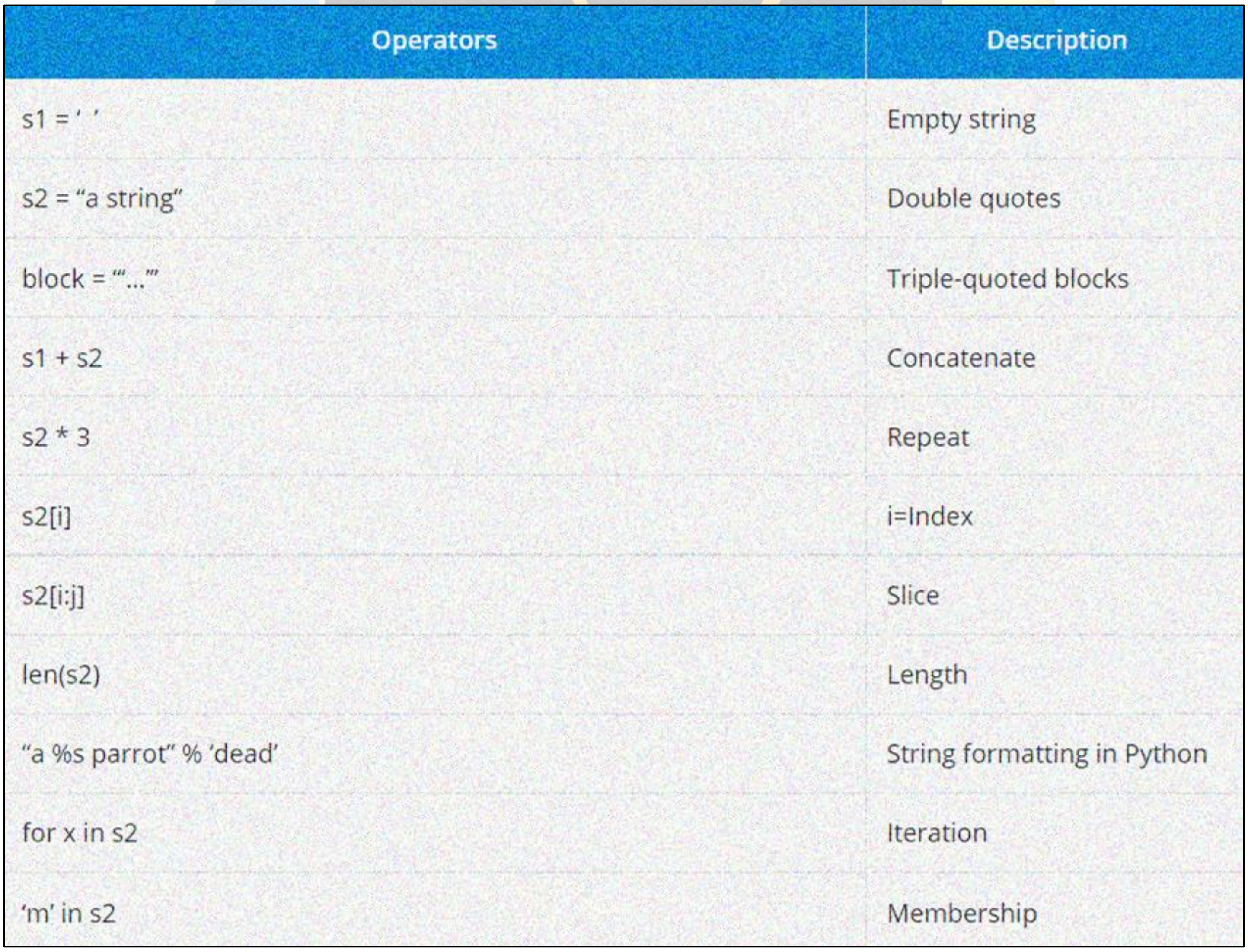
```
# Slice first three characters from the string
S = 'ABCDEFGHI'
print(S[:3])    #  ABC
```

```
# Slice last three characters from the string
S = 'ABCDEFGHI'
print(S[6:])    #  GHI
```

- Reverse a String
  - You can reverse a string by omitting both start and stop indices and specifying a step as -1.

```
S = 'ABCDEFGHI'
print(S[::-1])    #  IHGFEDCBA
```

**Overview**



| Operators | Description |
|---|---|
| s1 = ‘ ’ | Empty string |
| s2 = “a string” | Double quotes |
| block = ‘‘‘…’’’ | Triple-quoted blocks |
| s1 + s2 | Concatenate |
| s2 * 3 | Repeat |
| s2[i] | i=Index |
| s2[i:j] | Slice |
| len(s2) | Length |
| “a %s parrot” % ‘dead’ | String formatting in Python |
| for x in s2 | Iteration |
| ‘m’ in s2 | Membership |

| Method | Description |
|---|---|
| capitalize() | Converts the first character to upper case |
| casefold() | Converts string into lower case |
| count() | Returns the number of times a specified value occurs in a string |
| endswith() | Returns true if the string ends with the specified value |
| find() | Searches the string for a specified value and returns the position of where it was found |
| format() | 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 |
| isspace() | Returns True if all characters in the string are whitespaces |

| | |
|---|---|
| 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 |
| lower() | Converts a string into lower case |
| lstrip() | Returns a left trim version of the string |
| 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 |
| 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 |
| upper() | Converts a string into upper case |