

PYTHON PROGRAMMING LANGUAGE

ASSIGNMENT

OPERATORS

Operators are the symbols used to perform a specific operation on different values and variables. These values and variables are considered as the Operands, on which the operator is applied. Operators serve as the foundation upon which logic is constructed in a program in a particular programming language. In every programming language, some operators perform several tasks.

Arithmetic Operators

- Python Arithmetic Operators are used on two operands to perform basic mathematical operators like addition, subtraction, multiplication, and division.
- There are different types of arithmetic operators available in Python including the '+' operator for addition, '-' operator for subtraction, '*' for multiplication, '/' for division, '%' for modulus, '**' for exponent and '//' for floor division.

Example 1: Using all Arithmetic Operators:

Output:

The screenshot shows a Jupyter Notebook environment. On the left, the code editor displays a Python script named 'main.py' with the following content:

```
1 a = 46 # Initializing the value of a
2 b = 4 # Initializing the value of b
3
4 print("For a =", a, "and b =", b, "\nCalculate the
   following:")
5
6 # Printing different results
7 print('1. Addition of two numbers: a + b =', a + b)
8 print('2. Subtraction of two numbers: a - b =', a - b)
9 print('3. Multiplication of two numbers: a * b =', a * b)
10 print('4. Division of two numbers: a / b =', a / b)
11 print('5. Floor division of two numbers: a // b =', a // b)
12 print('6. Remainder of two numbers: a mod b =', a % b)
13 print('7. Exponent of two numbers: a ** b =', a ** b) # Fixed exponentiation operator
14
```

On the right, the 'Console' tab shows the output of the executed code. The output includes:
For a = 46 and b = 4
Calculate the following:
1. Addition of two numbers: a + b = 50
2. Subtraction of two numbers: a - b = 42
3. Multiplication of two numbers: a * b = 184
4. Division of two numbers: a / b = 11.5
5. Floor division of two numbers: a // b = 11
6. Remainder of two numbers: a mod b = 2
7. Exponent of two numbers: a ** b = 4477456

Comparison Operators

Python Comparison operators are mainly used for the purpose of comparing two values or variables (operands) and return a Boolean value as either True or False accordingly. There are various types of comparison operators available in Python including the '==', '!=', '<=', '>=', '<', and '>'.

Example 2: Using all Comparison Operators:

Output:

The screenshot shows a Python code editor on Replit. The code in main.py initializes variables a=46 and b=4, then prints various comparison results. The console output shows the results for each comparison operator.

```
a = 46 # Initializing the value of a
b = 4 # Initializing the value of b
print("For a =", a, "and b =", b, "\nCheck the following:")
# Printing different results
print('1. Check if two numbers are equal:', a == b)
print('2. Check if two numbers are not equal:', a != b)
print('3. a is less than or equal to b:', a <= b)
print('4. a is greater than or equal to b:', a >= b)
print('5. a is greater than b:', a > b)
print('6. a is less than b:', a < b)
```

Console Output:

```
For a = 46 and b = 4
Check the following:
1. Check if two numbers are equal: False
2. Check if two numbers are not equal: True
3. a is less than or equal to b: False
4. a is greater than or equal to b: True
5. a is greater than b: True
6. a is less than b: False
```

Assignment Operators

Using the assignment operators, the right expression's value is assigned to the left operand. Python offers different assignment operators to assign values to the variable. These assignment operators include '=', '+=' , '-=' , '*=' , '/=' , '%=' , '//=' , '**=' , '&=' , '|=' , '^=' , '>>=' , and '<<='.

Example 3: Using all Assignment Operators:

Output:

The screenshot shows a Python code editor on Replit demonstrating various assignment operators. The code initializes a=34 and b=6, then uses augmented assignment operators to perform addition, subtraction, multiplication, division, modulus, power, bitwise AND, OR, XOR, left shift, and right shift operations.

```
a = 34 # Initialize the value of a
b = 6 # Initialize the value of b
# Using augmented assignment operators
a += b
print('a += b:', a) # a = 34 + 6
a -= b
print('a -= b:', a) # a = 40 - 6
a *= b
print('a *= b:', a) # a = 34 * 6
a /= b
print('a /= b:', a) # a = 204 / 6
a %= b
print('a %= b:', a) # a = 34 % 6
a **= b
print('a **= b:', a) # a = 4 ** 6
a // b
print('a // b:', a) # a = 4096 // 6
```

Console Output:

```
a += b: 40
a -= b: 34
a *= b: 204
a /= b: 34.0
a %= b: 4.0
a **= b: 4096.0
a // b: 682.0
```

Bitwise Operators

The two operands' values are processed bit by bit by the bitwise operators. There are various Bitwise operators used in Python, such as bitwise OR (`|`), bitwise AND (`&`), bitwise XOR (`^`), negation (`~`), Left shift (`<>`). Consider the case below.

Example 4: Using all Bitwise Operators:

Output:

A screenshot of the Replit IDE interface. On the left, the code editor shows `main.py` with the following content:

```
1 # Initializing values
2 a = 7 # Binary: 0111
3 b = 6 # Binary: 0110
4
5 # Performing bitwise operations
6 print("a & b =", a & b) # 0111 & 0110 = 0010 (Decimal 6)
7 print("a | b =", a | b) # 0111 | 0110 = 0111 (Decimal 7)
8 print("a ^ b =", a ^ b) # 0111 ^ 0110 = 0001 (Decimal 1)
9 print("~a =", ~a) # ~0111 = -8 (Two's complement: -(7+1))
10
11 # Additional bitwise operations
12 x = 5 # Binary: 0101
13 y = 8 # Binary: 1000
14
15 print("x | y =", x | y) # 0101 | 1000 = 1101 (Decimal 13)
16 print("x & y =", x & y) # 0101 & 1000 = 0000 (Decimal 0)
17 print("x ^ y =", x ^ y) # 0101 ^ 1000 = 1101 (Decimal 13)
18 print("~x =", ~x) # ~0101 = -(5+1) = -6
19
```

The right side shows the `Console` tab with the output of the code:

```
a & b = 6
a | b = 7
a ^ b = 1
~a = -8
x | y = 13
x & y = 0
x ^ y = 13
~x = -6
```

Example 5: Using all Bitwise Operators:

Output:

A screenshot of the Replit IDE interface. On the left, the code editor shows `main.py` with the following content:

```
1 a = 7 # Initializing the value of a
2 b = 8 # Initializing the value of b
3
4 # Printing different results
5 print('a & b :', a & b) # Bitwise AND
6 print('a | b :', a | b) # Bitwise OR
7 print('a ^ b :', a ^ b) # Bitwise XOR
8 print('~a :', ~a) # Bitwise NOT (Two's complement)
9 print('a << b :', a << b) # Left shift (Multiplication by 2^b)
10 print('a >> b :', a >> b) # Right shift (Division by 2^b, integer result)
11
```

The right side shows the `Console` tab with the output of the code:

```
a & b : 0
a | b : 15
a ^ b : 15
~a : -8
a << b : 1792
a >> b : 0
```

Logical Operators

The assessment of expressions to make decisions typically uses logical operators. Python offers different types of logical operators such as and, or, and not. In the case of the logical AND, if the first one is 0, it does not depend upon the second one. In the case of the logical OR, if the first one is 1, it does not depend on the second one.

Example 6: Using all Logical Operators:

Output:

The screenshot shows a Jupyter Notebook interface. The code cell contains the following Python script:

```
a = 7 # Initializing the value of a
# Printing different results
print("For a = 7, checking whether the following conditions are True or False:")
print('"a > 5 and a < 7" =>', a > 5 and a < 7) # False (7 is not < 7)
print('"a > 5 or a < 7" =>', a > 5 or a < 7) # True (7 > 5)
print('"not (a > 5 and a < 7)" =>', not (a > 5 and a < 7)) # True (negation of False)
```

The output cell shows the results of the print statements:

```
For a = 7, checking whether the following conditions are True or False:
"a > 5 and a < 7" => False
"a > 5 or a < 7" => True
"not (a > 5 and a < 7)" => True
```

Membership Operators

We can verify the membership of a value inside a Python data structure using the Python membership operators. The result is said to be true if the value or variable is in the sequence (list, tuple, or dictionary); otherwise, it returns false.

Example 7: Using all Membership Operators:

Output:

The screenshot shows a Jupyter Notebook interface. The code cell contains the following Python script:

```
# Initializing a list
myList = [12, 22, 28, 35, 42, 49, 54, 65, 92, 103, 245, 874]
# Initializing x and y with some values
x = 31
y = 28
# Printing the given list
print("Given List:", myList)
# Checking if x is present in the list or not
if x not in myList:
    print("x =", x, "is NOT present in the given list.")
else:
    print("x =", x, "is present in the given list.")
# Checking if y is present in the list or not
if y in myList:
    print("y =", y, "is present in the given list.")
else:
    print("y =", y, "is NOT present in the given list.)
```

The output cell shows the results of the print statements:

```
Given List: [12, 22, 28, 35, 42, 49, 54, 65, 92, 103, 245, 874]
x = 31 is NOT present in the given list.
y = 28 is present in the given list.
```

Identity Operators

Python offers two identity operators as `is` and `is not`, that are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

Example 8: Using all Identity Operators:

Output:

```
# Initializing two variables a and b
a = ["Rose", "Lotus"]
b = ["Rose", "Lotus"]

# Initializing a variable c and storing the value of a in c
c = a

# Printing the different results
print("a is c =>", a is c)      # True (c is the same object as a)
print("a is not c =>", a is not c) # False (c is the same object as a)
print("a is b =>", a is b)      # False (a and b are different objects)
print("a is not b =>", a is not b) # True (a and b are different objects)
print("a == b =>", a == b)      # True (values inside are equal)
print("a != b =>", a != b)      # False (values inside are the same)
```

The output shows the results of each print statement:

```
a is c => True
a is not c => False
a is b => False
a is not b => True
a == b => True
a != b => False
```

How to read CSV file in Python

The CSV file stands for a comma-separated values file. It is a type of plain text file where the information is organized in the tabular form. It can contain only the actual text data. The textual data don't need to be separated by the commas (,). There are also many separator characters such as tab (\t), colon(:), and semi-colon(;), which can be used as a separator. Let's understand the following example.

```
# Importing the csv module
import csv

# Open file by passing the file path
with open(r'C:\Users\DEVANSH SHARMA\Desktop\example.csv', encoding='utf-8') as csv_file:
    csv_read = csv.reader(csv_file, delimiter=',') # Delimiter is a comma
    count_line = 0

    # Iterate over each row in the file
    for row in csv_read:
        if count_line == 0:
            print(f'Column names are: {", ".join(row)}')
        else:
            print(f'\t{row[0]} roll number is: {row[1]} and
department is: {row[2]}.')
        count_line += 1 # Increment line count

# Print the number of lines processed
print(f'Processed {count_line} lines.')
```

The output shows the printed rows from the CSV file:

```
a is c => True
a is not c => False
a is b => False
a is not b => True
a == b => True
a != b => False
```

How to reverse a string in Python

The collection of Unicode characters is Python String. Python has various capabilities for string control, yet Python string library doesn't uphold the in-constructed "switch()" capability. However, there are numerous methods for reversing the string. The following reverse Python String method is being defined.

```
1 # Function to reverse a string
2 def reverse_string(s):
3     str1 = "" # Declaring an empty string to store the reversed
4         string
5     for i in s:
6         str1 = i + str1 # Adding characters in reverse order
7     return str1 # Returning the reversed string
8
9 # Given String
10 input_str = "JavaTpoint"
11
12 # Printing results
13 print("The original string is:", input_str)
14 print("The reverse string is:", reverse_string(input_str)) # Function call
```

Using while loop

We can also reverse a string using a while loop. Let's understand the following example.

```
1 # Reverse string using a while loop
2
3 input_str = "JavaTpoint" # String variable
4 print("The original string is:", input_str)
5
6 reverse_string = "" # Empty String to store the reversed string
7 count = len(input_str) # Find the length of the string and save in
8 the count variable
9
10 # While loop to reverse the string
11 while count > 0:
12     reverse_string += input_str[count - 1] # Append character at
13     index (count-1) to reverse_string
14     count -= 1 # Decrement count
15
16 # Print the reversed string
17 print("The reversed string using a while loop is:", reverse_string)
```

Using the slice ([]) operator

We can also reverse the given string using the extended slice operator. Let's understand the following example.

The screenshot shows a Python code editor interface. On the left, the code file 'main.py' contains the following code:

```
1 # Reverse a string using slice syntax
2
3 # reverse() Function to reverse a string
4 def reverse(input_str): # Renamed parameter from 'str' to 'input_str'
5     input_str = input_str[::-1] # Reverse using extended slice
6     return input_str
7
8 # Given string
9 s = "JavaPoint"
10
11 # Print original and reversed strings
12 print("The original string is:", s)
13 print("The reversed string using extended slice operator is:",
reverse(s))
14
```

The right side of the interface shows the 'Console' tab with the output of the run command:

```
The original string is: JavaPoint
The reversed string using extended slice operator is: tniopTavaJ
```

The status bar at the bottom indicates the date and time as 05-02-2025 17:10.

Using reverse function with join

Python provides the `reversed()` function to reverse the string. Let's understand the following example.

The screenshot shows a Python code editor interface. On the left, the code file 'main.py' contains the following code:

```
1 # Reverse a string using reversed()
2
3 # Function to reverse a string
4 def reverse(input_str): # Renamed parameter from 'str' to 'input_str'
5     string = "".join(reversed(input_str)) # reversed() function
inside the join() function
6     return string
7
8 # Given string
9 s = "JavaPoint"
10
11 # Print original and reversed strings
12 print("The original string is:", s)
13 print("The reversed string using reversed() is:", reverse(s))
14
```

The right side of the interface shows the 'Console' tab with the output of the run command:

```
The original string is: JavaPoint
The reversed string using reversed() is: tniopTavaJ
```

The status bar at the bottom indicates the date and time as 05-02-2025 17:12.

Using recursion()

The recursion can also be used to turn the string around. Recursion is a cycle where capability calls itself. Look at the following example.

A screenshot of a Python code editor interface. The left pane shows the code file 'main.py' with the following content:

```
1 # Reverse a string using recursion
2
3 # Function to reverse a string
4 def reverse(input_str): # Renamed parameter from 'str' to 'input_str'
5     if len(input_str) == 0: # Base case: when the string is empty
6         return input_str
7     else:
8         # Recursively reverse the string and append the first
9         # character
10        return reverse(input_str[1:]) + input_str[0]
11
12 # Given string
13 str_value = "Devansh Sharma"
14
15 # Print original and reversed strings
16 print("The original string is:", str_value)
17 print("The reversed string (using recursion) is:", reverse(str_value))
```

The right pane shows the 'Console' tab with the output of running the code:

```
The original string is: Devansh Sharma
The reversed string (using recursion) is: amrahs hsnaveD
```

Python If-else statements

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the conditions. Condition checking is the backbone of decision making.

A screenshot of a Python code editor interface. The left pane shows the code file 'main.py' with the following content:

```
1
2 # Simple Python program to understand the if statement
3 num = int(input("Enter the number:"))
4 # Here, we are taking an integer num and taking input dynamically
5
6 if num % 2 == 0:
7     # Here, we are checking the condition. If the condition is true, we will enter the block
8     print("The given number is an even number")
9
```

The right pane shows the 'Terminal' tab with the output of running the code:

```
Enter the number:
10
The given number is an even number
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Program to print the largest of the three numbers.

The screenshot shows the Online Python Beta interface. The code in main.py is:

```
1 # Simple Python Program to print the largest of the three numbers.
2 a = int(input("Enter a: "))
3 b = int(input("Enter b: "))
4 c = int(input("Enter c: "))
5
6 if a > b and a > c:
7     # Here, we are checking the condition. If the condition is true, we will enter the block
8     print("From the above three numbers, given 'a' is the largest")
9
10 if b > a and b > c:
11    # Here, we are checking the condition. If the condition is true, we will enter the block
12    print("From the above three numbers, given 'b' is the largest")
13
14 if c > a and c > b:
15    # Here, we are checking the condition. If the condition is true, we will enter the block
16    print("From the above three numbers, given 'c' is the largest")
```

The terminal output shows the program running with inputs 100, 120, and 30, and the output "From the above three numbers, given 'b' is the largest".

** Process exited - Return Code: 0 **

The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

Program to check whether a person is eligible to vote or not.

The screenshot shows the Online Python Beta interface. The code in main.py is:

```
1 # Simple Python Program to check whether a person is eligible to vote or not.
2 age = int(input("Enter your age: "))
3 # Here, we are taking an integer age and taking input dynamically
4
5 if age >= 18:
6     # Here, we are checking the condition. If the condition is true, we will enter the block
7     print("You are eligible to vote!!")
8 else:
9     # This block will execute if the condition is false
10    print("Sorry! You have to wait!!")
```

A tooltip "Help Us Keep the Code Flowing!!" is visible.

The terminal output shows the program running with input 90, and the output "You are eligible to vote!!".

** Process exited - Return Code: 0 **
Press Enter to exit terminal

Program to check whether a number is even or not.

The screenshot shows the Online Python Beta interface. The code in the editor is:

```
1 # Simple Python Program to check whether a number is even or not.
2 num = int(input("Enter the number: "))
3 # Here, we are taking an integer num and taking input dynamically
4
5 if num % 2 == 0:
6     # Here, we are checking the condition. If the condition is true, we will enter the block
7     print("The given number is an even number")
8 else:
9     # This block will execute if the condition is false
10    print("The given number is an odd number")
```

The terminal output shows the program running and printing "The given number is an even number" for the input 10.

The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The screenshot shows the Online Python Beta interface. The code in the editor is:

```
1 # A simple python program to understand elif statement
2 number = int(input("Enter the number: "))
3 # Here, we are taking an integer number and taking input dynamically
4
5 if number == 10:
6     # Here, we are checking the condition. If the condition is true, we will enter the block
7     print("The given number is equal to 10")
8 elif number == 50:
9     # Here, we are checking the condition. If the condition is true, we will enter the block
10    print("The given number is equal to 50")
11 elif number == 100:
12     # Here, we are checking the condition. If the condition is true, we will enter the block
13     print("The given number is equal to 100")
14 else:
15     # This block will execute if none of the above conditions are true
16     print("The given number is not equal to 10, 50, or 100")
```

The terminal output shows the program running and printing "The given number is not equal to 10, 50, or 100" for the input 15.

```

1 # Simple Python program to understand elif statement
2 marks = int(input("Enter the marks: "))
3 # Here, we are taking an integer marks and taking input dynamically
4
5 if marks > 85 and marks <= 100:
6     # Here, we are checking the condition. If the condition is true, we will enter the block
7     print("Congrats! You scored grade A.")
8 elif marks > 60 and marks <= 85:
9     # Here, we are checking the condition. If the condition is true, we will enter the block
10    print("You scored grade B+.")
11 elif marks > 40 and marks <= 60:
12     # Here, we are checking the condition. If the condition is true, we will enter the block
13     print("You scored grade B.")
14 elif marks > 30 and marks <= 40:
15     # Here, we are checking the condition. If the condition is true, we will enter the block
16     print("You scored grade C.")
17 else:
18     # This block will execute if none of the above conditions are true
19     print("Sorry, you failed.")

Ln: 20, Col: 1
Run Share $ Command Line Arguments

```

Enter the marks:
89
Congrats! You scored grade A.

** Process exited - Return Code: 0 **
Press Enter to exit terminal

Python Loops

The following loops are available in Python to fulfil the looping needs. Python offers 3 choices for running the loops. The basic functionality of all the techniques is the same, although the syntax and the amount of time required for checking the condition differ.

For Loop

Python's for loop is designed to repeatedly execute a code block while iterating through a list, tuple, dictionary, or other iterable objects of Python. The process of traversing a sequence is known as iteration.

```

1 # Python program to show how the for loop works
2
3 # Creating a sequence which is a list of numbers
4 numbers = [4, 2, 6, 7, 3, 5, 8, 10, 6, 1, 9, 2]
5
6 # Creating an empty list to store squares
7 squares = []
8
9 # Creating a for loop
10 for value in numbers:
11     square = value ** 2
12     squares.append(square)
13
14 # Printing the list of squares after the loop
15 print("The list of squares is", squares)
16

Ln: 16, Col: 1
Run Share $ Command Line Arguments

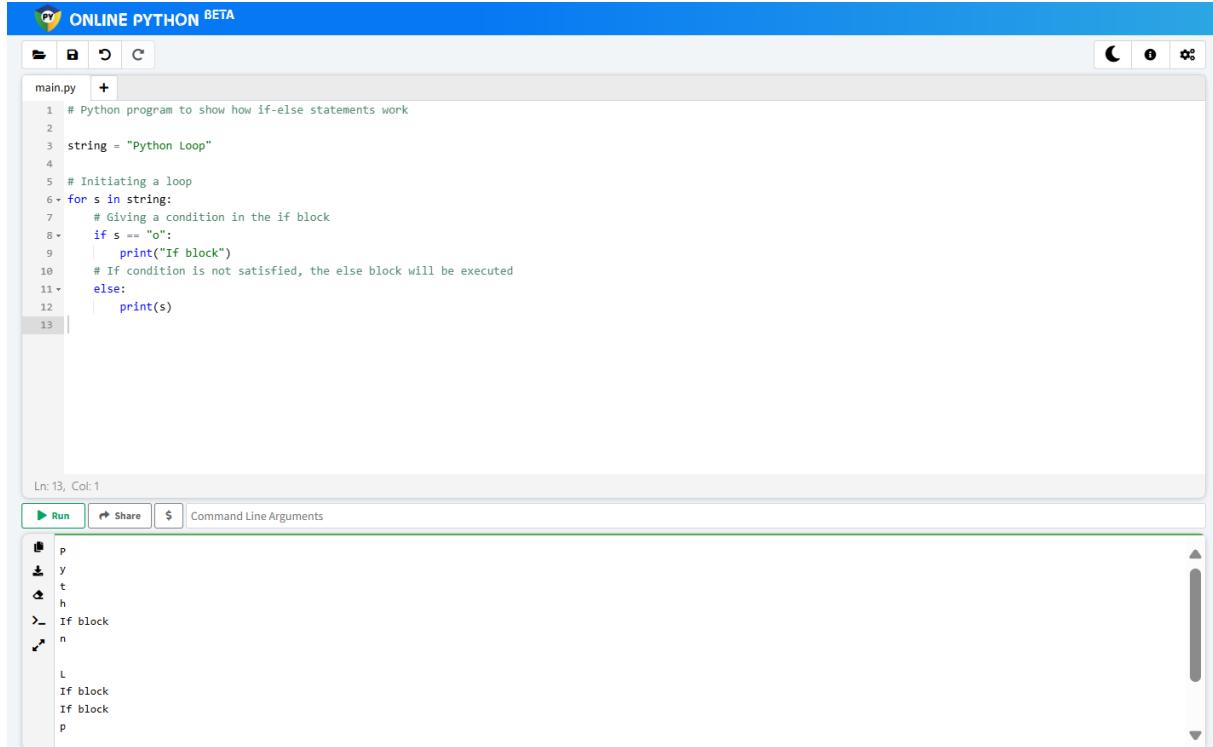
```

The list of squares is [16, 4, 36, 49, 9, 25, 64, 100, 36, 1, 81, 4]

** Process exited - Return Code: 0 **
Press Enter to exit terminal

Using else Statement with for Loop

As already said, a for loop executes the code block until the sequence element is reached. The statement is written right after the for loop is executed after the execution of the for loop is complete.

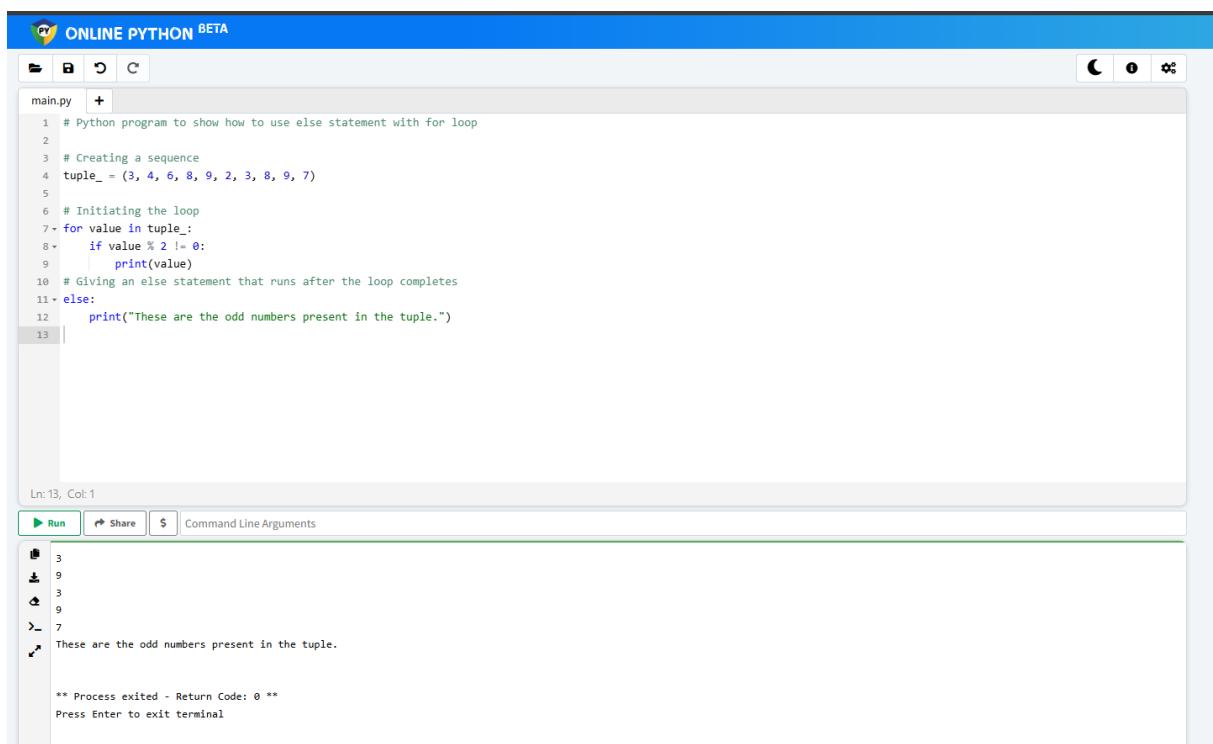


The screenshot shows the Online Python Beta interface. The code in the editor is:

```
1 # Python program to show how if-else statements work
2
3 string = "Python Loop"
4
5 # Initiating a loop
6 for s in string:
7     # Giving a condition in the if block
8     if s == "o":
9         print("If block")
10    # If condition is not satisfied, the else block will be executed
11 else:
12     print(s)
```

The terminal output shows the characters of the string followed by the character 'p':

```
p
y
t
h
I
f
b
o
l
o
p
```



The screenshot shows the Online Python Beta interface. The code in the editor is:

```
1 # Python program to show how to use else statement with for loop
2
3 # Creating a sequence
4 tuple_ = (3, 4, 6, 8, 9, 2, 3, 8, 9, 7)
5
6 # Initiating the loop
7 for value in tuple_:
8     if value % 2 != 0:
9         print(value)
10 # Giving an else statement that runs after the loop completes
11 else:
12     print("These are the odd numbers present in the tuple.)
```

The terminal output shows the odd numbers from the tuple followed by a message: "These are the odd numbers present in the tuple." and the exit message: "Process exited - Return Code: 0".

```
3
9
3
9
7
These are the odd numbers present in the tuple.

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

The range() Function

With the help of the range() function, we may produce a series of numbers. range(10) will produce values between 0 and 9. (10 numbers).

This screenshot shows the Online Python Beta interface. The code editor contains a file named main.py with the following content:

```
1 # Python program to show the working of range() function
2
3 # Printing a range object (won't show the actual numbers)
4 print(range(15))
5
6 # Printing the range as a list (shows numbers from 0 to 14)
7 print(list(range(15)))
8
9 # Printing the range from 4 to 8 (does not include 9)
10 print(list(range(4, 9)))
11
12 # Printing the range from 5 to 24, with a step of 4
13 print(list(range(5, 25, 4)))
14
```

The terminal window shows the output of running the code:

```
range(0, 15)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[4, 5, 6, 7, 8]
[5, 9, 13, 17, 21]

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

This screenshot shows the Online Python Beta interface. The code editor contains a file named main.py with the following content:

```
1 # Python program to iterate over a sequence with the help of indexing
2
3 tuple_ = ("Python", "Loops", "Sequence", "Condition", "Range")
4
5 # Iterating over tuple_ using range() function
6 for iterator in range(len(tuple_)):
7     print(tuple_[iterator].upper())
8
```

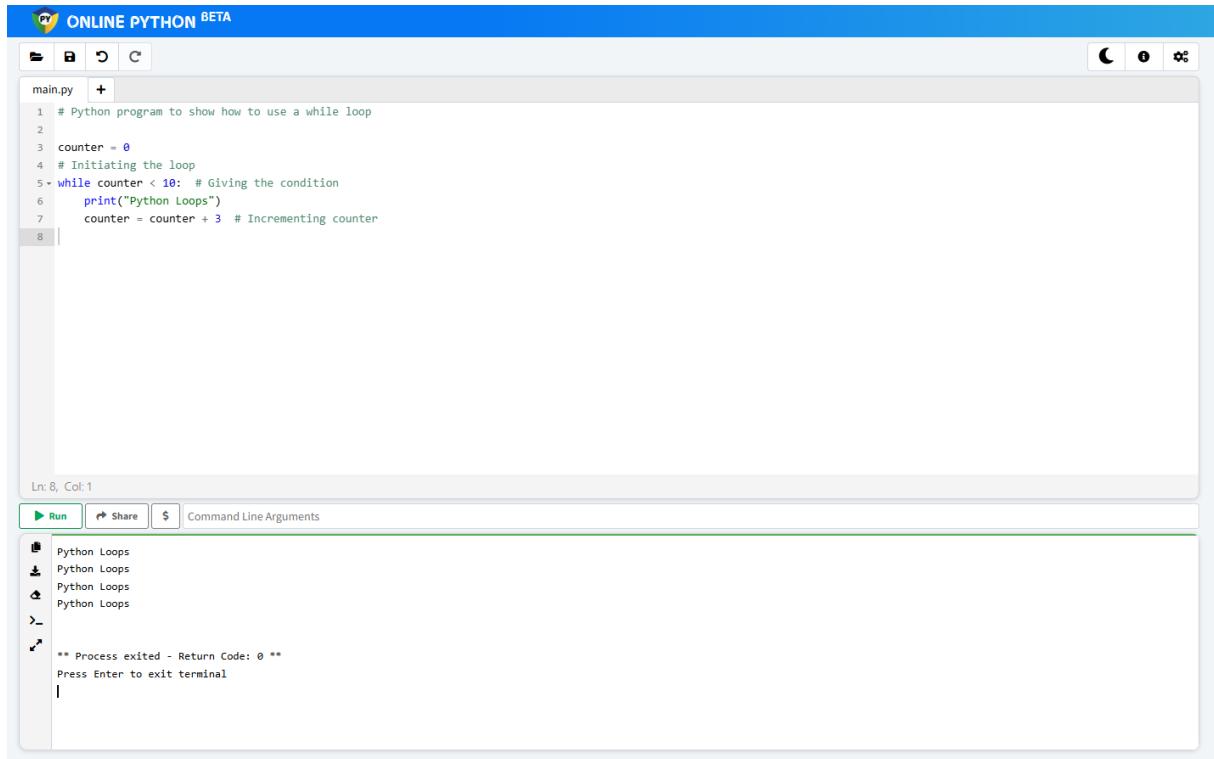
The terminal window shows the output of running the code:

```
PYTHON
LOOPS
SEQUENCE
CONDITION
RANGE

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

While Loop

While loops are used in Python to iterate until a specified condition is met. However, the statement in the program that follows the while loop is executed once the condition changes to false.



The screenshot shows a Python code editor interface with the title "ONLINE PYTHON BETA". The code in "main.py" is:

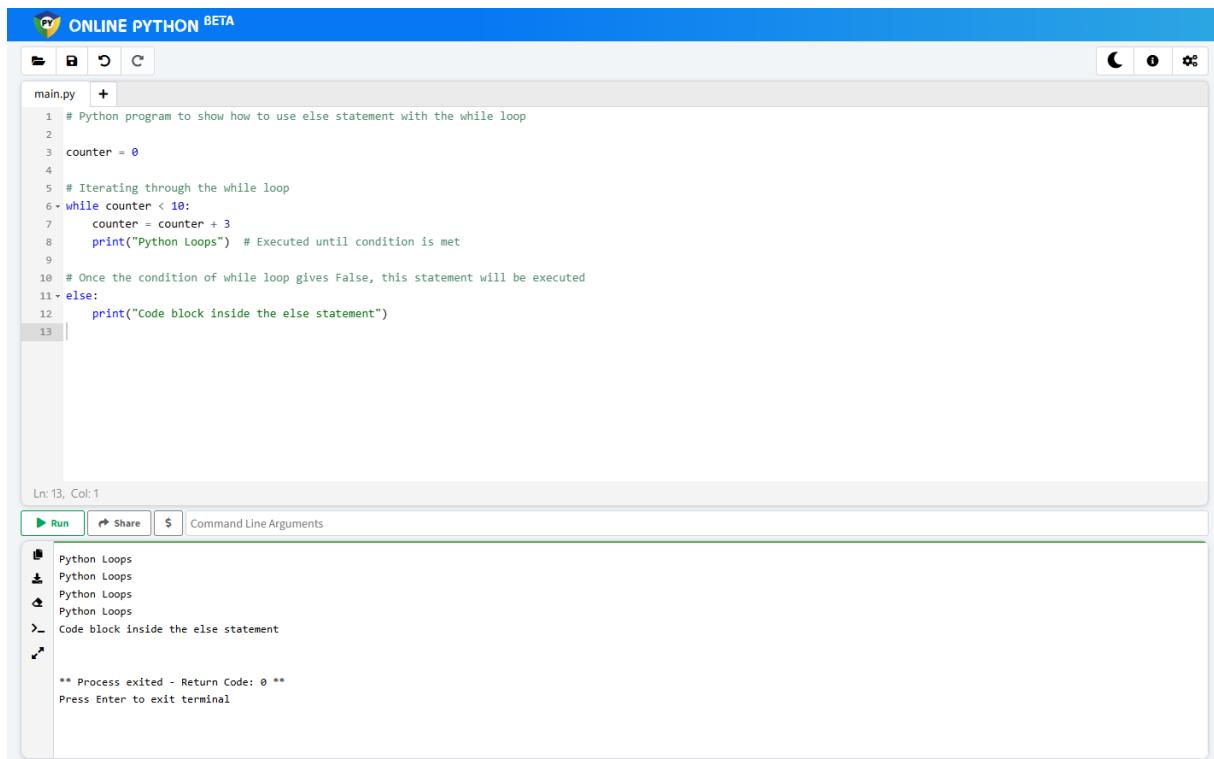
```
1 # Python program to show how to use a while loop
2
3 counter = 0
4 # Initiating the loop
5 while counter < 10: # Giving the condition
6     print("Python Loops")
7     counter = counter + 3 # Incrementing counter
```

The terminal window below shows the output of the program:

```
Python Loops
Python Loops
Python Loops
Python Loops
>_
<-- ** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Using else Statement with while Loops

As discussed earlier in the for loop section, we can use the else statement with the while loop also. It has the same syntax.



The screenshot shows a Python code editor interface with the title "ONLINE PYTHON BETA". The code in "main.py" is:

```
1 # Python program to show how to use else statement with the while loop
2
3 counter = 0
4
5 # Iterating through the while loop
6 while counter < 10:
7     counter = counter + 3
8     print("Python Loops") # Executed until condition is met
9
10 # Once the condition of while loop gives False, this statement will be executed
11 else:
12     print("Code block inside the else statement")
```

The terminal window below shows the output of the program:

```
Python Loops
Python Loops
Python Loops
Python Loops
>_
<-- Code block inside the else statement
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Single statement while Block

The loop can be declared in a single statement, as seen below. This is similar to the if-else block, where we can write the code block in a single line.

A screenshot of the Online Python Beta interface. The code editor window shows a file named 'main.py' with the following content:

```
1 # Python program to show how to write a single statement while loop
2
3 counter = 0
4 while counter < 3:
5     print("Python Loops")
6     counter += 1 # Make sure to increment counter to avoid an infinite loop
```

The terminal window below shows the output of running the script:

```
Python Loops
Python Loops
Python Loops
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Loop Control Statements

Now we will discuss the loop control statements in detail. We will see an example of each control statement.

Continue Statement

It returns the control to the beginning of the loop

A screenshot of the Online Python Beta interface. The code editor window shows a file named 'main.py' with the following content:

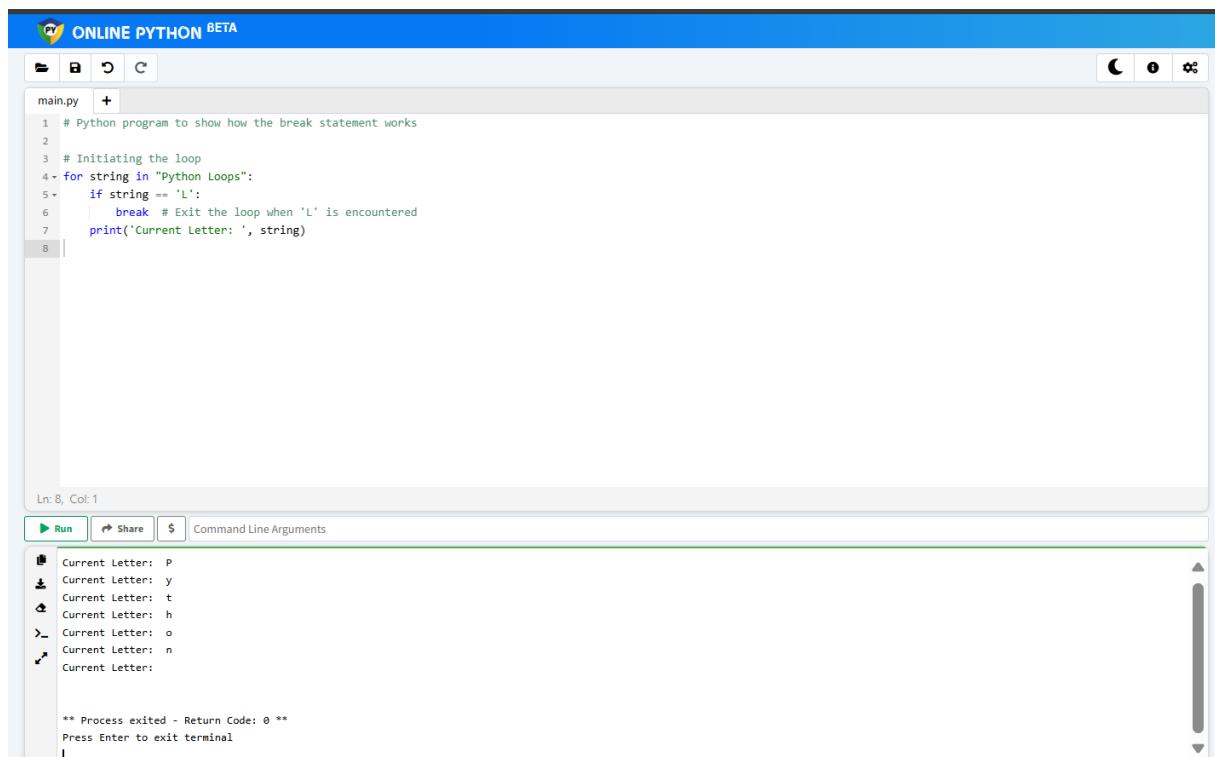
```
1 # Python program to show how the continue statement works
2
3 # Initiating the loop
4 for string in "Python Loops":
5     if string in ["o", "p", "t"]:  
6         continue  
7     print('Current Letter:', string)
```

The terminal window below shows the output of running the script:

```
Current Letter: P
Current Letter: y
Current Letter: h
Current Letter: n
Current Letter: n
Current Letter: l
Current Letter: s
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Break Statement

It stops the execution of the loop when the break statement is reached.



The screenshot shows the Online Python Beta interface. The code editor window contains a file named 'main.py' with the following content:

```
1 # Python program to show how the break statement works
2
3 # Initiating the loop
4 for string in "Python Loops":
5     if string == 'L':
6         break # Exit the loop when 'L' is encountered
7     print('Current Letter:', string)
8
```

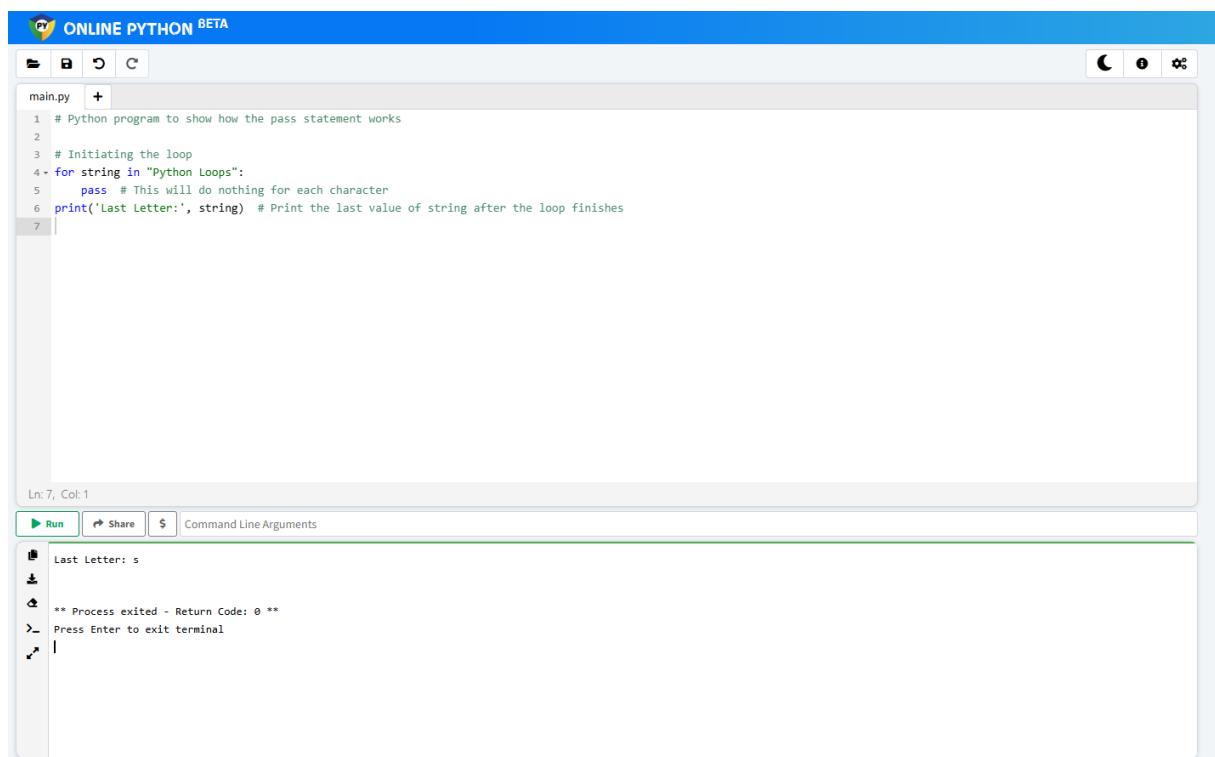
The terminal window below shows the output of running the script:

```
Ln: 8, Col: 1
▶ Run Share $ Command Line Arguments
Current Letter: P
Current Letter: y
Current Letter: t
Current Letter: h
Current Letter: o
Current Letter: n
Current Letter:

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Pass Statement

Pass statements are used to create empty loops. Pass statement is also employed for classes, functions, and empty control statements.



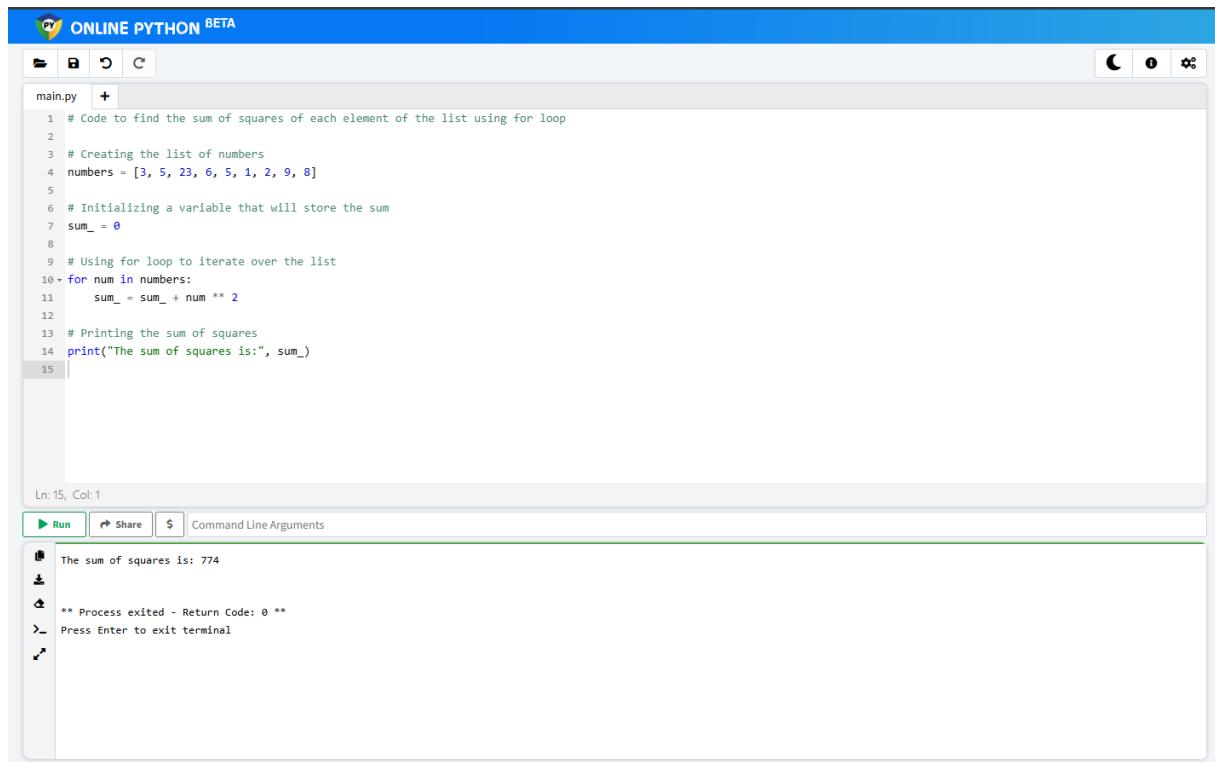
The screenshot shows the Online Python Beta interface. The code editor window contains a file named 'main.py' with the following content:

```
1 # Python program to show how the pass statement works
2
3 # Initiating the loop
4 for string in "Python Loops":
5     pass # This will do nothing for each character
6 print('Last Letter:', string) # Print the last value of string after the loop finishes
7
```

The terminal window below shows the output of running the script:

```
Ln: 7, Col: 1
▶ Run Share $ Command Line Arguments
Last Letter: s
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Example of Python for Loop



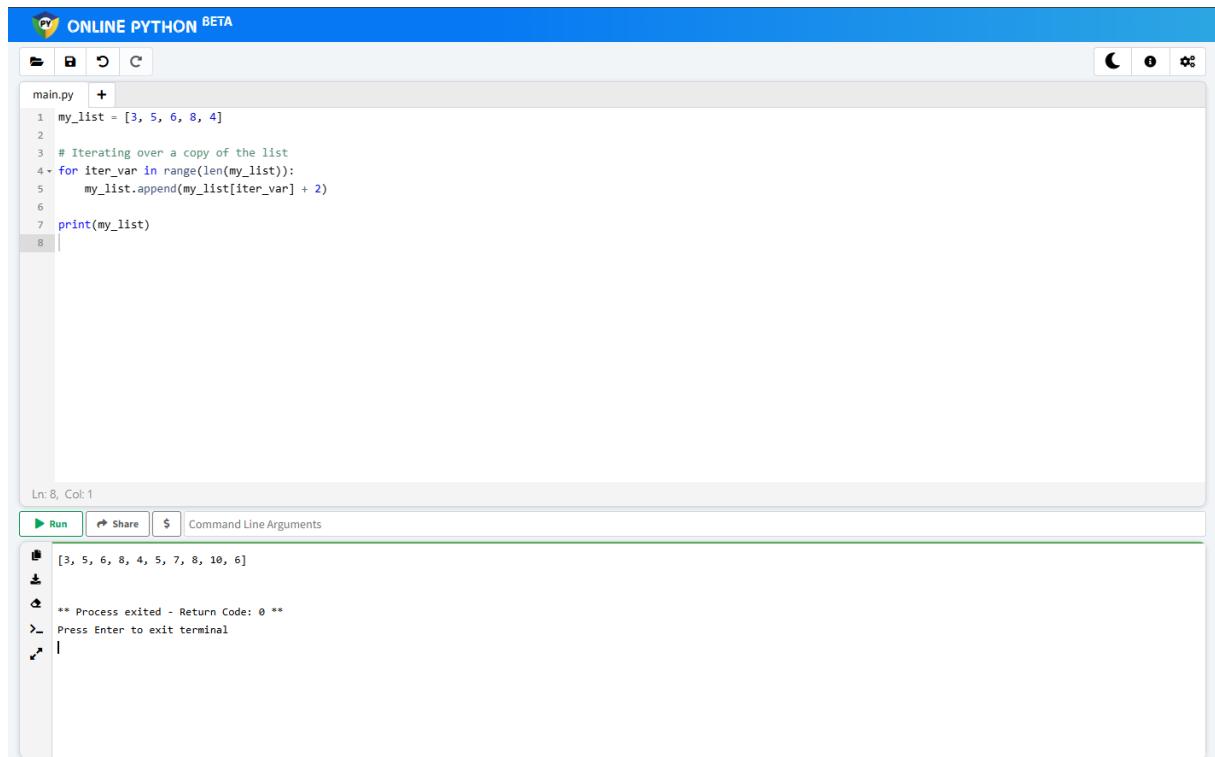
The screenshot shows a Python code editor interface. The code in 'main.py' is:

```
1 # Code to find the sum of squares of each element of the list using for loop
2
3 # Creating the list of numbers
4 numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
5
6 # Initializing a variable that will store the sum
7 sum_ = 0
8
9 # Using for loop to iterate over the list
10 for num in numbers:
11     sum_ = sum_ + num ** 2
12
13 # Printing the sum of squares
14 print("The sum of squares is:", sum_)
15
```

The output terminal shows:

```
The sum of squares is: 774
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

The range() Function



The screenshot shows a Python code editor interface. The code in 'main.py' is:

```
1 my_list = [3, 5, 6, 8, 4]
2
3 # Iterating over a copy of the list
4 for iter_var in range(len(my_list)):
5     my_list.append(my_list[iter_var] + 2)
6
7 print(my_list)
8
```

The output terminal shows:

```
[3, 5, 6, 8, 4, 5, 7, 8, 10, 6]
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Iterating by Using Index of Sequence

The screenshot shows a Python code editor window titled "ONLINE PYTHON BETA". The code in "main.py" is:

```
1 # Code to find the sum of squares of each element of the list using for loop
2
3 # Creating the list of numbers
4 numbers = [3, 5, 23, 6, 5, 1, 2, 9, 8]
5
6 # Initializing a variable that will store the sum
7 sum_ = 0
8
9 # Using for loop to iterate over the list
10 for num in numbers:
11     sum_ = sum_ + num ** 2
12
13 # Printing the sum of squares
14 print("The sum of squares is:", sum_)
15
```

The terminal output shows the result of running the code:

```
The sum of squares is: 774
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Using else Statement with for Loop

A loop expression and an else expression can be connected in Python.

The screenshot shows a Python code editor window titled "ONLINE PYTHON BETA". The code in "main.py" is:

```
1 # Code to print marks of a student from the record
2
3 student_name_1 = 'Itika'
4 student_name_2 = 'Parker'
5
6 # Creating a dictionary of records of the students
7 records = {'Itika': 90, 'Arshia': 92, 'Peter': 46}
8
9 def marks(student_name):
10     # Iterating over the keys of the dictionary
11     if student_name in records:
12         return records[student_name]
13     else:
14         return f'There is no student of name {student_name} in the records'
15
16 # Giving the function marks() the names of two students
17 print(f'Marks of {student_name_1} are: ', marks(student_name_1))
18 print(f'Marks of {student_name_2} are: ', marks(student_name_2))
19
```

The terminal output shows the result of running the code:

```
Marks of Itika are: 90
Marks of Parker are: There is no student of name Parker in the records
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Nested Loops

If we have a piece of content that we need to run various times and, afterward, one more piece of content inside that script that we need to run B several times, we utilize a "settled circle." While working with an iterable in the rundowns, Python broadly uses these.

The screenshot shows the Online Python Beta interface. The code in main.py is:

```
1 import random
2
3 # Creating an empty list
4 numbers = []
5
6 # Generating 11 random numbers between 0 and 10
7 for val in range(0, 11):
8     numbers.append(random.randint(0, 11))
9
10 # Set to track printed numbers
11 printed_numbers = set()
12
13 # Printing numbers from 0 to 10 that exist in the list
14 for num in range(0, 11):
15     if num in numbers and num not in printed_numbers:
16         print(num, end=" ")
17         printed_numbers.add(num) # Add the number to the set to avoid printing it again
18
```

The terminal output shows the numbers 0, 1, 2, 4, 6, 7, and 9, indicating that the inner loop only prints numbers that haven't been seen before.

Python While Loops

The screenshot shows the Online Python Beta interface. The code in main.py is:

```
1 # Python program example to show the use of while loop
2
3 num = 15
4
5 # initializing summation and a counter for iteration
6 summation = 0
7 c = 1
8
9 while c <= num: # specifying the condition of the loop
10     # beginning the code block
11     summation = c*c + summation
12     c = c + 1 # incrementing the counter
13
14 # print the final sum
15 print("The sum of squares is", summation)
16
```

The terminal output shows the result of the summation: 1240.

Prime Numbers and Python While Loop

The screenshot shows the Online Python BETA interface. The code in the editor is:

```
1 num = [34, 12, 54, 23, 75, 34, 11]
2
3 def prime_number(number):
4     condition = 0
5     iteration = 2
6     while iteration <= number / 2:
7         if number % iteration == 0:
8             condition = 1
9             break
10        iteration = iteration + 1
11
12 if condition == 0:
13     print(f"{number} is a PRIME number")
14 else:
15     print(f"{number} is not a PRIME number")
16
17 for i in num:
18     prime_number(i)
19
```

The terminal output shows the results for each number in the list:

```
34 is not a PRIME number
12 is not a PRIME number
54 is not a PRIME number
23 is a PRIME number
75 is not a PRIME number
34 is not a PRIME number
11 is a PRIME number

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Armstrong and Python While Loop

The screenshot shows the Online Python BETA interface. The code in the editor is:

```
1 n = int(input()) # Input the number
2 n1 = str(n) # Convert the number to a string to get the length
3 l = len(n1) # Get the number of digits
4 temp = n # Store the original number in temp
5 s = 0 # Variable to store the sum of the powers of the digits
6
7 while n != 0: # Loop through each digit
8     r = n % 10 # Get the last digit
9     s = s + (r ** l) # Add the digit raised to the power of the number of digits
10    n = n // 10 # Remove the last digit
11
12 # Check if the sum equals the original number
13 if s == temp:
14     print("It is an Armstrong number")
15 else:
16     print("It is not an Armstrong number")
17
```

The terminal output shows the result for the input 342:

```
342
It is not an Armstrong number

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Multiplication Table using While Loop

The screenshot shows the Online Python Beta interface. The code in the editor is:

```
main.py
1 num = 21 # The number for which we want the multiplication table
2 counter = 1 # Initialize the counter to 1
3
4 # Print the header for the multiplication table
5 print("The Multiplication Table of:", num)
6
7 # We will use a while loop for iterating 10 times for the multiplication table
8 while counter <= 10: # Loop condition to run the table from 1 to 10
9     ans = num * counter # Calculate the result
10    print(num, 'x', counter, '=', ans) # Print the multiplication result
11    counter += 1 # Increment the counter by 1
```

The terminal output shows the multiplication table for 21:

```
Ln: 12, Col: 1
Run Share $ Command Line Arguments
21 x 4 = 84
21 x 5 = 105
21 x 6 = 126
21 x 7 = 147
21 x 8 = 168
21 x 9 = 189
21 x 10 = 210

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Python While Loop with List

The screenshot shows the Online Python Beta interface. The code in the editor is:

```
main.py
1 # Python program to square every number of a list
2
3 # Initializing a list
4 list_ = [3, 5, 1, 4, 6]
5 squares = []
6
7 # Programming a while loop to square every number
8 while list_: # While list is not empty
9     squares.append((list_.pop()) ** 2) # Pop the last element and square it
10
11 # Print the squares of all numbers
12 print(squares)
13
```

The terminal output shows the squared values of the list elements:

```
Ln: 13, Col: 1
Run Share $ Command Line Arguments
[36, 16, 1, 25, 9]

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Now we give code examples of while loops in Python for determine odd and even number from every number of a list. The code is given below -

The screenshot shows a Python code editor interface. The code in 'main.py' is:

```
1 list_ = [3, 4, 8, 10, 34, 45, 67, 80] # Initialize the list
2 index = 0
3
4 # Loop through the list using a while loop
5 while index < len(list_):
6     element = list_[index]
7
8     if element % 2 == 0:
9         print(f'{element} is an even number') # Print if the number is even
10    else:
11        print(f'{element} is an odd number') # Print if the number is odd
12
13 index += 1 # Increment the index to check the next element
```

The terminal output shows the execution of the code, printing even numbers from 4 to 80.

```
4 is an even number
8 is an even number
10 is an even number
34 is an even number
45 is an odd number
67 is an odd number
80 is an even number

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Now we give code examples of while loops in Python for determine the number letters of every word from the given list. The code is given below -

The screenshot shows a Python code editor interface. The code in 'main.py' is:

```
1 List_ = ['Priya', 'Neha', 'Cow', 'To'] # Initialize the list
2 index = 0
3
4 # Loop through the list using a while loop
5 while index < len(List_):
6     element = List_[index]
7     print(len(element)) # Print the length of the current string
8     index += 1 # Increment the index to check the next element
```

The terminal output shows the execution of the code, printing the lengths of the words in the list.

```
5
4
3
2

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Python While Loop Multiple Conditions

The screenshot shows a Python code editor interface. The code in `main.py` is:

```
1 num1 = 17
2 num2 = -12
3
4 # While loop with multiple conditions
5 while num1 > 5 and num2 < -5:
6     num1 -= 2 # Decrease num1 by 2
7     num2 += 3 # Increase num2 by 3
8     print((num1, num2)) # Print the current values of num1 and num2
```

The terminal output shows the loop's iterations:

```
(15, -9)
(13, -6)
(11, -3)

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

The screenshot shows a Python code editor interface. The code in `main.py` is:

```
1 num1 = 9
2 num2 = 14
3 maximum_value = 4
4 counter = 0
5
6 # Loop with corrected conditions
7 while (counter < num1 and counter < num2) and not counter >= maximum_value:
8     print(f"Number of iterations: {counter}")
9     counter += 1
10
```

The terminal output shows the loop's iterations:

```
Number of iterations: 0
Number of iterations: 1
Number of iterations: 2
Number of iterations: 3

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Single Statement While Loop

The screenshot shows the Online Python Beta interface. The code editor window contains the following Python script:

```
main.py +  
1 counter = 1  
2 while counter:  
3     print('Python While Loops')  
4     counter = 0 # Exit the loop after the first iteration  
5
```

The terminal window below shows the output of running the script:

```
Python While Loops  
** Process exited - Return Code: 0 **  
Press Enter to exit terminal
```

Loop Control Statements

The screenshot shows the Online Python Beta interface. The code editor window contains the following Python script:

```
main.py +  
1 # Python program to show how to use continue loop control  
2  
3 # Initiating the loop  
4 for string in "While Loops":  
5     if string == "o" or string == "i" or string == "e":  
6         continue # Skip the rest of the loop iteration if the letter is 'o', 'i', or 'e'  
7     print('Current Letter:', string)  
8
```

The terminal window below shows the output of running the script:

```
Current Letter: W  
Current Letter: h  
Current Letter: l  
Current Letter:  
Current Letter: L  
Current Letter: p  
Current Letter: s  
  
** Process exited - Return Code: 0 **  
Press Enter to exit terminal
```

Break Statement

The screenshot shows the Online Python Beta interface. The code in main.py is:

```
1 # Python program to show how to use the break statement
2
3 # Initiating the loop
4 for string in "Python Loops":
5     if string == 'n': # Check if the current letter is 'n'
6         break # Stop the loop if 'n' is found
7     print('Current Letter: ', string)
```

The terminal output shows the letters from the string "Python Loops" being printed until it reaches the letter 'n', at which point the loop is terminated.

```
Ln: 8, Col: 1
Run Share $ Command Line Arguments
Current Letter: P
Current Letter: y
Current Letter: t
Current Letter: h
Current Letter: o
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Pass Statement

The screenshot shows the Online Python Beta interface. The code in main.py is:

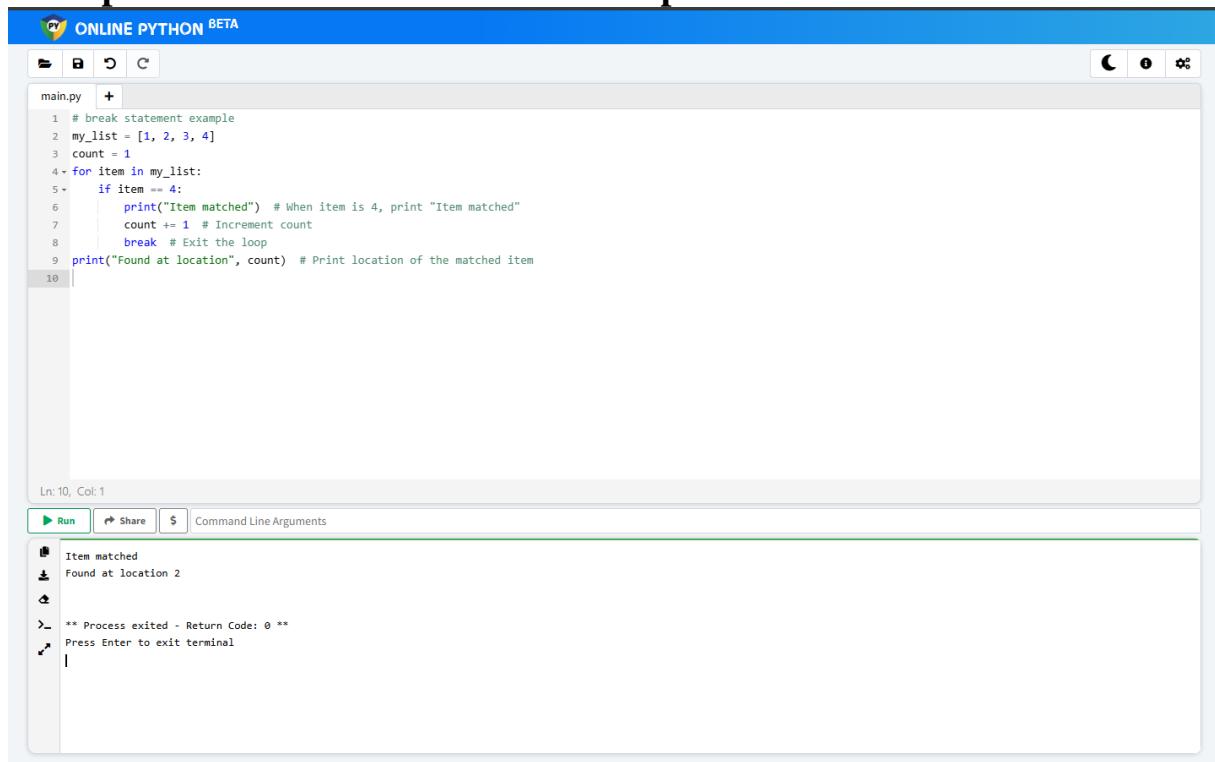
```
1 # Python program to show how to use the pass statement
2
3 for a_string in "Python Loops":
4     pass # The pass statement does nothing, it's just a placeholder
5
6 # Print the last letter of the given string
7 print('The Last Letter of given string is:', a_string)
```

The terminal output shows the last letter of the string "Python Loops" being printed.

```
Ln: 8, Col: 1
Run Share $ Command Line Arguments
The Last Letter of given string is: s
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Python break statement

Example 1 : break statement with for loop



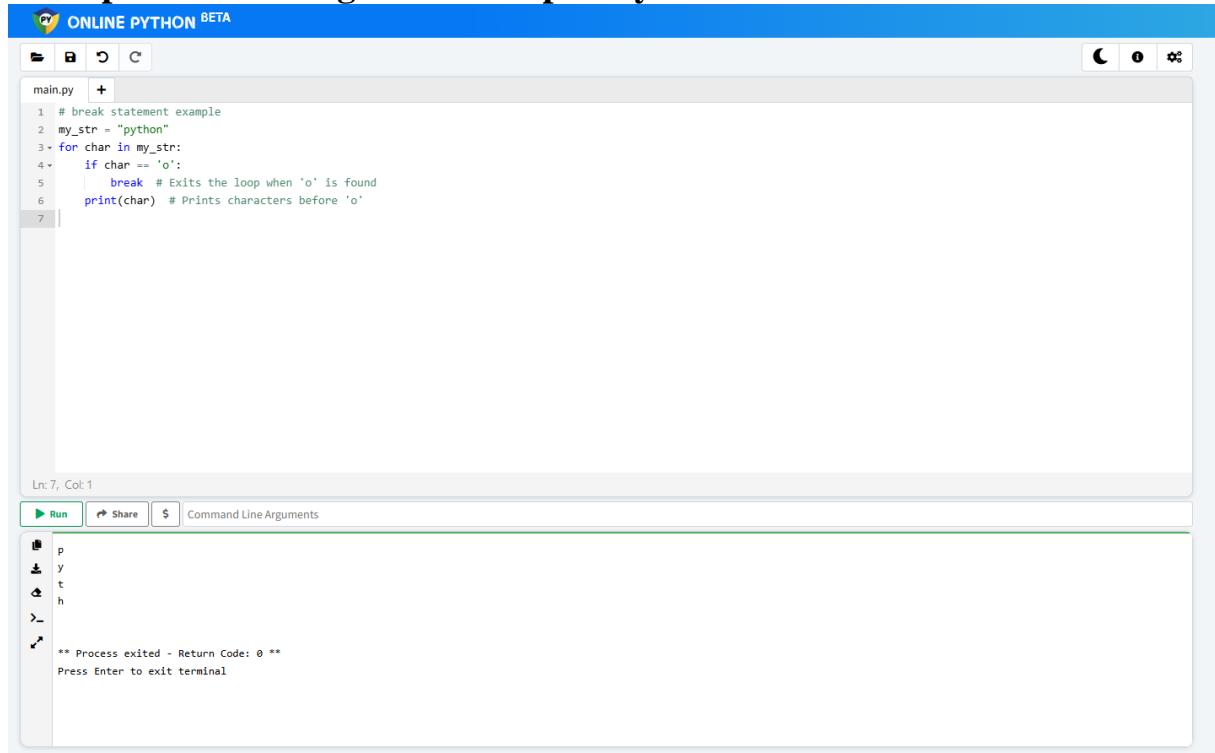
The screenshot shows the Online Python BETA interface. The code in main.py is:

```
1 # break statement example
2 my_list = [1, 2, 3, 4]
3 count = 1
4 for item in my_list:
5     if item == 4:
6         print("Item matched") # When item is 4, print "Item matched"
7         count += 1 # Increment count
8         break # Exit the loop
9 print("Found at location", count) # Print location of the matched item
10
```

The terminal output shows:

```
Item matched
Found at location 2
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Example 2 : Breaking out of a loop early



The screenshot shows the Online Python BETA interface. The code in main.py is:

```
1 # break statement example
2 my_str = "python"
3 for char in my_str:
4     if char == 'o':
5         break # Exits the loop when 'o' is found
6     print(char) # Prints characters before 'o'
7
```

The terminal output shows:

```
p
y
t
h
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Example 3: break statement with while loop

ONLINE PYTHON BETA

```
main.py +  
1 # break statement example  
2 i = 0  
3 while 1: # Infinite loop  
4     print(i, " ", end="") # Print current value of i  
5     i = i + 1 # Increment i  
6     if i == 10: # Check if i is 10  
7         break # Exit the loop  
8 print("came out of while loop") # Print after breaking the loop  
9
```

Ln: 9, Col: 1

Run Share \$ Command Line Arguments

```
0 1 2 3 4 5 6 7 8 9 came out of while loop  
  
** Process exited - Return Code: 0 **  
Press Enter to exit terminal
```

Example 4 : break statement with nested loops

ONLINE PYTHON BETA

```
main.py +  
1 # break statement example  
2 i = 0  
3 while 1: # Infinite loop  
4     print(i, " ", end="") # Print current value of i  
5     i = i + 1 # Increment i  
6     if i == 10: # Check if i is 10  
7         break # Exit the loop  
8 print("came out of while loop") # Print after breaking the loop  
9
```

Ln: 9, Col: 1

Run Share \$ Command Line Arguments

```
0 1 2 3 4 5 6 7 8 9 came out of while loop  
  
** Process exited - Return Code: 0 **  
Press Enter to exit terminal
```

Python continue Statement

The screenshot shows the Online Python Beta interface. The code editor contains a file named 'main.py' with the following content:

```
1 # Python code to show example of continue statement
2
3 # looping from 10 to 20
4 for iterator in range(10, 21):
5
6     # If iterator is equals to 15, loop will continue to the next iteration
7     if iterator == 15:
8         continue
9
10    # otherwise printing the value of iterator
11    print(iterator)
12
```

The terminal window below shows the output of running the code. Lines 13 through 19 are part of the code itself, and lines 20 and 21 are the output from the print statements.

```
Ln: 12, Col: 1
▶ Run Share $ Command Line Arguments
13
14
15
16
17
18
19
20

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Python Continue statement in list comprehension

The screenshot shows the Online Python Beta interface. The code editor contains a file named 'main.py' with the following content:

```
1 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3 # Using a list comprehension with continue
4 sq_num = [num ** 2 for num in numbers if num % 2 == 0]
5 # This will skip odd numbers and only square the even numbers
6 print(sq_num)
7
```

The terminal window below shows the output of running the code. Lines 1 through 6 are part of the code, and line 7 shows the resulting list [4, 16, 36, 64, 100].

```
Ln: 7, Col: 1
▶ Run Share $ Command Line Arguments
[4, 16, 36, 64, 100]

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Python String

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.

Creating String in Python

The screenshot shows the Online Python Beta interface. The code in `main.py` demonstrates three ways to create strings:

```
1 # Using single quotes
2 str1 = 'Hello Python'
3 print(str1)
4
5 # Using double quotes
6 str2 = "Hello Python"
7 print(str2)
8
9 # Using triple quotes
10 str3 = '''Triple quotes are generally used for
11 representing multiline strings or
12 docstrings'''
13 print(str3)
```

The terminal output shows the printed strings:

```
Hello Python
Hello Python
Triple quotes are generally used for
representing multiline strings or
docstrings
```

** Process exited - Return Code: 0 **

Strings indexing and splitting

The screenshot shows the Online Python Beta interface. The code in `main.py` prints each character of the string "HELLO" to demonstrate indexing:

```
1 str = "HELLO"
2 print(str[0]) # H
3 print(str[1]) # E
4 print(str[2]) # L
5 print(str[3]) # L
6 print(str[4]) # O
```

The terminal output shows the individual characters:

```
H
E
L
L
O
```

An `IndexError` is caught at index 6:

```
Traceback (most recent call last):
  File "<main.py>", line 9, in <module>
    print(str[6])
IndexError: string index out of range
```

** Process exited - Return Code: 1 **

ONLINE PYTHON BETA

```
main.py +  
1 # Given String  
2 str = "JAVATPOINT"  
3  
4 # Start from 0th index to the end  
5 print(str[0:])      # Output: JAVATPOINT  
6  
7 # Start from 1st index to 4th index (index 5 is excluded)  
8 print(str[1:5])     # Output: AVAT  
9  
10 # Start from 2nd index to 3rd index (index 4 is excluded)  
11 print(str[2:4])     # Output: VA  
12  
13 # Start from 0th index to 2nd index (index 3 is excluded)  
14 print(str[:3])      # Output: JAV  
15  
16 # Start from 4th index to 6th index (index 7 is excluded)  
17 print(str[4:7])     # Output: TPO  
18 |
```

Ln: 18, Col: 1

Run Share \$ Command Line Arguments

```
JAVATPOINT  
AVAT  
VA  
JAV  
TPO  
  
** Process exited - Return Code: 0 **  
Press Enter to exit terminal  
|
```

ONLINE PYTHON BETA

```
main.py +  
1 str = 'JAVATPOINT'  
2  
3 # Accessing elements using negative indexing  
4 print(str[-1])       # Output: T (Last character)  
5 print(str[-3])       # Output: I (Third from the end)  
6  
7 # Slicing using negative indices  
8 print(str[-2:])      # Output: NT (Last two characters)  
9 print(str[-4:-1])    # Output: OIN (From the 4th-last to the 2nd-last, excluding the last character)  
10 print(str[-7:-2])   # Output: VATPO (From the 7th-last to the 3rd-last, excluding the 2nd-last)  
11  
12 # Reversing the given string  
13 print(str[::-1])    # Output: TNIOPTAVAJ  
14  
15 # Attempt to access an index that is out of range (will cause an error)  
16 print(str[-12])     # This will raise an IndexError because the string length is only 10  
17 |
```

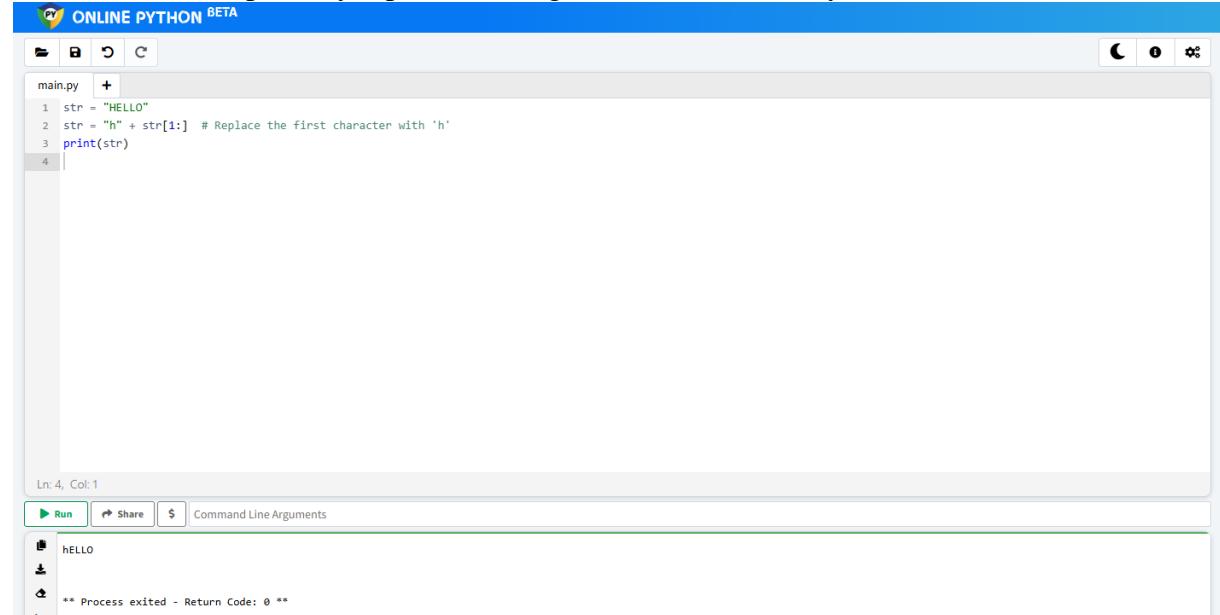
Ln: 17, Col: 1

Run Share \$ Command Line Arguments

```
T  
I  
NT  
OIN  
ATPOI  
TNIOPTAVAJ  
  
Traceback (most recent call last):  
  File "main.py", line 16, in <module>  
    print(str[-12])      # This will raise an IndexError because the string length is only 10  
IndexError: string index out of range
```

Reassigning Strings

Updating the content of the strings is as easy as assigning it to a new string. The string object doesn't support item assignment i.e., A string can only be replaced with new string since its content cannot be partially replaced. Strings are immutable in Python.

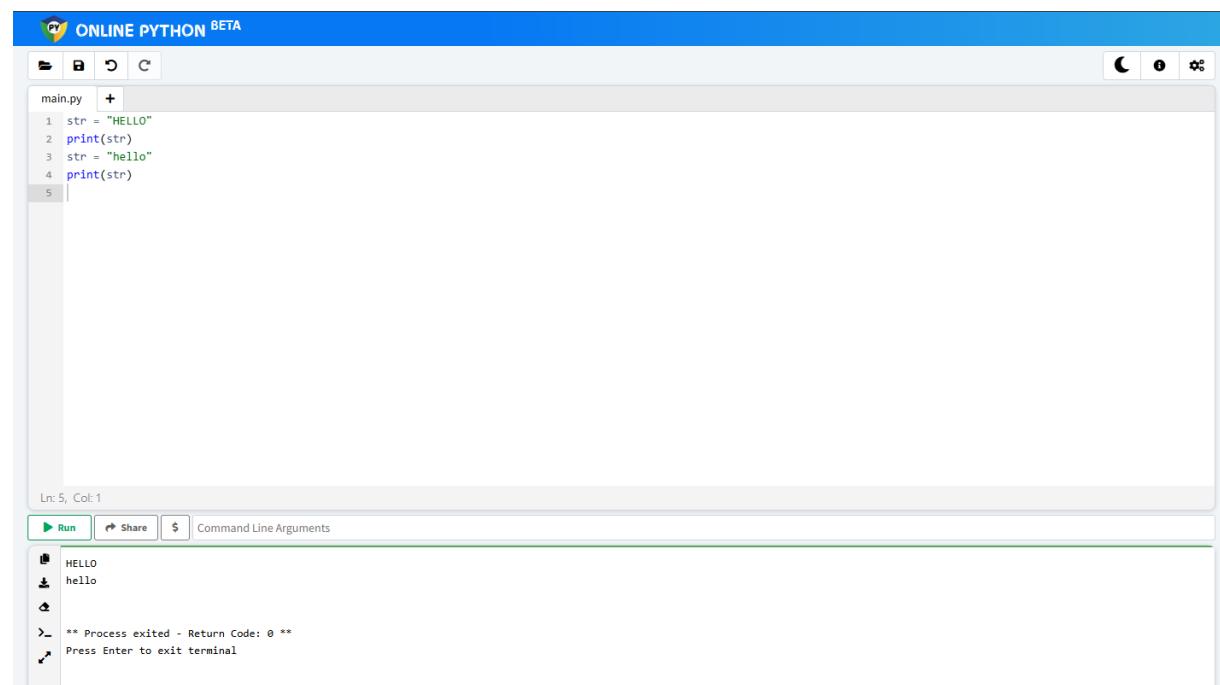


The screenshot shows the Online Python BETA interface. The code in the editor is:

```
main.py
1 str = "HELLO"
2 str = "h" + str[1:] # Replace the first character with 'h'
3 print(str)
4
```

The terminal output shows:

```
Ln: 4, Col: 1
▶ Run | Share | $ | Command Line Arguments
hELLO
** Process exited - Return Code: 0 **
>_
```



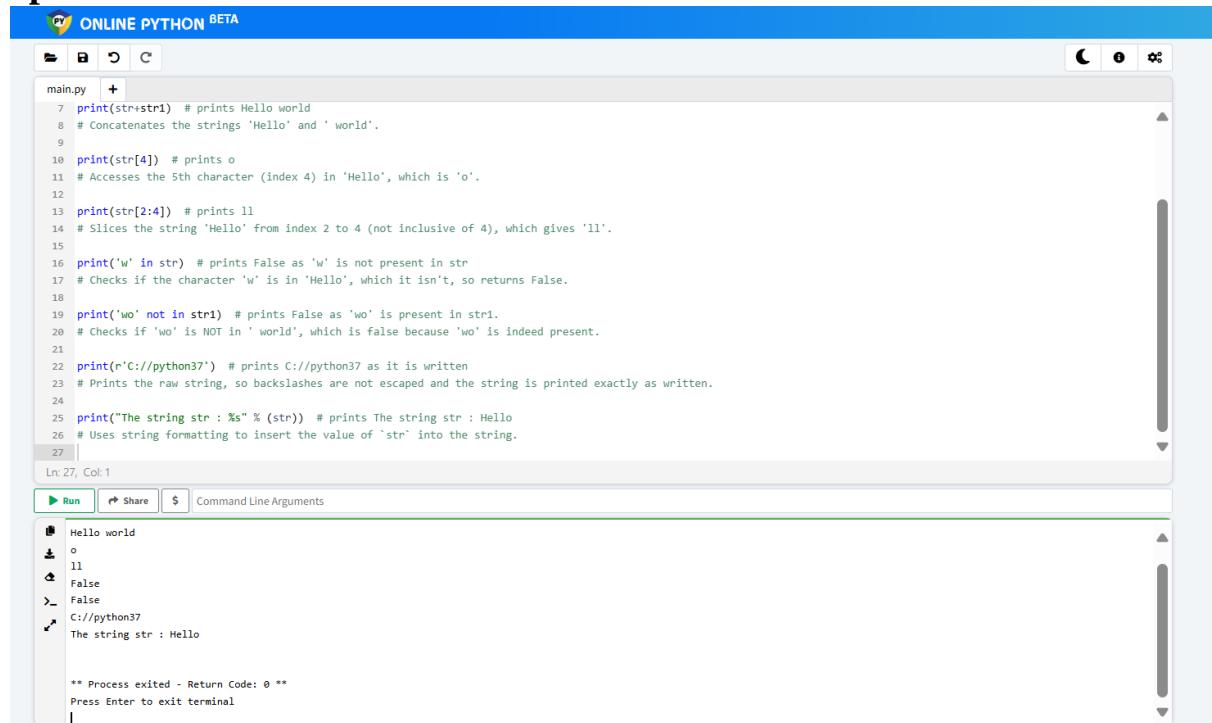
The screenshot shows the Online Python BETA interface. The code in the editor is:

```
main.py
1 str = "HELLO"
2 print(str)
3 str = "hello"
4 print(str)
5
```

The terminal output shows:

```
Ln: 5, Col: 1
▶ Run | Share | $ | Command Line Arguments
HELLO
hello
** Process exited - Return Code: 0 **
Press Enter to exit terminal
>_
```

Consider the following example to understand the real use of Python operators.



The screenshot shows the Online Python Beta interface. The code editor window contains a file named 'main.py' with the following content:

```
7 print(str+str1) # prints Hello world
8 # Concatenates the strings 'Hello' and ' world'.
9
10 print(str[4]) # prints o
11 # Accesses the 5th character (index 4) in 'Hello', which is 'o'.
12
13 print(str[2:4]) # prints ll
14 # Slices the string 'Hello' from index 2 to 4 (not inclusive of 4), which gives 'll'.
15
16 print('w' in str) # prints False as 'w' is not present in str
17 # Checks if the character 'w' is in 'Hello', which it isn't, so returns False.
18
19 print('wo' not in str1) # prints False as 'wo' is present in str1.
20 # Checks if 'wo' is NOT in ' world', which is false because 'wo' is indeed present.
21
22 print(r'C://python37') # prints C://python37 as it is written
23 # Prints the raw string, so backslashes are not escaped and the string is printed exactly as written.
24
25 print("The string str : %s" % (str)) # prints The string str : Hello
26 # Uses string formatting to insert the value of 'str' into the string.
27
```

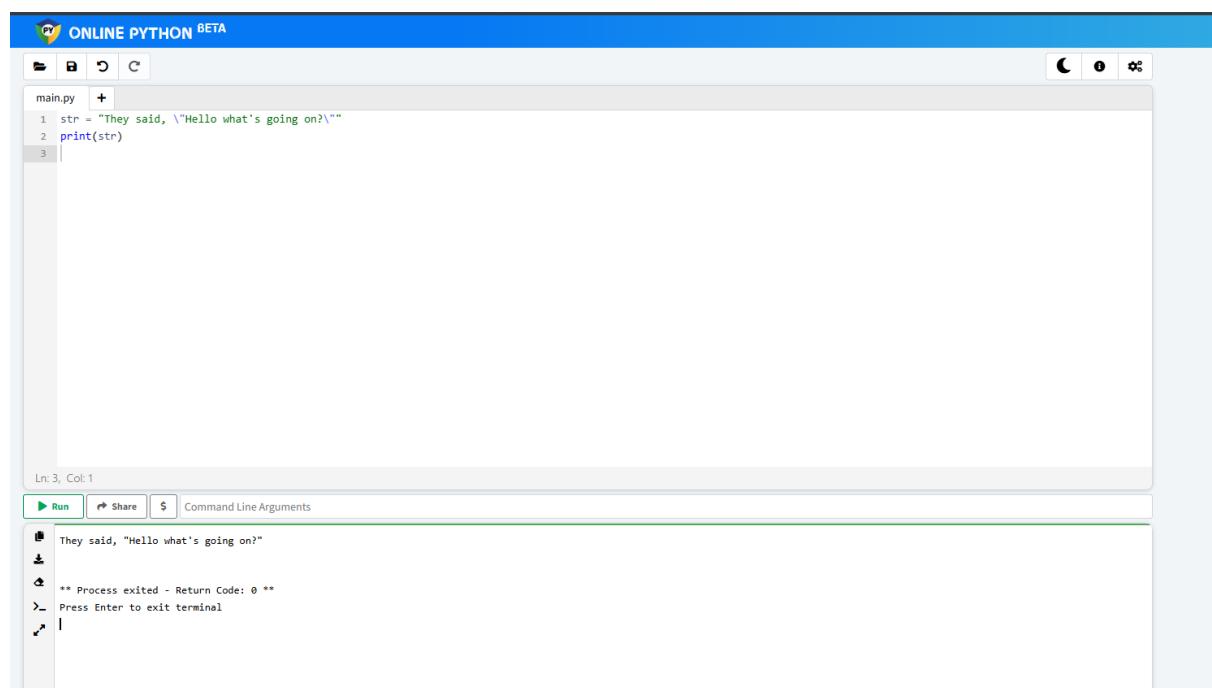
The terminal window below shows the output of running the script:

```
Hello world
o
ll
False
False
C://python37
The string str : Hello

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Python String Formatting Escape Sequence

Let's suppose we need to write the text as - They said, "Hello what's going on?" - the given statement can be written in single quotes or double quotes but it will raise the SyntaxError as it contains both single and double-quotes.



The screenshot shows the Online Python Beta interface. The code editor window contains a file named 'main.py' with the following content:

```
1 str = "They said, \"Hello what's going on?\""
2 print(str)
3
```

The terminal window below shows the output of running the script:

```
They said, "Hello what's going on?"

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

ONLINE PYTHON BETA

main.py +

```
1 # using triple quotes
2 print('''They said, "What's there?'''')
3
4 # escaping single quotes
5 print('They said, "What\'s going on?"')
6
7 # escaping double quotes
8 print("They said, \"What's going on?\"")
9 |
```

Ln: 9, Col: 1

Run Share \$ Command Line Arguments

```
'They said, "What's there?"'
They said, "What's going on?"
They said, "What's going on?"

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

ONLINE PYTHON BETA

main.py +

```
1 print("C:\\\\Users\\\\DEVANSH SHARMA\\\\Python32\\\\Lib")
2 print("This is the \\n multiline quotes")
3 print("This is \\x48\\x45\\x58 representation")
4
```

Ln: 4, Col: 1

Run Share \$ Command Line Arguments

```
C:\\Users\\DEVANSH SHARMA\\Python32\\Lib
This is the
multiline quotes
This is HEX representation

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

ONLINE PYTHON BETA

main.py +

```
1 print(r"C:\\\\Users\\\\DEVANSH SHARMA\\\\Python32")
2
```

Ln: 2, Col: 1

Run Share \$ Command Line Arguments

```
C:\\\\Users\\\\DEVANSH SHARMA\\\\Python32

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

The format() method

This screenshot shows the Online Python BETA interface. The code in main.py demonstrates various ways to use the `format()` method:

```
1 # Using Curly braces
2 print("{} and {} both are the best friend".format("Devansh", "Abhishek"))
3
4 # Positional Argument
5 print("{1} and {0} best players ".format("Virat", "Rohit"))
6
7 # Keyword Argument
8 print("{a},{b},{c}".format(a = "James", b = "Peter", c = "Ricky"))
```

The output terminal shows the results of running the code:

```
Devansh and Abhishek both are the best friend
Rohit and Virat best players
James,Peter,Ricky
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Python String Formatting Using % Operator

This screenshot shows the Online Python BETA interface. The code in main.py demonstrates string formatting using the `%` operator:

```
1 Integer = 10
2 Float = 1.290
3 String = "Devansh"
4 print("Hi I am Integer ... My value is %d\nHi I am float ... My value is %f\nHi I am string ... My value is %s" % (Integer, Float, String))
```

The output terminal shows the results of running the code:

```
Hi I am Integer ... My value is 10
Hi I am float ... My value is 1.290000
Hi I am string ... My value is Devansh
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

List and Tuple Syntax Differences

The screenshot shows a Python code editor interface with the title "ONLINE PYTHON BETA". The code in "main.py" is:

```
1 # Python code to show the difference between creating a list and a tuple
2
3 list_ = [4, 5, 7, 1, 7]
4 tuple_ = (4, 1, 8, 3, 9)
5
6 print("List is: ", list_)
7 print("Tuple is: ", tuple_)
```

The output window shows the results of running the code:

```
List is: [4, 5, 7, 1, 7]
Tuple is: (4, 1, 8, 3, 9)

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Mutable List vs. Immutable Tuple

The screenshot shows a Python code editor interface with the title "ONLINE PYTHON BETA". The code in "main.py" is:

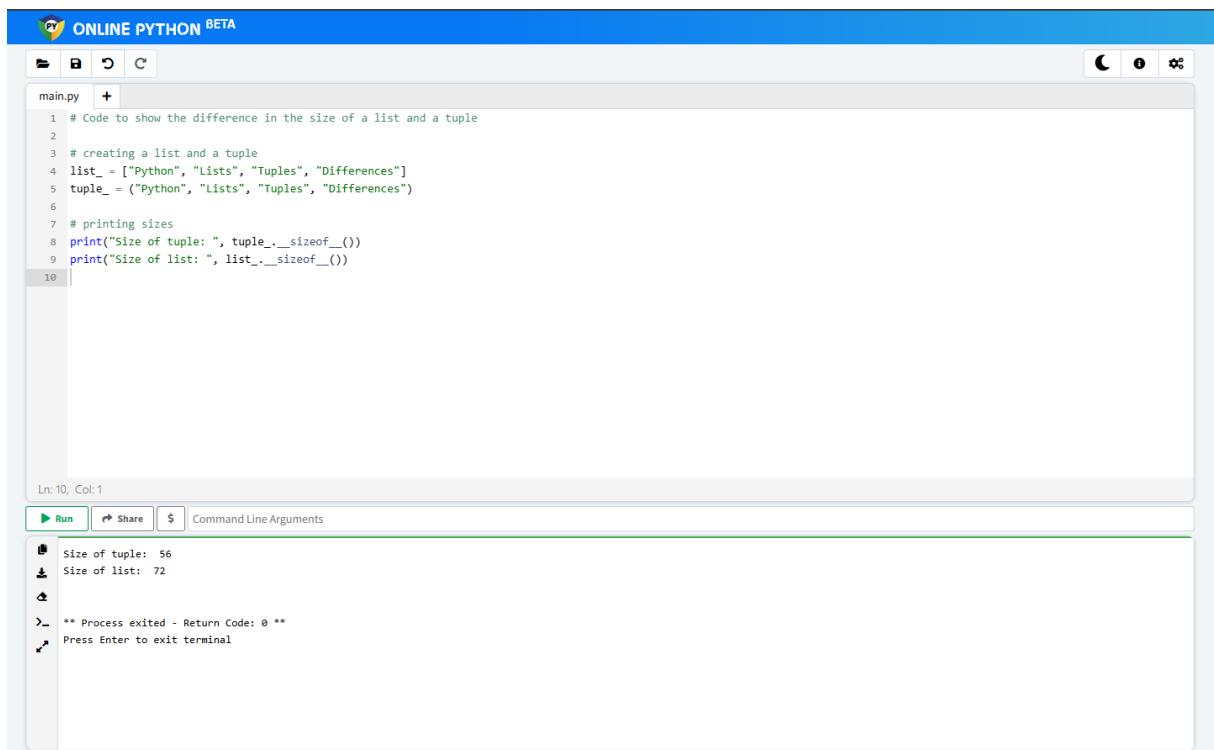
```
1 # Updating the element of list and tuple at a particular index
2
3 # creating a list and a tuple
4 list_ = ["Python", "Lists", "Tuples", "Differences"]
5 tuple_ = ("Python", "Lists", "Tuples", "Differences")
6
7 # modifying the last string in both data structures
8 list_[3] = "Mutable"
9 print(list_)
10
11 try:
12     tuple_[3] = "Immutable"
13     print(tuple_)
14 except TypeError:
15     print("Tuples cannot be modified because they are immutable")
16
```

The output window shows the results of running the code:

```
['Python', 'Lists', 'Tuples', 'Mutable']
Tuples cannot be modified because they are immutable

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Size Difference



The screenshot shows a Python code editor interface with the title "ONLINE PYTHON BETA". The code in "main.py" is:

```
1 # Code to show the difference in the size of a list and a tuple
2
3 # creating a list and a tuple
4 list_ = ["Python", "Lists", "Tuples", "Differences"]
5 tuple_ = ("Python", "Lists", "Tuples", "Differences")
6
7 # printing sizes
8 print("Size of tuple: ", tuple_.__sizeof__())
9 print("Size of list: ", list_.__sizeof__())
10
```

The output terminal shows:

```
Ln: 10, Col: 1
Run Share $ Command Line Arguments
█ Size of tuple: 56
█ Size of list: 72
█
> ** Process exited - Return Code: 0 **
Press Enter to exit terminal
```