SPLIT FUNCTION IN PYTHON-

In Python, the **split()** function is a built-in method that allows you to split a string into a list of substrings based on a specified delimiter. The **split()** function is available for string objects and takes an optional argument for the delimiter.

string.split(delimiter, maxsplit)

- **delimiter** (optional): It specifies the character or substring at which the string should be split. If no delimiter is provided, whitespace characters (spaces, tabs, and newlines) are used as the default delimiter.
- **maxsplit** (optional): It defines the maximum number of splits to be performed. If not provided, all occurrences of the delimiter are considered for splitting.

EXAMPLE 1 :

```
# Splitting a string using whitespace as the delimiter
text = "Hello World! Welcome to Python"
words = text.split()
print(words)
# Output: ['Hello', 'World!', 'Welcome', 'to', 'Python']

# Splitting a string using a specific delimiter
data = "apple,banana,orange,mango"
fruits = data.split(",")
print(fruits)
# Output: ['apple', 'banana', 'orange', 'mango']

# Splitting a string with a maximum number of splits
numbers = "1-2-3-4-5"
parts = numbers.split("-", 2)
print(parts)
```

```
# Output: ['1', '2', '3-4-5']
```

(In the above example- if we want to split numbers- count index wise- Now split the first two digits. i.e; [1,2] and rest will be same [3-4-5]

EXAMPLE 2 :

```
url = "https://www.example.com/page.html"
components = url.split("//")
print(components)
# Output: ['https:', 'www.example.com/page.html']
```

EXAMPLE 3 :

```
csv_data = "John,Doe,30,New York"
values = csv_data.split(",")
print(values)
# Output: ['John', 'Doe', '30', 'New York']
```

EXAMPLE 4 : (Parsing a data string)

```
date_str = "2023-06-09"
year, month, day = date_str.split("-")
print(year, month, day)
 # Output: 2023 06 09
```

EXAMPLE 5 : (Splitting a multiline string into lines)

```
text = "Line 1\nLine 2\nLine 3"
```

```
lines = text.split("\n")
print(lines)
# Output: ['Line 1', 'Line 2', 'Line 3']
```

EXAMPLE 6 : (Splitting a sentence into words and counting word occurrences)

```
sentence = "The quick brown fox jumps over the lazy dog"
words = sentence.split()
word_count = len(words)
print(words)
# Output: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
print(word_count)
 # Output: 9
```

RANGE FUNCTION-

In Python, the **range()** function is a built-in function that generates a sequence of numbers. It is commonly used in loops to iterate over a specific range of values. The **range()** function can take one, two, or three arguments and returns an iterable object that represents the sequence of numbers.

Here is the general syntax for the **range()** function:

```
range(start, stop, step)
```

- **start** (optional): It specifies the starting value of the sequence (inclusive). If not provided, the default value is 0.
- **stop** (required): It specifies the ending value of the sequence (exclusive). The **range()** function generates numbers up to, but not including, this value.
- **step** (optional): It specifies the step or increment between each number in the sequence. If not provided, the default step value is 1.

Generating a sequence of numbers up to a specific value :

```
for num in range(5):
    print(num)
# Output: 0, 1, 2, 3, 4
```

(In the above example, the range is upto 5 only so we don't generate 5)

Generating a sequence of numbers within a specific range :

```
for num in range(2, 7):
    print(num)
# Output: 2, 3, 4, 5, 6
```

(Here, the **range(2, 7)** generates numbers starting from 2 and up to (but not including 7). The loop iterates over each number in the range, and they are printed.)

Generating a sequence of numbers with a specific step size :

```
for num in range(1, 10, 2):
    print(num)
# Output: 1, 3, 5, 7, 9
```

(In this example, the **range(1, 10, 2)** generates numbers starting from 1 and up to (but not including) 10, with a step size of 2. The loop iterates over each number in the range, and they are printed.)

Creating a list of numbers using the 'list()' function and 'range()' :

```
numbers = list(range(1, 6))

print(numbers)

# Output: [1, 2, 3, 4, 5]
```

(Here, the **range(1, 6)** generates numbers from 1 to 5, and the **list()** function converts the range object into a list.

The **range()** function is particularly useful when you need to iterate over a specific range of values in a loop or create a sequence of numbers. By leveraging the **start**, **stop**, and **step** parameters, you can customize the behavior of the **range()** function to suit your specific requirements.)

IMPORTANT POINTS-

Example of Python range() function

- Python3

```
# print first 5 integers
# using python range() function
for i in range(5):
    print(i, end=" ")
print()
```

Output:

```
0 1 2 3 4
```

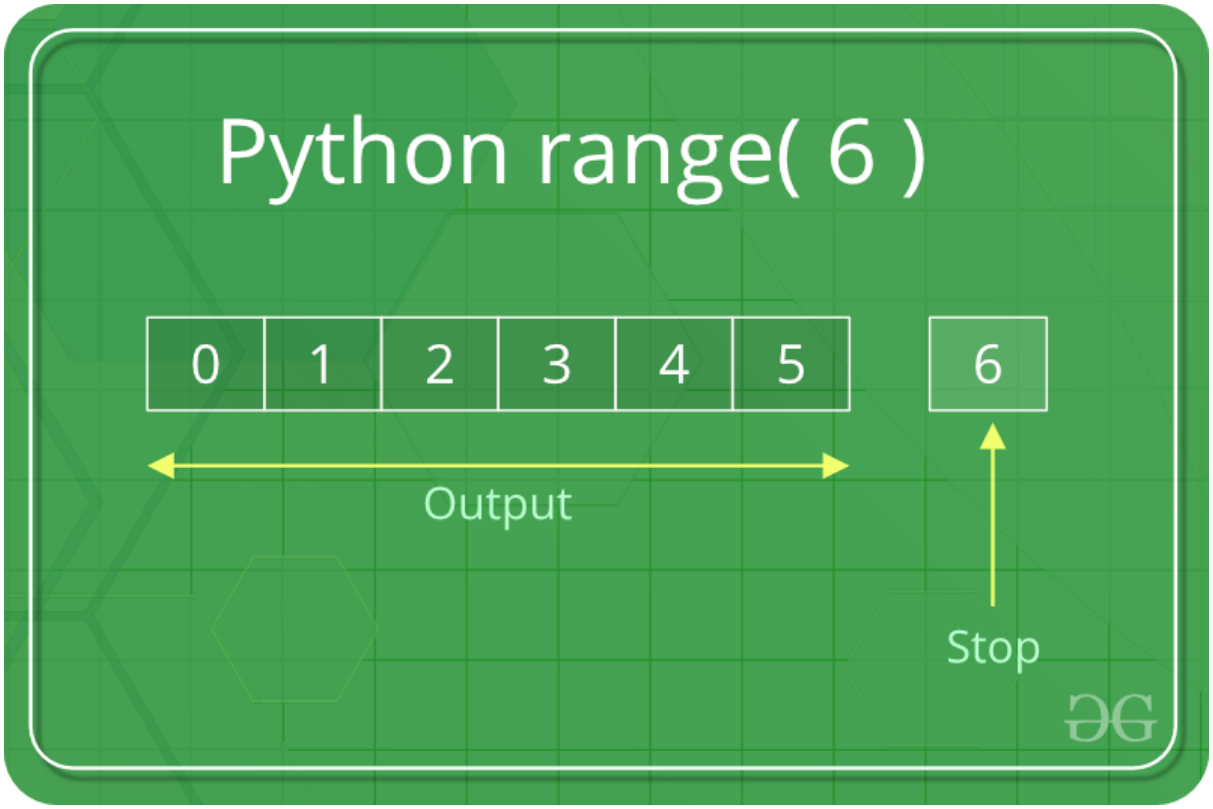
What is the use of the range function in Python

In simple terms, range() allows the user to generate a series of numbers within a given range. Depending on how many arguments the user is passing to the function, the user can decide where that series of numbers will begin and end, as well as how big the difference will be between one number and the next. Python range() function takes can be initialized in 3 ways.

- range (stop) takes one argument.
- range (start, stop) takes two arguments.
- range (start, stop, step) takes three arguments.

Python range (stop)

When the user call range() with one argument, the user will get a series of numbers that starts at 0 and includes every whole number up to, but not including, the number that the user has provided as the stop.



Python range visualization

Example: Demonstration of Python range (stop)

- Python3

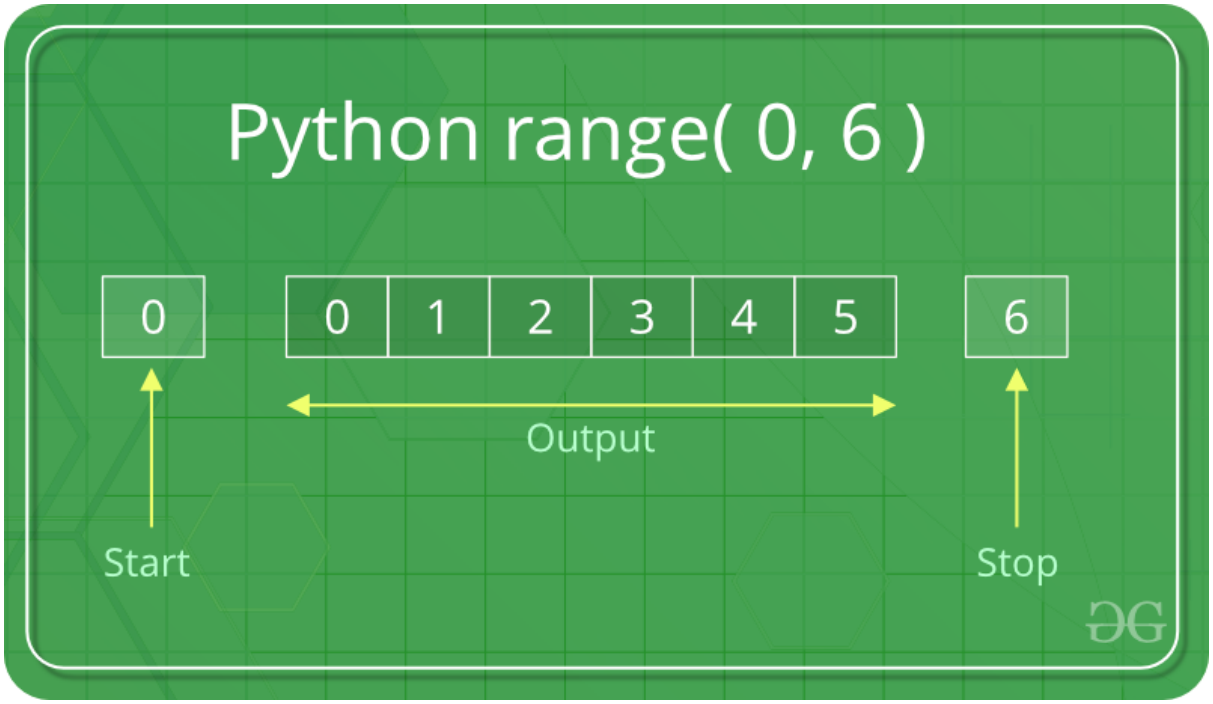
```
# printing first 6
# whole number
for i in range(6):
    print(i, end=" ")
print()
```

Output:

```
0 1 2 3 4 5
```

Python range (start, stop)

When the user call **range()** with two arguments, the user gets to decide not only where the series of numbers stops but also where it starts, so the user don't have to start at 0 all the time. Users can use range() to generate a series of numbers from X to Y using range(X, Y).



Python range visualization

Example: Demonstration of Python range (start, stop)

- Python3

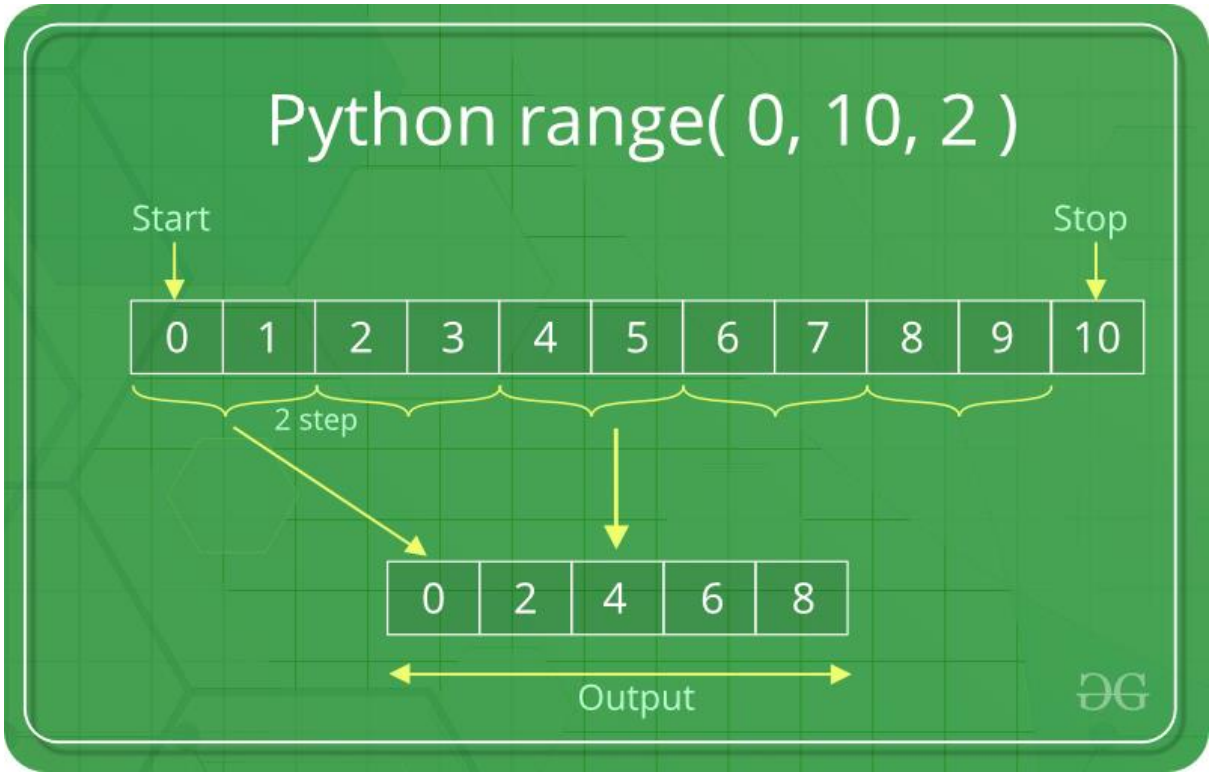
```
# printing a natural
# number from 5 to 20
for i in range(5, 20):
    print(i, end=" ")
```

Output:

```
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Python range (start, stop, step)

When the user call range() with three arguments, the user can choose not only where the series of numbers will start and stop, but also how big the difference will be between one number and the next. If the user doesn't provide a step, then range() will automatically behave as if the step is 1. In this example, we are printing even numbers between 0 and 10, so we choose our starting point from 0(start = 0) and stop the series at 10(stop = 10). For printing an even number the difference between one number and the next must be 2 (step = 2) after providing a step we get the following output (0, 2, 4, 8).



Python range visualization

Example: Demonstration of Python range (start, stop, step)

- Python3

```
for i in range(0, 10, 2):
    print(i, end=" ")
print()
```

Output:

```
0 2 4 6 8
```

Python range() with Examples

Example 1: Incrementing the range using a positive step

If a user wants to increment, then the user needs steps to be a positive number. (add 4- 0+4=4, 4+4=8, 8+4=12,..and so on)

- Python3

```
# incremented by 4
for i in range(0, 30, 4):
    print(i, end=" ")
print()
```

Output :

```
0 4 8 12 16 20 24 28
```

Example 2: Python range() using negative step

If a user wants to decrement, then the user needs steps to be a negative number. (subtract 25-2=23, 23-2=21, 21-2=19,..and so on)

- Python3

```
# incremented by -2
for i in range(25, 2, -2):
    print(i, end=" ")
print()
```

Output :

```
25 23 21 19 17 15 13 11 9 7 5 3
```

Example 3: Python range() with float

Python range() function doesn't support the float numbers. i.e. user cannot use floating-point or non-integer numbers in any of its argument. Users can use only integer numbers.

- Python3

```
# using a float number
for i in range(3.3):
    print(i)
```

Output :

```
for i in range(3.3):

TypeError: 'float' object cannot be interpreted as an integer
```

Example 4: Concatenation of two range() functions using itertools chain() method

The result from two range() functions can be concatenated by using the chain() method of itertools module. The chain() method is used to print all the values in iterable targets one after another mentioned in its arguments.

- Python3

```
from itertools import chain

# Using chain method
print("Concatenating the result")
res = chain(range(5), range(10, 20, 2))
```

```
for i in res:
    print(i, end=" ")
```

Output:

```
Concatenating the result

0 1 2 3 4 10 12 14 16 18
```

Example 5: Accessing range() with an index value

A sequence of numbers is returned by the range() function as its object that can be accessed by its index value. Both positive and negative indexing is supported by its object.

- Python3

```
ele = range(10)[0]
print("First element:", ele)

ele = range(10)[-1]
print("\nLast element:", ele)

ele = range(10)[4]
print("\nFifth element:", ele)
```

Output:

```
First element: 0


Last element: 9


Fifth element: 4
```

Some Important points to remember about the Python range() function:

- range() function only works with the integers, i.e. whole numbers.
- All arguments must be integers. Users can not pass a string or float number or any other type in a **start**, **stop** and **step** argument of a range().
- All three arguments can be positive or negative.
- The **step** value must not be zero. If a step is zero, python raises a ValueError exception.
- range() is a type in Python
- Users can access items in a range() by index, just as users do with a list: