

MANAV RACHNA UNIVERSITY
AIML SUPERVISED LEARNING 3rd SEM
ASSIGNMENT 1

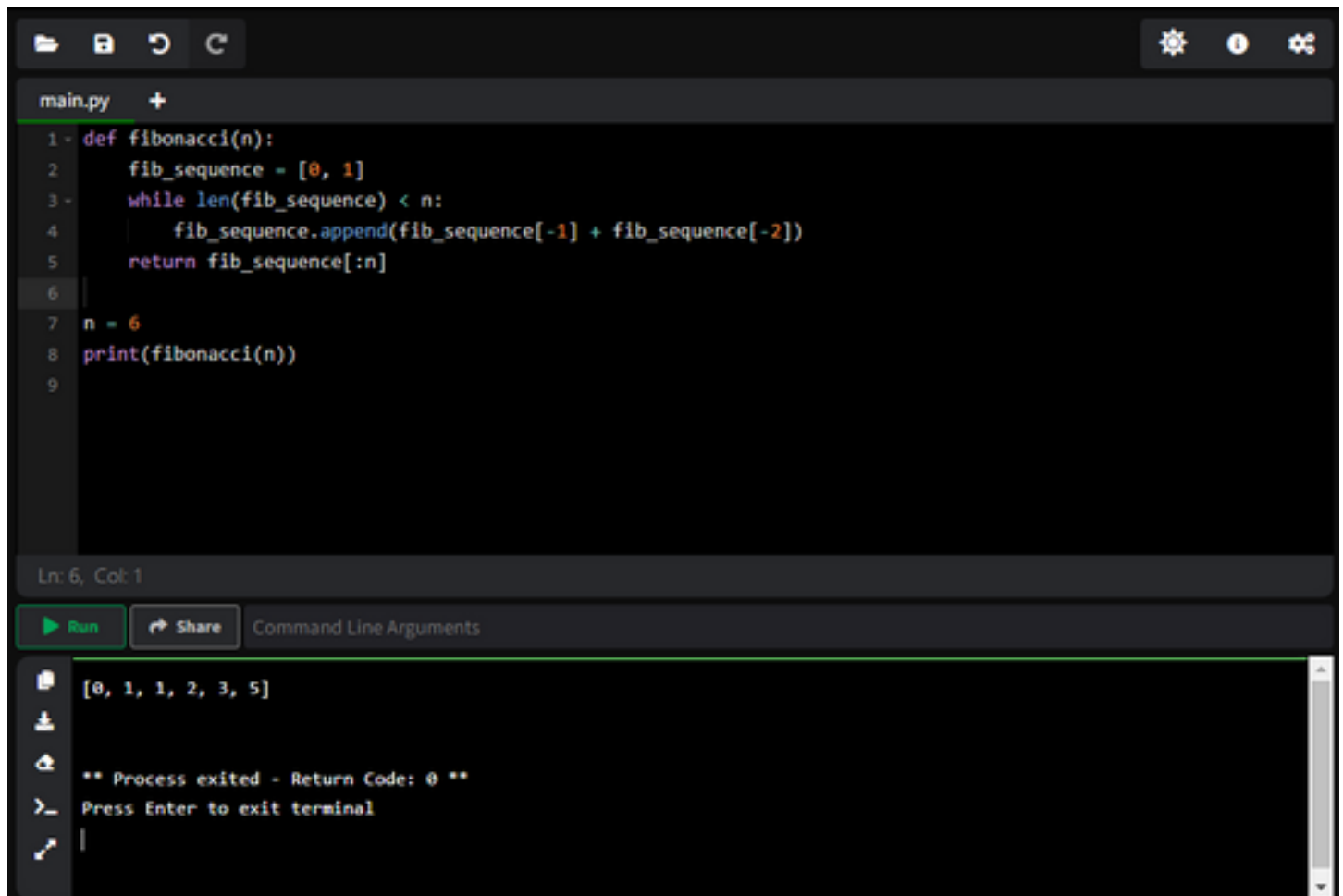
NAME: Abhijeet Kumar Soni

23K23CSUNO1240

AIML 3B

1. Fibonacci Sequence

Write a function to generate the first n Fibonacci numbers using a loop. For example, if n = 6, the output should be [0, 1, 1, 2, 3, 5].



```
main.py +
1 def fibonacci(n):
2     fib_sequence = [0, 1]
3     while len(fib_sequence) < n:
4         fib_sequence.append(fib_sequence[-1] + fib_sequence[-2])
5     return fib_sequence[:n]
6
7 n = 6
8 print(fibonacci(n))
9
```

Ln: 6, Col: 1

Run Share Command Line Arguments

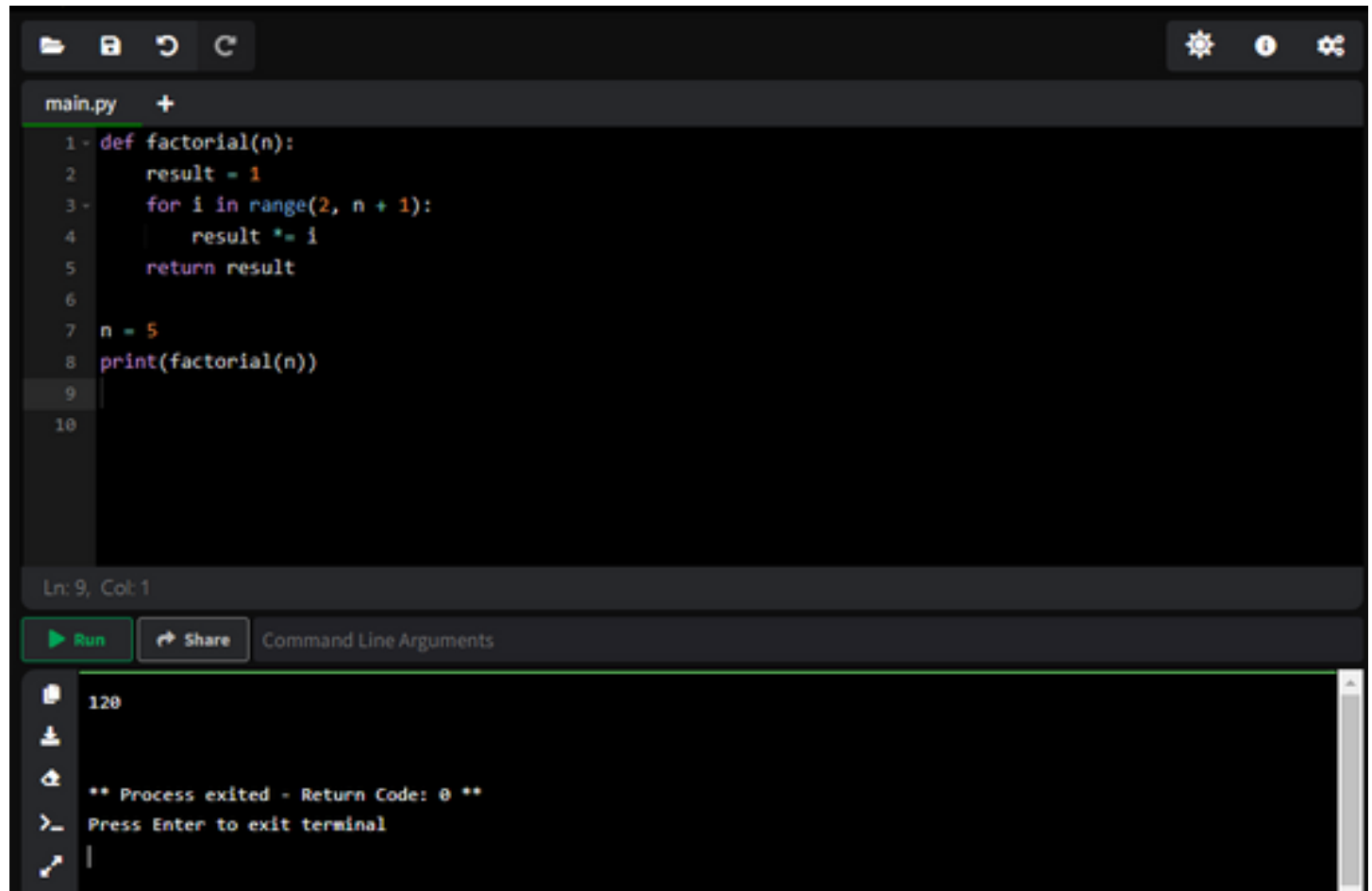
[0, 1, 1, 2, 3, 5]

** Process exited - Return Code: 0 **

>_ Press Enter to exit terminal

2. Factorial Calculation

Write a function to compute the factorial of a given number using a loop. For example, factorial(5) should return 120.



The image shows a code editor window with a dark theme. The editor displays a Python script named `main.py`. The script defines a `factorial` function that uses a `for` loop to calculate the factorial of a given number `n`. The function initializes `result` to 1 and iterates from 2 to `n + 1`, multiplying `result` by each value of `i`. After the function definition, the script sets `n = 5` and prints the result of `factorial(n)`. Below the code editor, there is a status bar showing the cursor position at line 9, column 1. A `Run` button is visible, and the output of the program is displayed in a terminal window at the bottom. The output shows the number 120, followed by a message indicating the process exited with a return code of 0, and a prompt to press Enter to exit the terminal.

```
main.py +
1 def factorial(n):
2     result = 1
3     for i in range(2, n + 1):
4         result *= i
5     return result
6
7 n = 5
8 print(factorial(n))
9
10
```

Ln: 9, Col: 1

[Run](#) [Share](#) Command Line Arguments

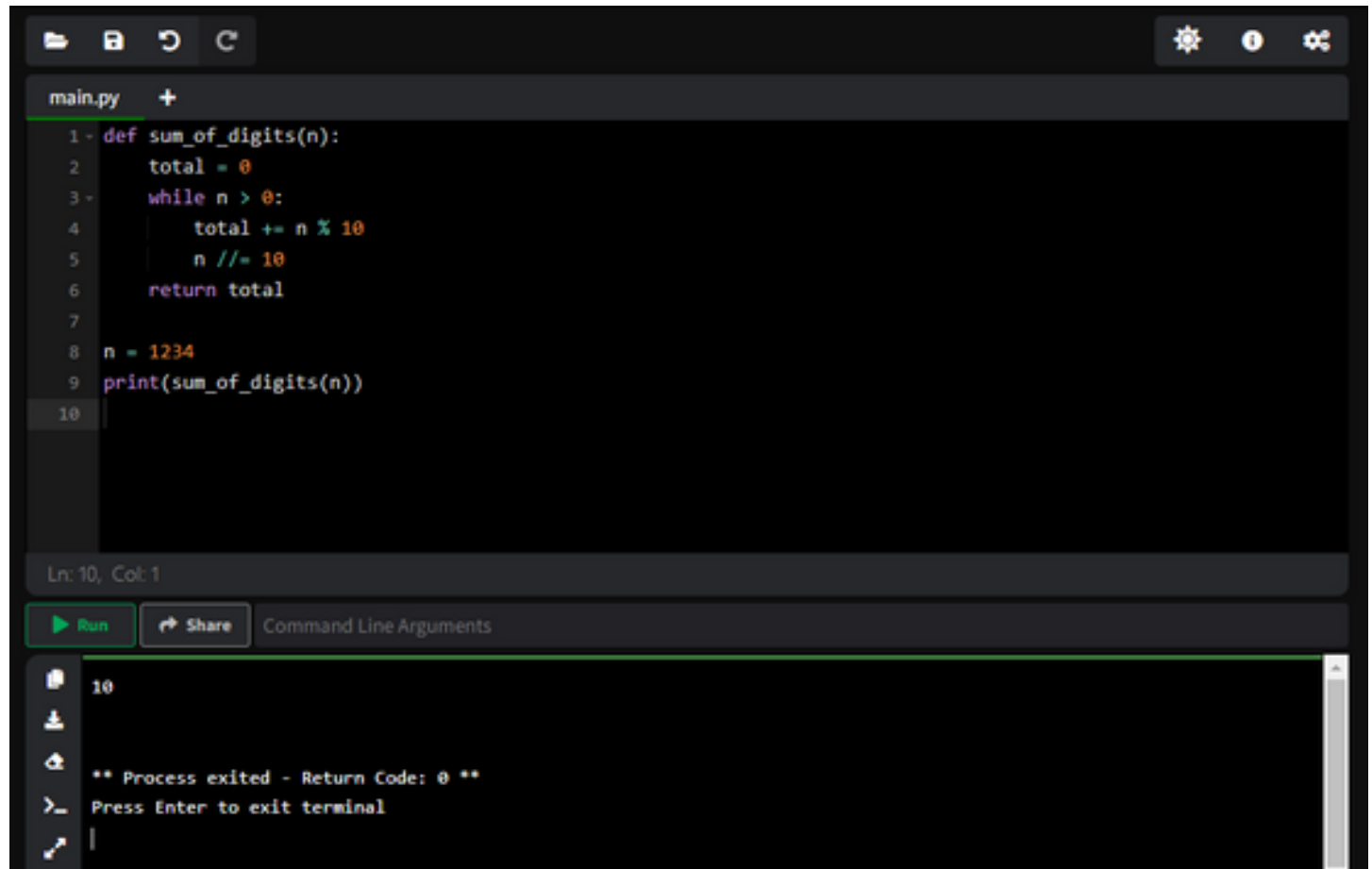
120

**** Process exited - Return Code: 0 ****

> Press Enter to exit terminal

3. Sum of Digits

Write a function that takes an integer and returns the sum of its digits. For example, for the number 1234, the output should be 10 (1 + 2 + 3 + 4).



The image shows a code editor window with a dark theme. The editor has a toolbar at the top with icons for file operations and settings. The main area displays a Python script in a file named `main.py`. The script defines a function `sum_of_digits(n)` that calculates the sum of the digits of a number `n` using a while loop. Below the function definition, the number `n` is set to 1234, and the function is called with `print(sum_of_digits(n))`. The line numbers 1 through 10 are visible on the left. At the bottom of the editor, there is a status bar showing 'Ln: 10, Col: 1' and buttons for 'Run' and 'Share'. Below the editor, a terminal window shows the output '10' and a message indicating the process exited successfully.

```
1 def sum_of_digits(n):
2     total = 0
3     while n > 0:
4         total += n % 10
5         n //= 10
6     return total
7
8 n = 1234
9 print(sum_of_digits(n))
10
```

Ln: 10, Col: 1

Run Share Command Line Arguments

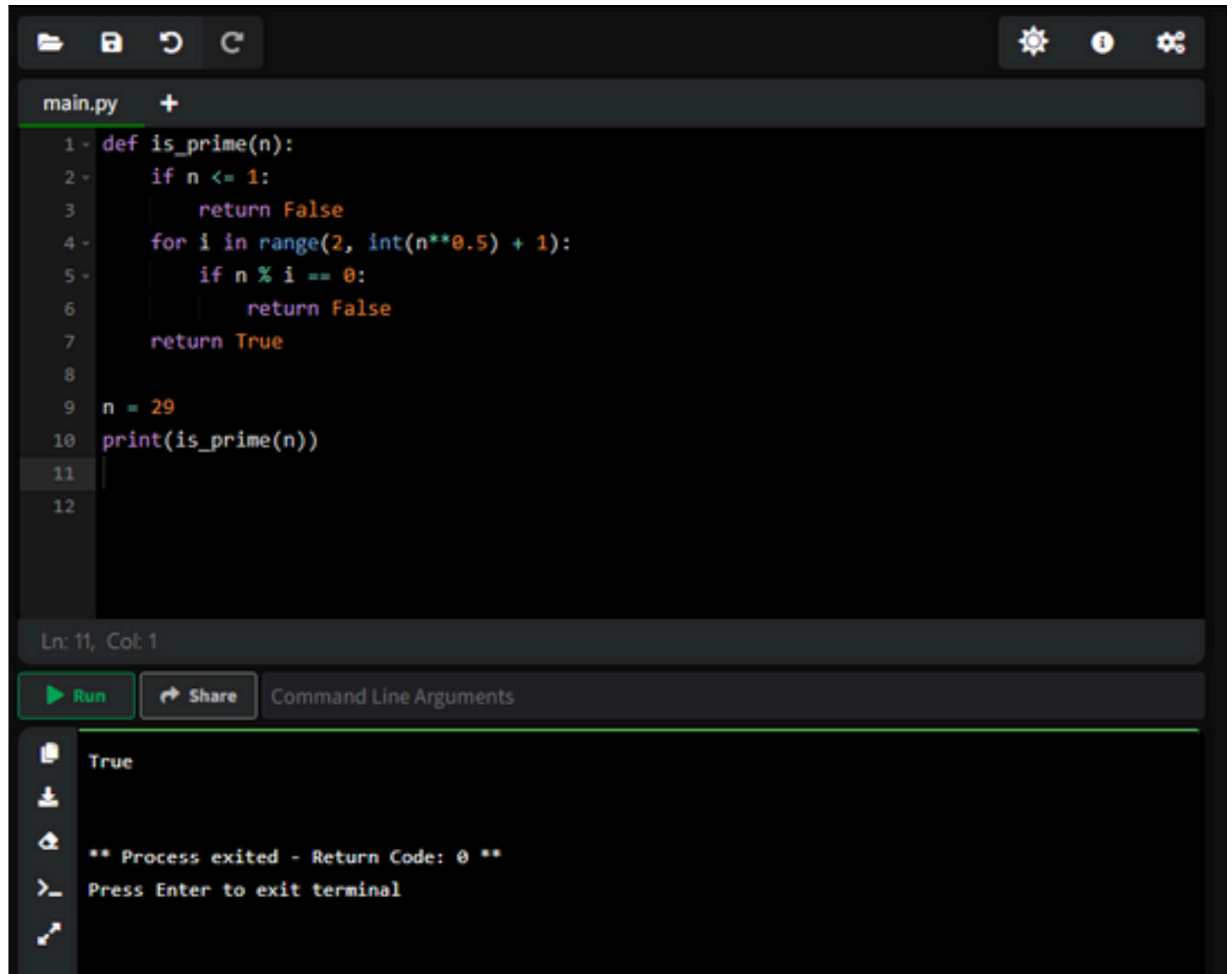
10

** Process exited - Return Code: 0 **

> Press Enter to exit terminal

4. Prime Number Check

Write a function to check if a given number is prime using a loop. For example, `is_prime(29)` should return `True`, and `is_prime(10)` should return `False`.



```
main.py +
1 def is_prime(n):
2     if n <= 1:
3         return False
4     for i in range(2, int(n**0.5) + 1):
5         if n % i == 0:
6             return False
7     return True
8
9 n = 29
10 print(is_prime(n))
11
12
```

Ln: 11, Col: 1

Run **Share** Command Line Arguments

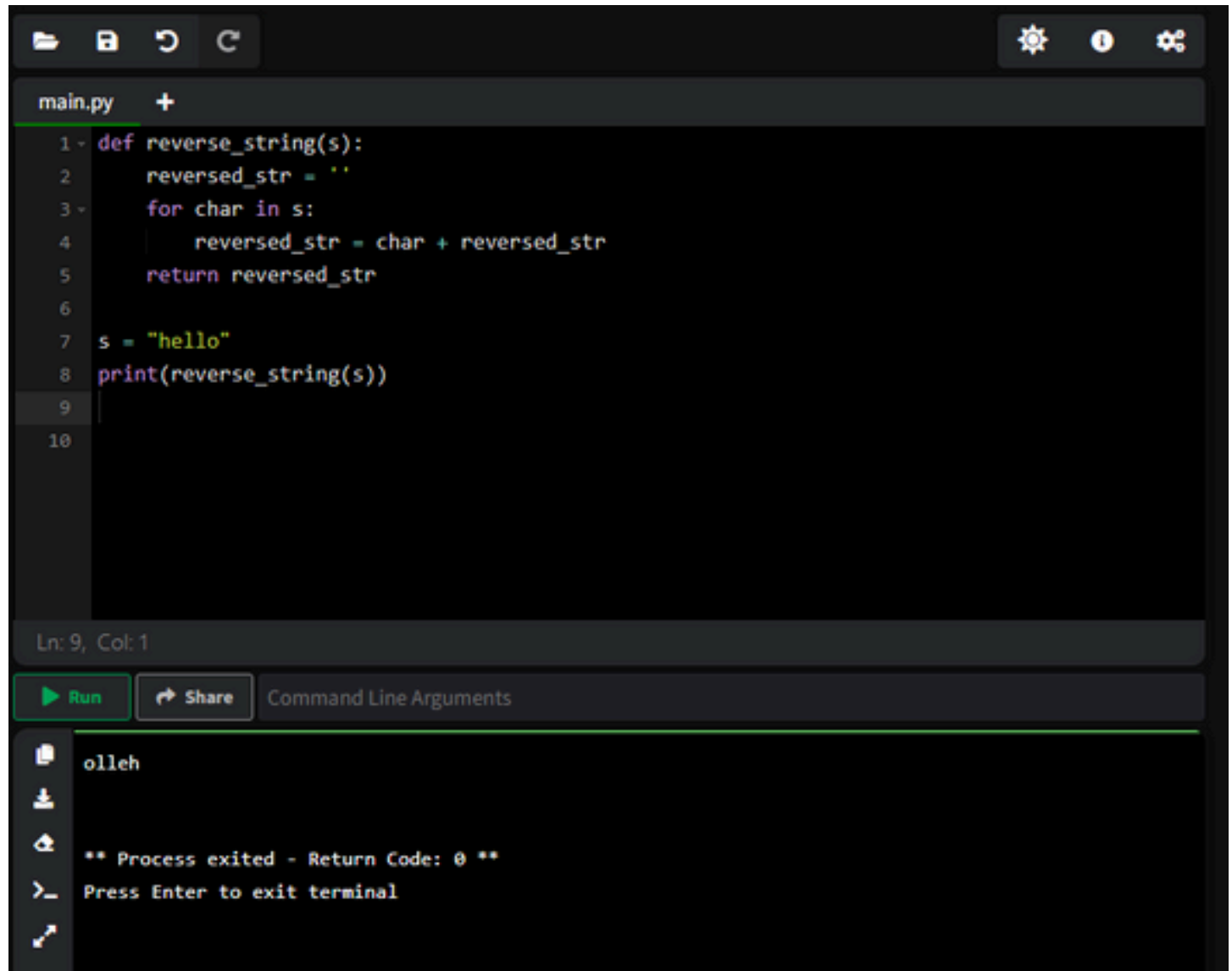
True

** Process exited - Return Code: 0 **

>_ Press Enter to exit terminal

5. Reverse a String

Write a function to reverse a given string using a loop. For example, for the input "hello", the output should be "olleh".



The image shows a code editor interface with a dark theme. At the top, there is a toolbar with icons for file operations (save, open, undo, redo) and settings (gear, info, share). Below the toolbar, the editor displays a file named 'main.py'. The code in the editor is as follows:

```
1 def reverse_string(s):
2     reversed_str = ''
3     for char in s:
4         reversed_str = char + reversed_str
5     return reversed_str
6
7 s = "hello"
8 print(reverse_string(s))
9
10
```

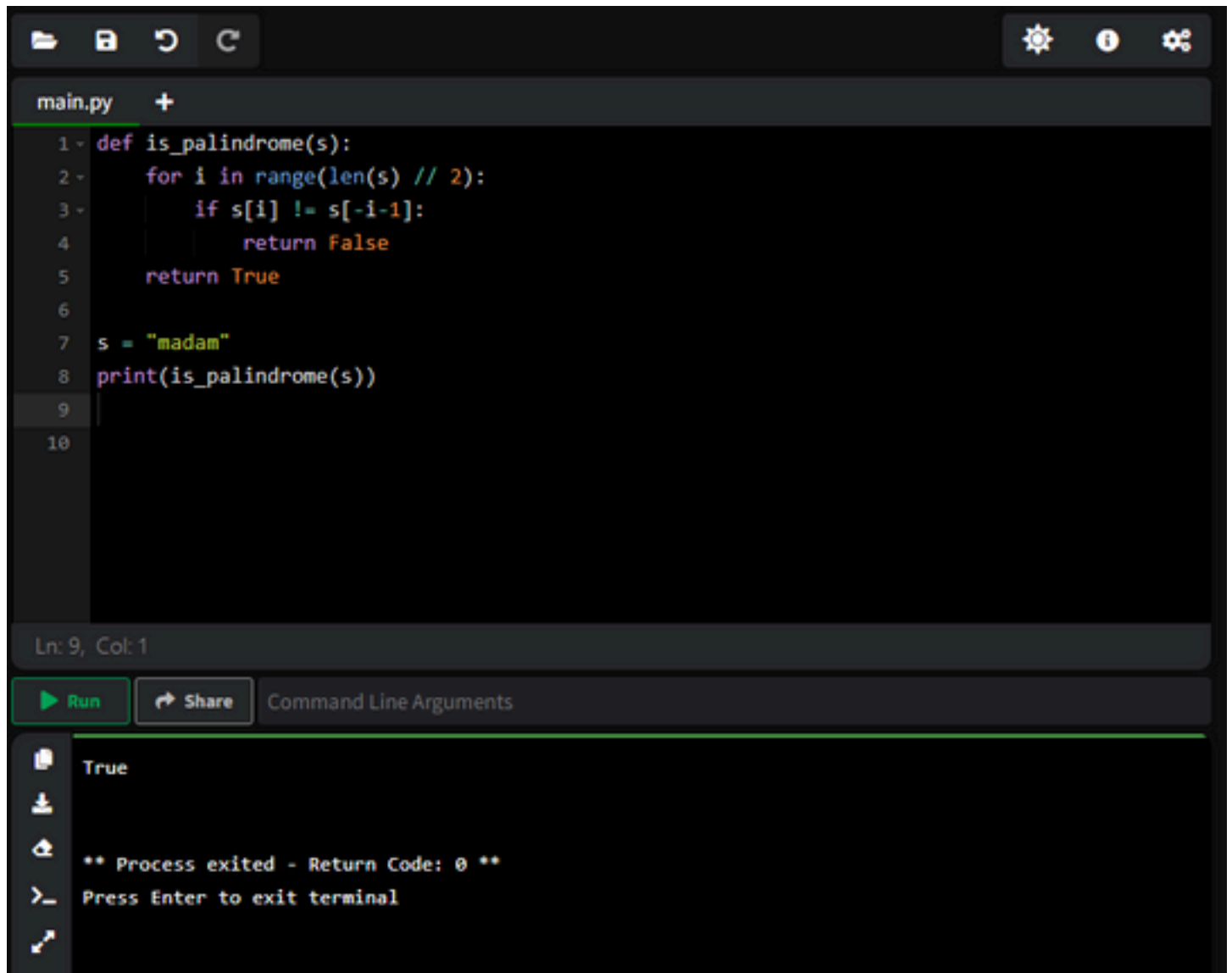
Below the code editor, there is a status bar showing 'Ln: 9, Col: 1'. To the left of the terminal, there are buttons for 'Run' (a green play icon), 'Share' (a share icon), and 'Command Line Arguments'. The terminal window at the bottom shows the output of the program:

```
olleh

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

6. Palindrome Check

Write a function to check if a given string is a palindrome using a loop. For example, "madam" is a palindrome.



The image shows a code editor window with a dark theme. The editor has a toolbar at the top with icons for file operations (new, open, save, undo, redo) and settings. The file name 'main.py' is displayed in the top left. The code is as follows:

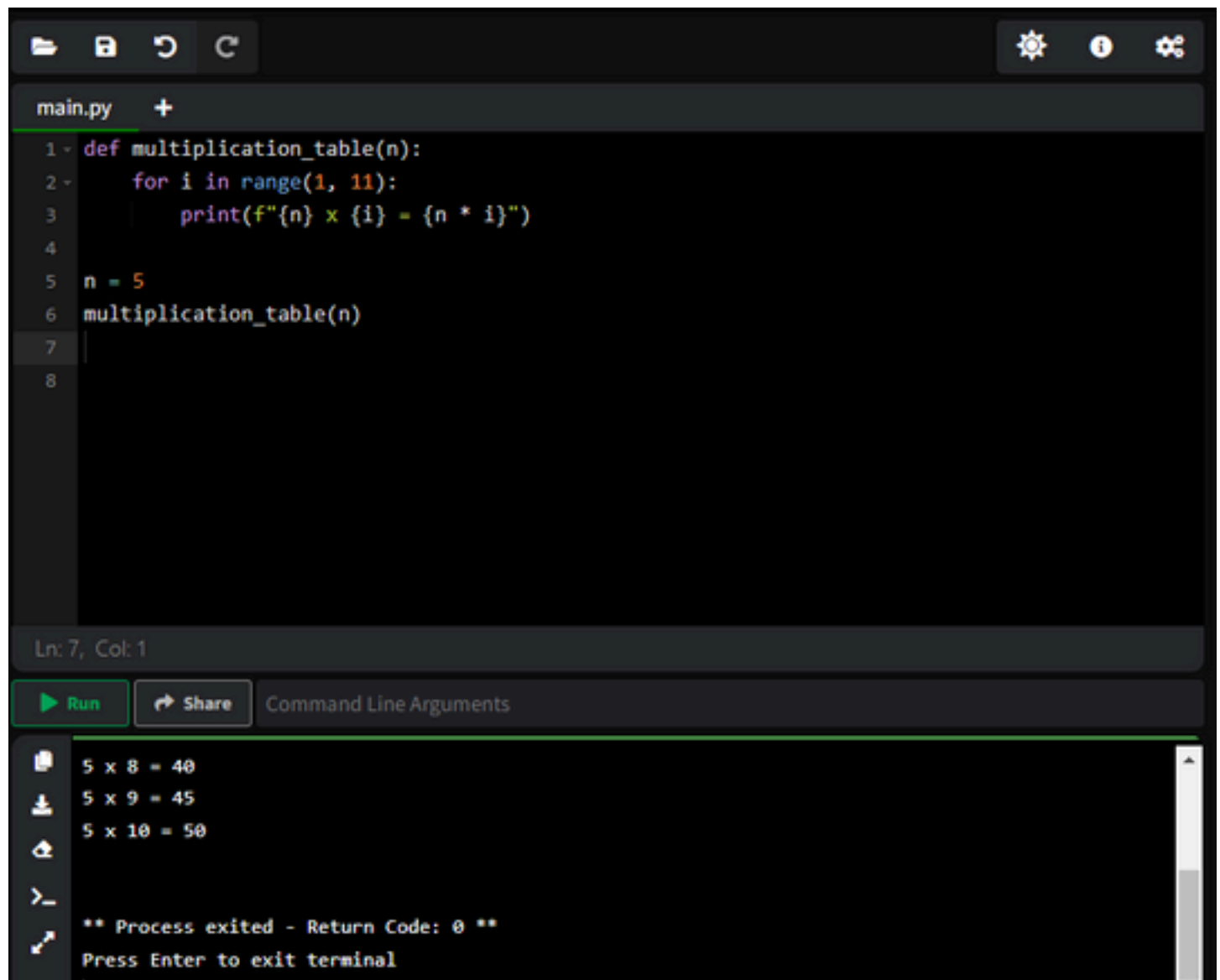
```
1 ~ def is_palindrome(s):
2 ~     for i in range(len(s) // 2):
3 ~         if s[i] != s[-i-1]:
4 ~             return False
5 ~     return True
6
7 s = "madam"
8 print(is_palindrome(s))
9
10
```

Below the code editor, there is a status bar showing 'Ln: 9, Col: 1'. Below that, there are buttons for 'Run' (with a green play icon), 'Share' (with a share icon), and a text input field for 'Command Line Arguments'. At the bottom, there is a terminal window with the following output:

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

7. Print Multiplication Table

Write a function that prints the multiplication table for a given number n up to 10.



The screenshot shows a code editor with a dark theme. The editor has a toolbar at the top with icons for file operations and settings. The file name 'main.py' is visible. The code defines a function 'multiplication_table(n)' that iterates from 1 to 10 and prints the multiplication of 'n' by each number. The variable 'n' is set to 5, and the function is called. Below the code editor, there is a status bar showing 'Ln: 7, Col: 1'. At the bottom, there is a 'Run' button, a 'Share' button, and a 'Command Line Arguments' field. The output of the program is displayed in a terminal window at the bottom, showing the multiplication results for 5 x 8, 5 x 9, and 5 x 10, followed by a process exit message.

```
main.py +
1 def multiplication_table(n):
2     for i in range(1, 11):
3         print(f"{n} x {i} = {n * i}")
4
5 n = 5
6 multiplication_table(n)
7
8
```

Ln: 7, Col: 1

Run Share Command Line Arguments

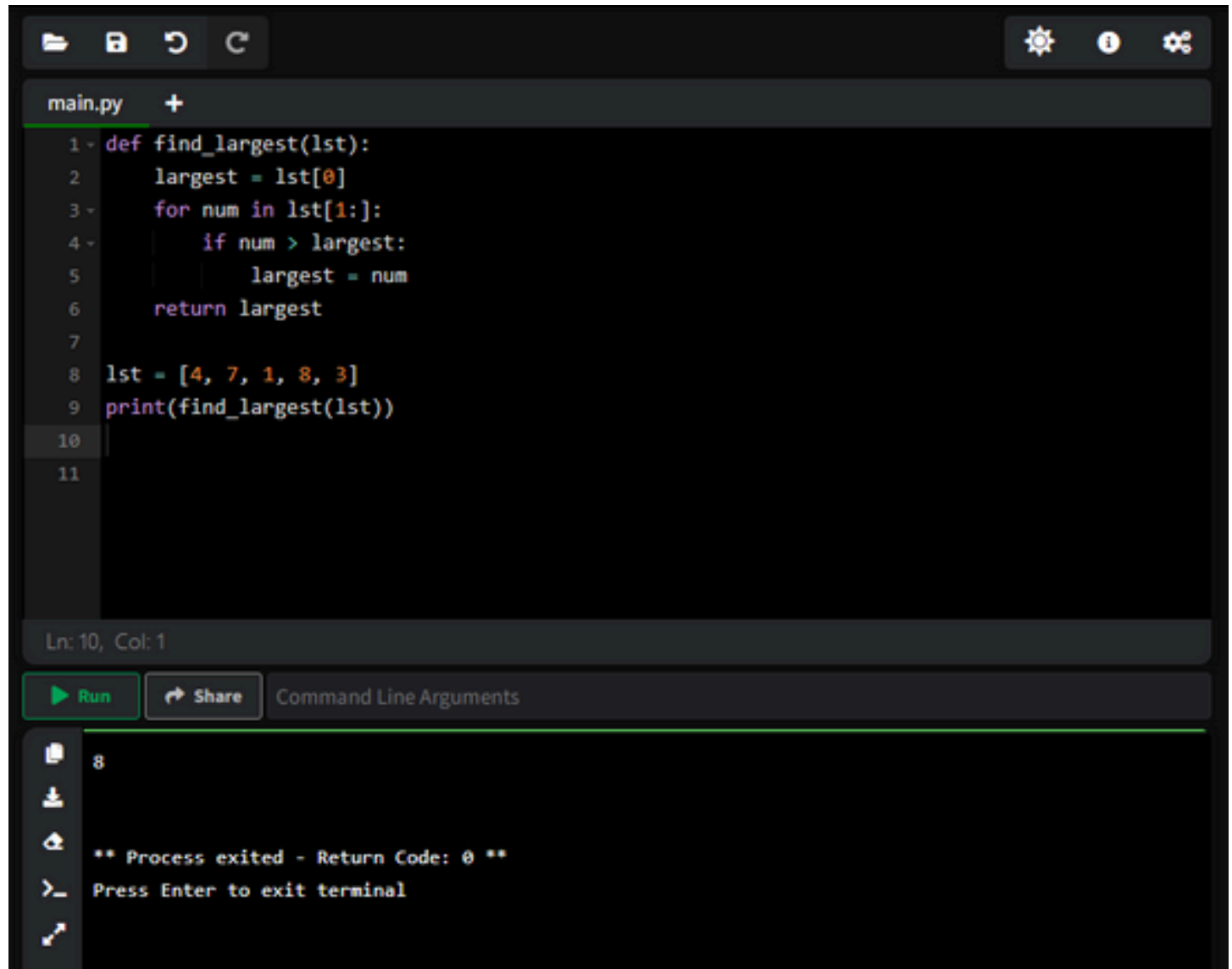
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

>_

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

8. Find the Largest Number in a List

Write a function that finds the largest number in a list using a loop. For example, given [4, 7, 1, 8, 3], the output should be 8.



The image shows a code editor window with a dark theme. The editor has a toolbar at the top with icons for file operations and settings. The main area displays a Python script in a file named 'main.py'. The script defines a function 'find_largest' that iterates through a list to find the maximum value. Below the function definition, a list 'lst' is initialized with the values [4, 7, 1, 8, 3], and the function is called with 'lst' as an argument. The status bar at the bottom indicates the cursor is at line 10, column 1. Below the editor, there are buttons for 'Run' and 'Share', and a section for 'Command Line Arguments'. The output of the program is shown in a terminal window at the bottom, displaying the number '8' and a message indicating the process exited successfully.

```
main.py +
1 def find_largest(lst):
2     largest = lst[0]
3     for num in lst[1:]:
4         if num > largest:
5             largest = num
6     return largest
7
8 lst = [4, 7, 1, 8, 3]
9 print(find_largest(lst))
10
11
```

Ln: 10, Col: 1

Run Share Command Line Arguments

8

** Process exited - Return Code: 0 **

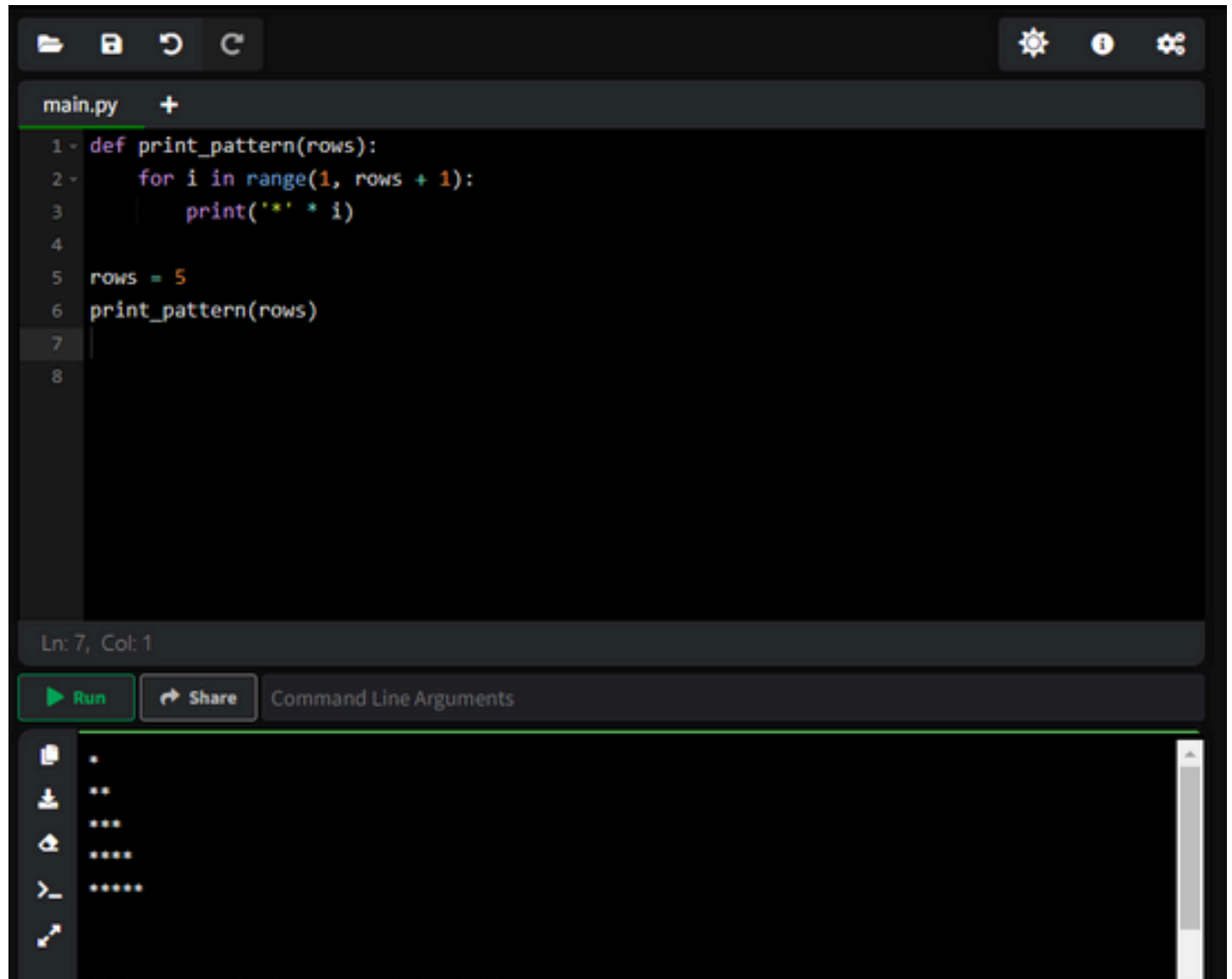
>_ Press Enter to exit terminal

10. Print a Pattern

Write a function that prints a pattern of stars based on the number of rows provided. For example, if rows = 5, the output should be:

*

**



The screenshot shows a code editor with a dark theme. The editor has a toolbar at the top with icons for file operations (save, undo, redo) and settings. The file name 'main.py' is displayed in the top left of the editor area. The code is as follows:

```
1 def print_pattern(rows):
2     for i in range(1, rows + 1):
3         print('*' * i)
4
5 rows = 5
6 print_pattern(rows)
7
8
```

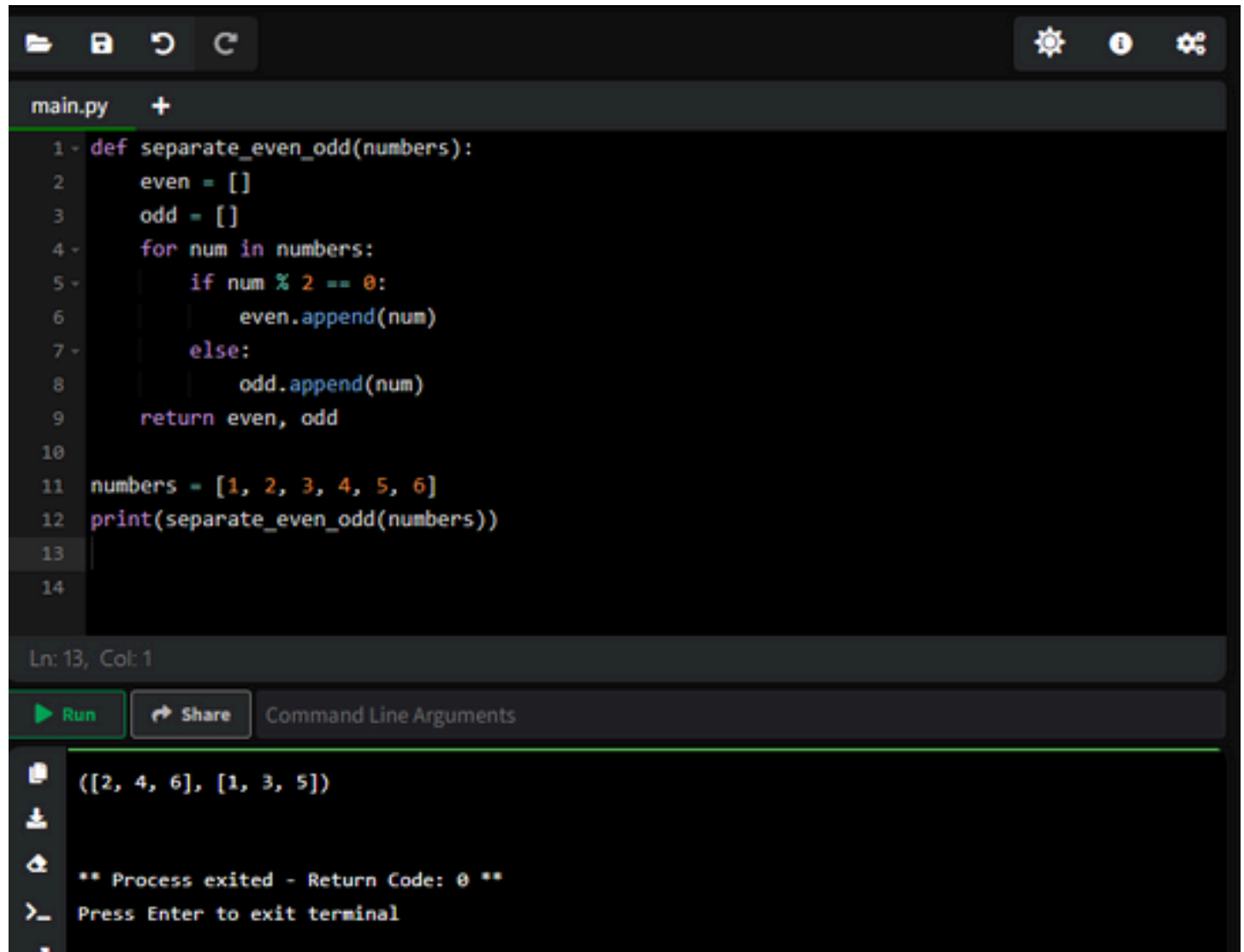
Below the code editor, there is a status bar showing 'Ln: 7, Col: 1'. At the bottom of the editor, there are buttons for 'Run' (a green play icon) and 'Share' (a share icon), followed by a text input field for 'Command Line Arguments'. Below the editor, there is a terminal window showing the output of the program:

```
*
**
***
****
*****
```

The terminal window also has a toolbar on the left with icons for copy, paste, and other terminal functions.

11. Even and Odd Numbers

Write a function that takes a list of integers and returns two lists: one containing all the even numbers and the other containing all the odd numbers.



```
main.py +
1 - def separate_even_odd(numbers):
2     even = []
3     odd = []
4     for num in numbers:
5         if num % 2 == 0:
6             even.append(num)
7         else:
8             odd.append(num)
9     return even, odd
10
11 numbers = [1, 2, 3, 4, 5, 6]
12 print(separate_even_odd(numbers))
13
14
```

Ln: 13, Col: 1

Run Share Command Line Arguments

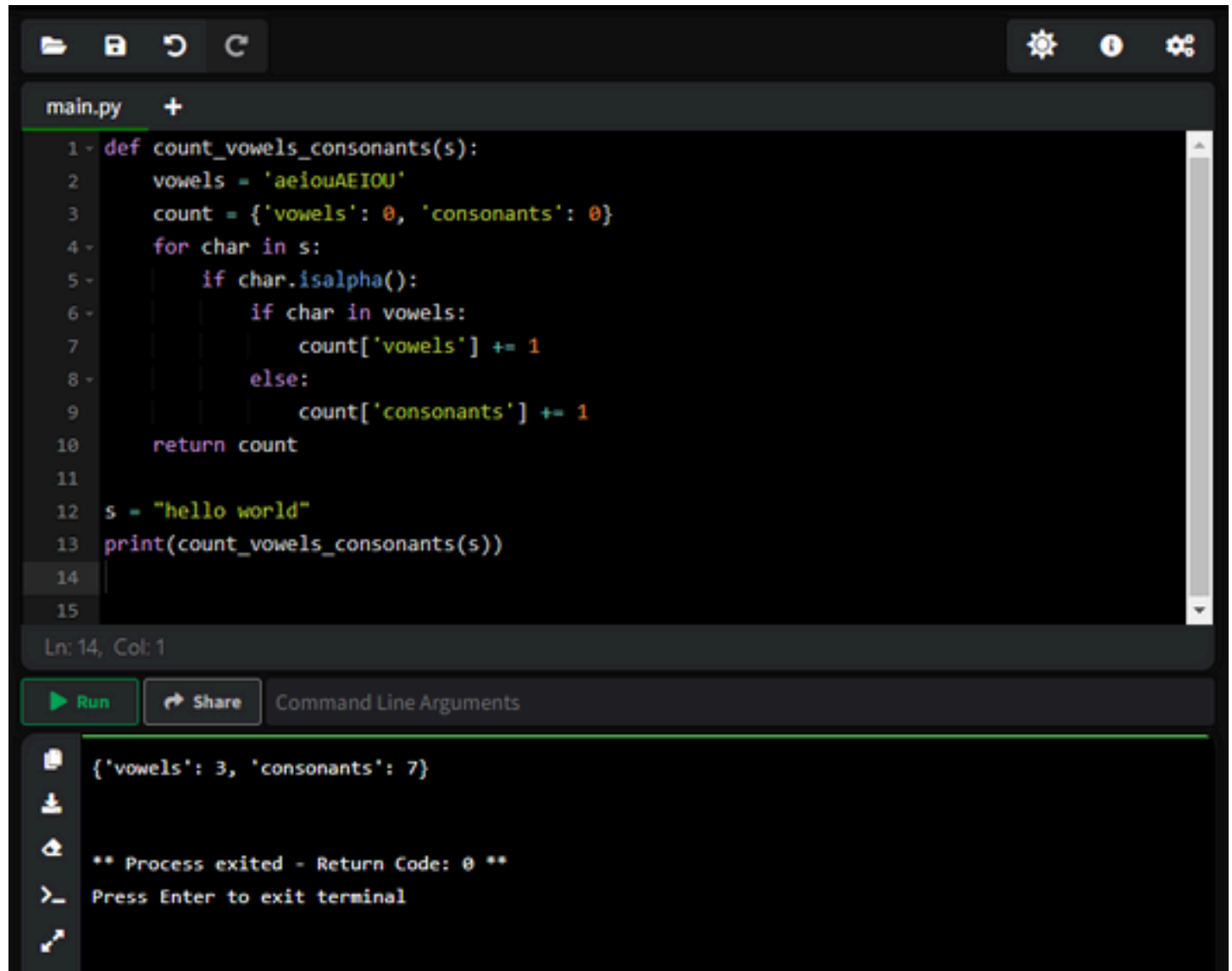
[[2, 4, 6], [1, 3, 5]]

** Process exited - Return Code: 0 **

>_ Press Enter to exit terminal

12. Count Vowels and Consonants

Write a function that takes a string and returns the count of vowels and consonants. For example, for the input "hello world", the output should be {'vowels': 3, 'consonants': 7}.



The image shows a code editor window with a dark theme. The editor has a toolbar at the top with icons for file operations and settings. The main area displays a Python script in a file named 'main.py'. The script defines a function 'count_vowels_consonants(s)' that iterates through each character in the string 's'. It uses 'char.isalpha()' to check if a character is a letter. If it is, it checks if it's in the string 'aeiouAEIOU'. If yes, it increments the 'vowels' count; otherwise, it increments the 'consonants' count. After the loop, it returns the count dictionary. Below the function, the string 'hello world' is assigned to 's', and the function is called with 's' as an argument, with the result printed. The status bar at the bottom indicates 'Ln: 14, Col: 1'. Below the editor, there are buttons for 'Run' and 'Share', and a section for 'Command Line Arguments'. The output of the program is shown in a terminal-like window at the bottom, displaying the dictionary {'vowels': 3, 'consonants': 7} and a message indicating the process exited successfully.

```
main.py +
1 - def count_vowels_consonants(s):
2     vowels = 'aeiouAEIOU'
3     count = {'vowels': 0, 'consonants': 0}
4     for char in s:
5         if char.isalpha():
6             if char in vowels:
7                 count['vowels'] += 1
8             else:
9                 count['consonants'] += 1
10    return count
11
12    s = "hello world"
13    print(count_vowels_consonants(s))
14
15
```

Ln: 14, Col: 1

Run Share Command Line Arguments

```
{'vowels': 3, 'consonants': 7}
** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

13. Number Pattern

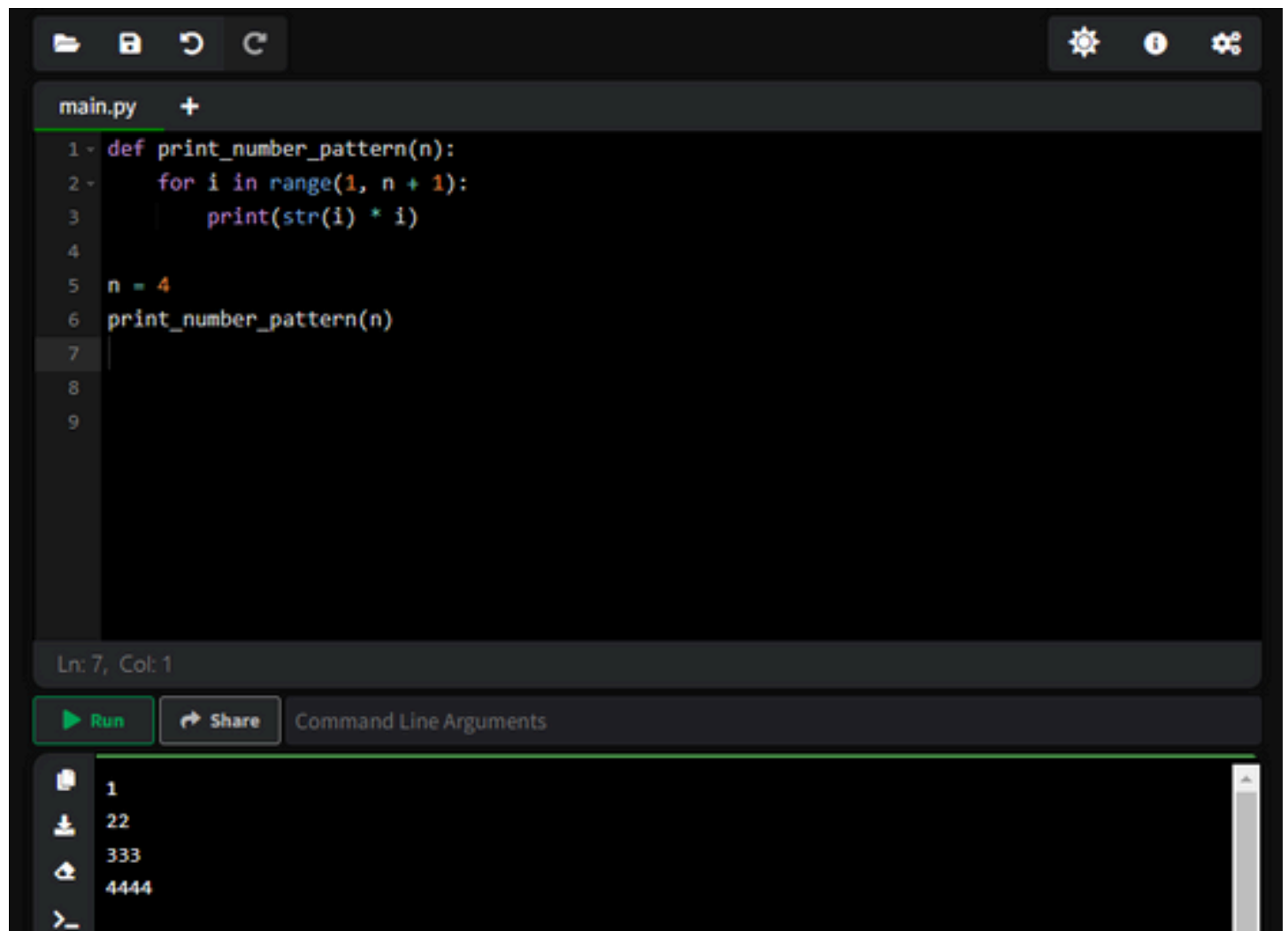
Write a function that prints a number pattern based on a given integer n. For example, for n = 4, the output should be:

1

22

333

4444



The screenshot shows a code editor with a dark theme. The editor has a toolbar at the top with icons for file operations and settings. The file name 'main.py' is visible in the top left. The code is as follows:

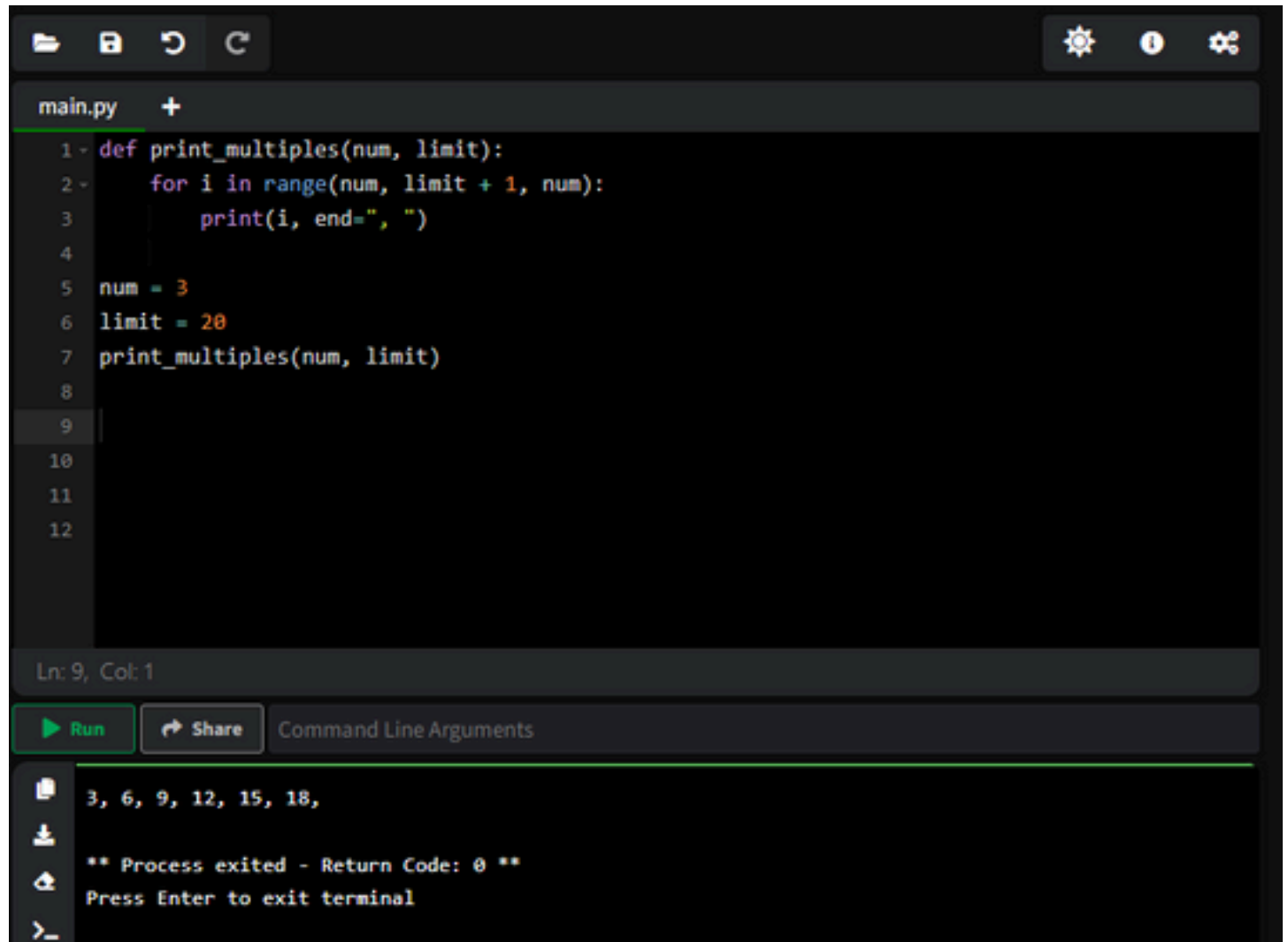
```
1 - def print_number_pattern(n):  
2 -     for i in range(1, n + 1):  
3 -         print(str(i) * i)  
4 -  
5 - n = 4  
6 - print_number_pattern(n)  
7 -  
8 -  
9 -
```

Below the code editor, there is a status bar showing 'Ln: 7, Col: 1'. There are also buttons for 'Run' and 'Share', and a text input field for 'Command Line Arguments'. At the bottom, there is a terminal window showing the output of the program:

```
1  
22  
333  
4444
```

14. Multiples of a Number

Write a function that prints all multiples of a given number up to a specified limit. For example, for `num = 3` and `limit = 20`, the output should be 3, 6, 9, 12, 15, 18.



The screenshot shows a code editor with a dark theme. The top bar has icons for file operations (save, undo, redo) and settings. The editor window displays a Python file named `main.py` with the following code:

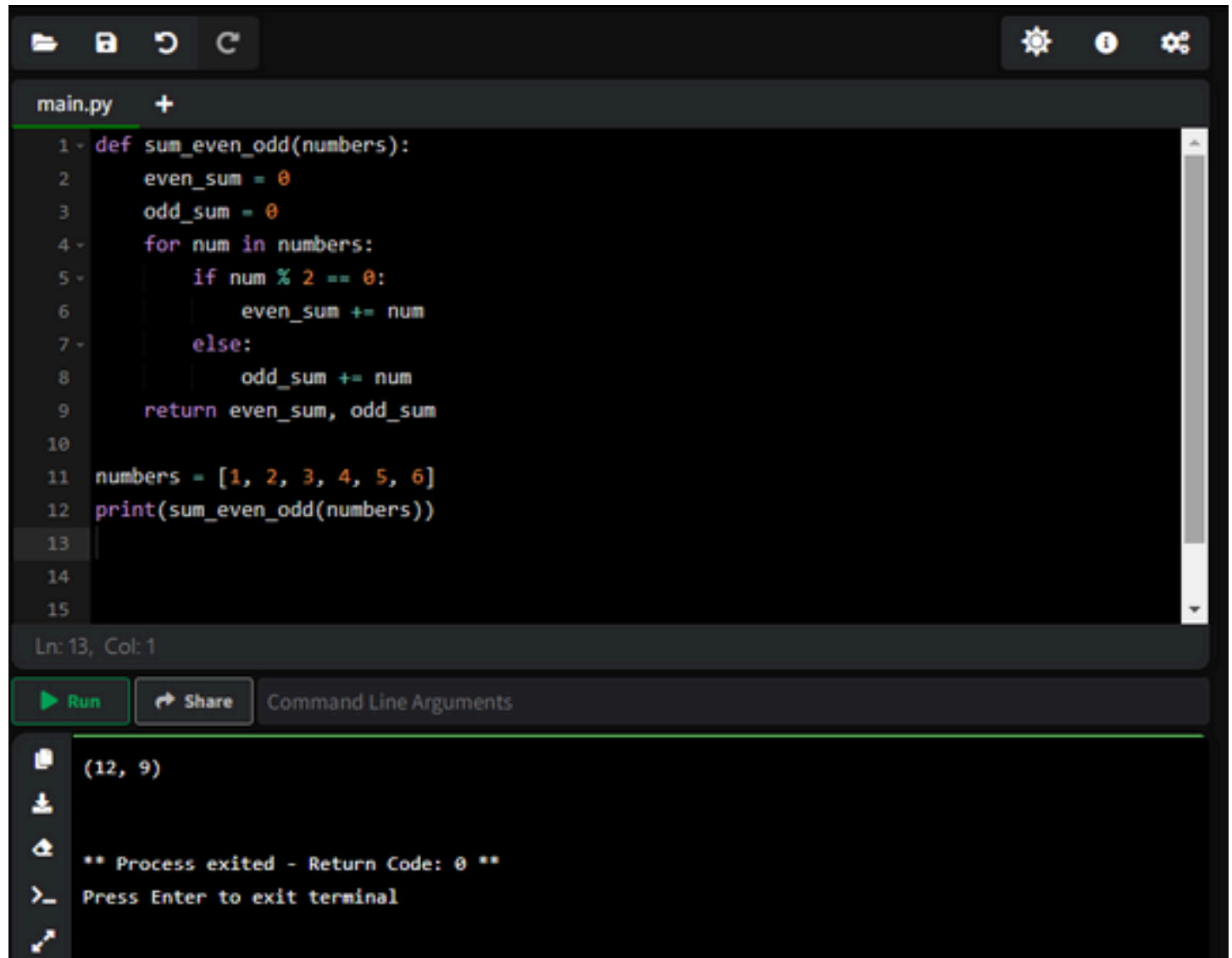
```
1 def print_multiples(num, limit):
2     for i in range(num, limit + 1, num):
3         print(i, end=", ")
4
5 num = 3
6 limit = 20
7 print_multiples(num, limit)
8
9
10
11
12
```

The status bar at the bottom of the editor indicates "Ln: 9, Col: 1". Below the editor, there are buttons for "Run" (a green play icon) and "Share" (a share icon), followed by a text input field labeled "Command Line Arguments".

Below the editor, the output of the program is displayed in a terminal window. It shows the output "3, 6, 9, 12, 15, 18," followed by a newline. Below that, it shows "** Process exited - Return Code: 0 **" and "Press Enter to exit terminal". The terminal prompt is ">_".

15. Sum of Even and Odd Numbers in a List

Write a function that takes a list of integers and returns a tuple with the sum of even numbers and the sum of odd numbers. For example, for the list [1, 2, 3, 4, 5, 6], the output should be (12, 9).



The image shows a code editor interface with a dark theme. The editor displays a Python script named `main.py`. The script defines a function `sum_even_odd(numbers)` that iterates through a list of numbers, calculating the sum of even and odd numbers separately. Below the function definition, a list `numbers = [1, 2, 3, 4, 5, 6]` is defined, and the function is called with `print(sum_even_odd(numbers))`. The editor's status bar indicates the cursor is at line 13, column 1. Below the editor, there are buttons for `Run` and `Share`, and a section for `Command Line Arguments`. The output of the program is shown in a terminal window at the bottom, displaying the tuple `(12, 9)` and a message indicating the process exited successfully.

```
main.py +
1 def sum_even_odd(numbers):
2     even_sum = 0
3     odd_sum = 0
4     for num in numbers:
5         if num % 2 == 0:
6             even_sum += num
7         else:
8             odd_sum += num
9     return even_sum, odd_sum
10
11 numbers = [1, 2, 3, 4, 5, 6]
12 print(sum_even_odd(numbers))
13
14
15
```

Ln: 13, Col: 1

Run **Share** Command Line Arguments

(12, 9)

** Process exited - Return Code: 0 **

> Press Enter to exit terminal