

1.Two Sum Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order. Example 1:

Input: `nums = [2,7,11,15]`, `target = 9`

Output: `[0,1]`

Explanation: Because `nums[0] + nums[1] == 9`, we return `[0, 1]`. Example 2: Input: `nums = [3,2,4]`, `target = 6` Output: `[1,2]` Example 3: Input: `nums = [3,3]`, `target = 6` Output: `[0,1]`

main.py

Save

Run

```
1 a = [2, 7, 11, 15]
2 b = int(input("Enter target sum:"))
3 for i in range(len(a)):
4     for j in range(i + 1, len(a)):
5         if a[i] + a[j] == b:
6             print(a[i], ",", a[j])
```

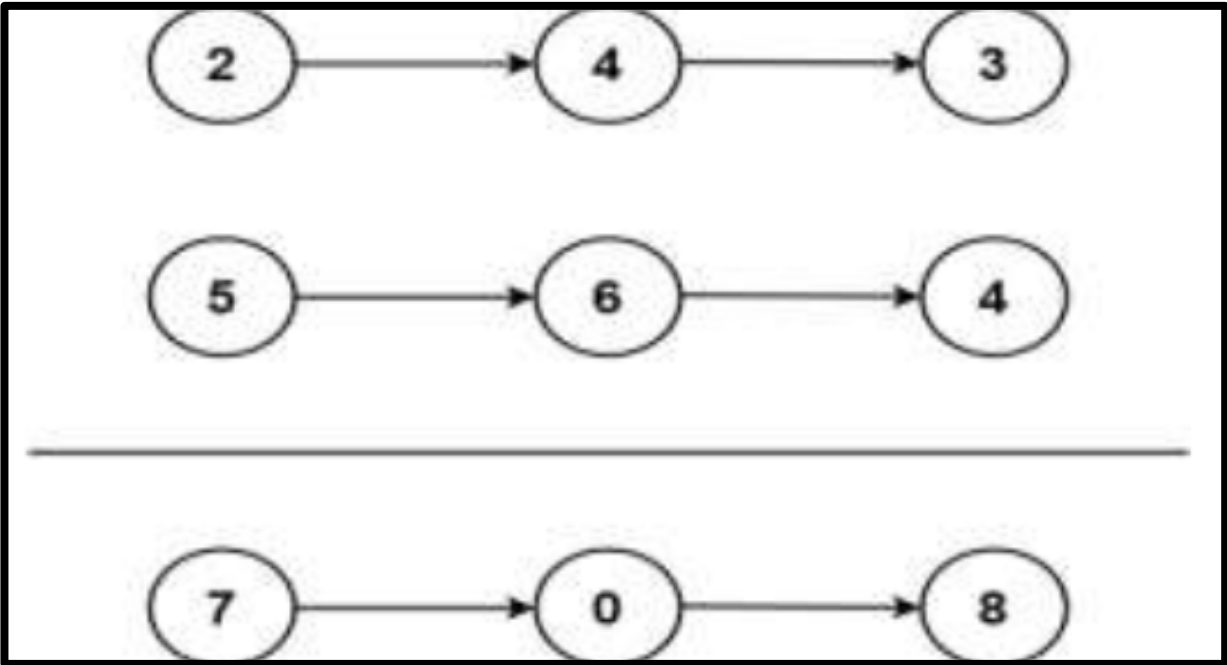
Output

Clear

Enter target sum:9
2 , 7

=== Code Execution Successful ===

2. Add Two Numbers You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.



Example 1: Input: `l1 = [2,4,3]`, `l2 = [5,6,4]` Output: `[7,0,8]` Explanation: `342 + 465 = 807`.

```
main.py  [ ] [ ] Save Run Output Clear
1  l1=[2,4,3]
2  l2=[5,6,4]
3  l3=[]
4  for i in range(len(l1)):
5      s=l1[i]+l2[i]
6      if s>=10:
7          l3.append(int(str(s)[-1]))
8      else:
9          l3.append(s)
10 for j in range(len(l3)):
11     print(l3[j],end="," )

7,0,7,
=== Code Execution Successful ===
```

3. Longest Substring without Repeating Characters Given a string s, find the length of the longest substring without repeating characters.

Example 1: Input: s = "abcabcbb" Output: 3 Explanation: The answer is "abc", with the length of 3.

Example 2: Input: s = "bbbbbb" Output: 1 Explanation: The answer is "b", with the length of 1

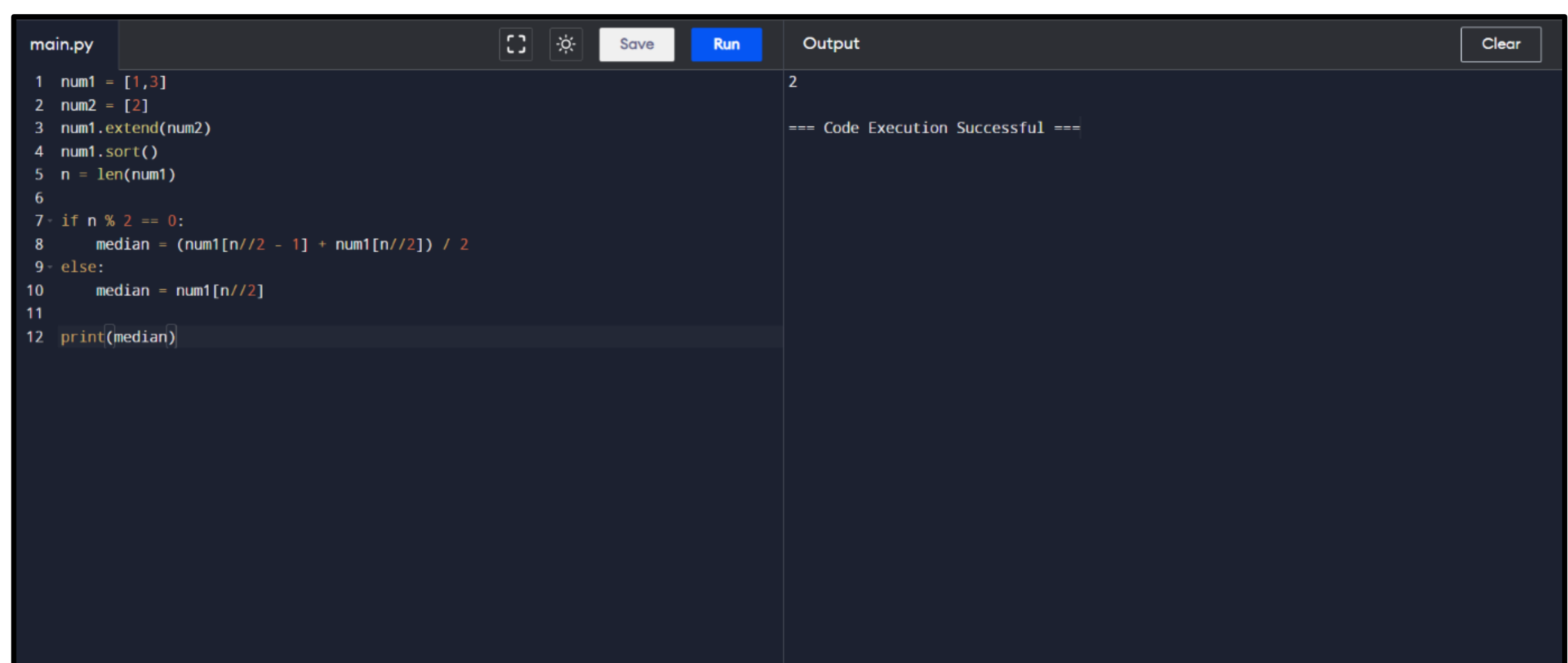
```
main.py  [ ] [ ] Save Run Output Clear
1  s1 = str(input("Enter a string with spaces"))
2  s = list(s1.split())
3  l = []
4  m=0
5  for c in s:
6      if c not in l:
7          l.append(c)
8          m = max(m,len(l))
9      else:
10         l=l[l.index(c)+1:]
11         l.append(c)
12 print(len(l))

Enter a string with spaces dha ranidha rani
2
=== Code Execution Successful ===
```

4. Median of Two Sorted Arrays Given two sorted arrays `nums1` and `nums2` of size `m` and `n` respectively, return the median of the two sorted arrays. The overall run time complexity should be $O(\log(m+n))$.

Example 1: Input: `nums1 = [1,3]`, `nums2 = [2]` Output: 2.00000 Explanation: merged array = `[1,2,3]` and median is 2.

Example 2: Input: `nums1 = [1,2]`, `nums2 = [3,4]` Output: 2.50000 Explanation: merged array = `[1,2,3,4]` and median is $(2 + 3) / 2 = 2.5$



```
main.py
1 num1 = [1,3]
2 num2 = [2]
3 num1.extend(num2)
4 num1.sort()
5 n = len(num1)
6
7 if n % 2 == 0:
8     median = (num1[n//2 - 1] + num1[n//2]) / 2
9 else:
10    median = num1[n//2]
11
12 print(median)
```

Output

```
2
=== Code Execution Successful ===
```

5. Longest Palindromic Substring Given a string `s`, return the longest palindromic substring in `s`.

Example 1: Input: `s = "babad"` Output: `"bab"` Explanation: `"aba"` is also a valid answer.

Example 2: Input: `s = "cbbd"` Output: `"bb"`

main.py

Save

Run

Output

Clear

```
1 def is_pali(s):
2     return s == s[::-1]
3
4 longest_palindrome = ""
5 s = input("Enter a string")
6 for i in range(len(s)):
7     for j in range(i+1, len(s)-1):
8         sub = s[i:j]
9         if is_pali(sub) and len(sub) > len(longest_palindrome):
10             longest_palindrome = sub
11
12 print("Longest palindrome substring:", longest_palindrome)
```

Enter a string dharudharu
Longest palindrome substring: d

=== Code Execution Successful ===

6. Zigzag Conversion The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility) P A H N A P L S I I G Y I R And then read line by line: "PAHNAPLSIIGYIR" Write the code that will take a string and make this conversion given a number of rows: string convert(string s, int numRows);
Example 1: Input: s = "PAYPALISHIRING", numRows = 3 Output: "PAHNAPLSIIGYIR"
Example 2: Input: s = "PAYPALISHIRING", numRows = 4 Output: "PINALSIGYAHRPI"

main.py

Save

Run

```
1 def convert(s: str, numRows: int) -> str:
2     if numRows == 1 or numRows >= len(s):
3         return s
4
5     rows = [''] * numRows
6     index, step = 0, 1
7
8     for char in s:
9         rows[index] += char
10        if index == 0:
11            step = 1
12        elif index == numRows - 1:
13            step = -1
14        index += step
15
16    return ''.join(rows)
17
18 s1 = "PAYPALISHIRING"
19 numRows1 = 3
20 print(convert(s1, numRows1))
21
22
```

Output

Clear

PAHNAPLSIIGYIR

=== Code Execution Successful ===

7. Reverse Integer Given a signed 32-bit integer x, return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0. Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Example 1: Input: x = 123 Output: 321

Example 2: Input: x = -123 Output: -321

main.py

Save

Run

```
1 x = input("Enter a number: ")
2 l = list(x)
3 if l[0] != "-":
4     l1 = list(map(int, l[::-1]))
5     print(*l1, sep="")
6 else:
7     l1 = list(map(int, l[1:][::-1]))
8     print("-", end="")
9     print(*l1, sep="")
```

Output

Clear

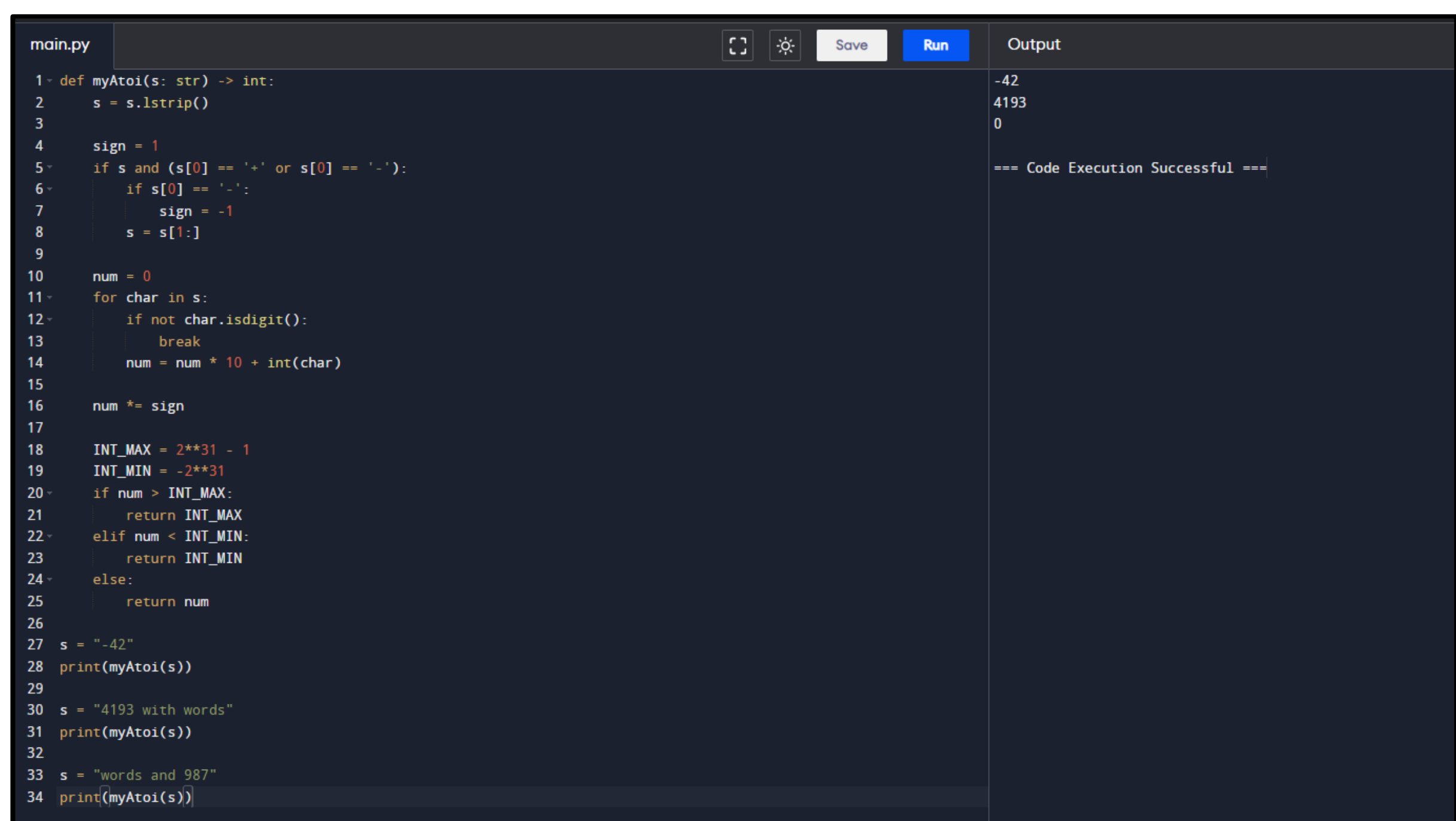
Enter a number: 123

321

=== Code Execution Successful ===

8. String to Integer (atoi) Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function). The algorithm for myAtoi(string s) is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then clamp the integer so that it remains in the range. Specifically, integers less than -2^{31} should be clamped to -2^{31} , and integers greater than $2^{31} - 1$ should be clamped to $2^{31} - 1$.
6. Return the integer as the final result



```
main.py
1 def myAtoi(s: str) -> int:
2     s = s.lstrip()
3
4     sign = 1
5     if s and (s[0] == '+' or s[0] == '-'):
6         if s[0] == '-':
7             sign = -1
8         s = s[1:]
9
10    num = 0
11    for char in s:
12        if not char.isdigit():
13            break
14        num = num * 10 + int(char)
15
16    num *= sign
17
18    INT_MAX = 2**31 - 1
19    INT_MIN = -2**31
20    if num > INT_MAX:
21        return INT_MAX
22    elif num < INT_MIN:
23        return INT_MIN
24    else:
25        return num
26
27    s = "-42"
28    print(myAtoi(s))
29
30    s = "4193 with words"
31    print(myAtoi(s))
32
33    s = "words and 987"
34    print(myAtoi(s))
```

Output

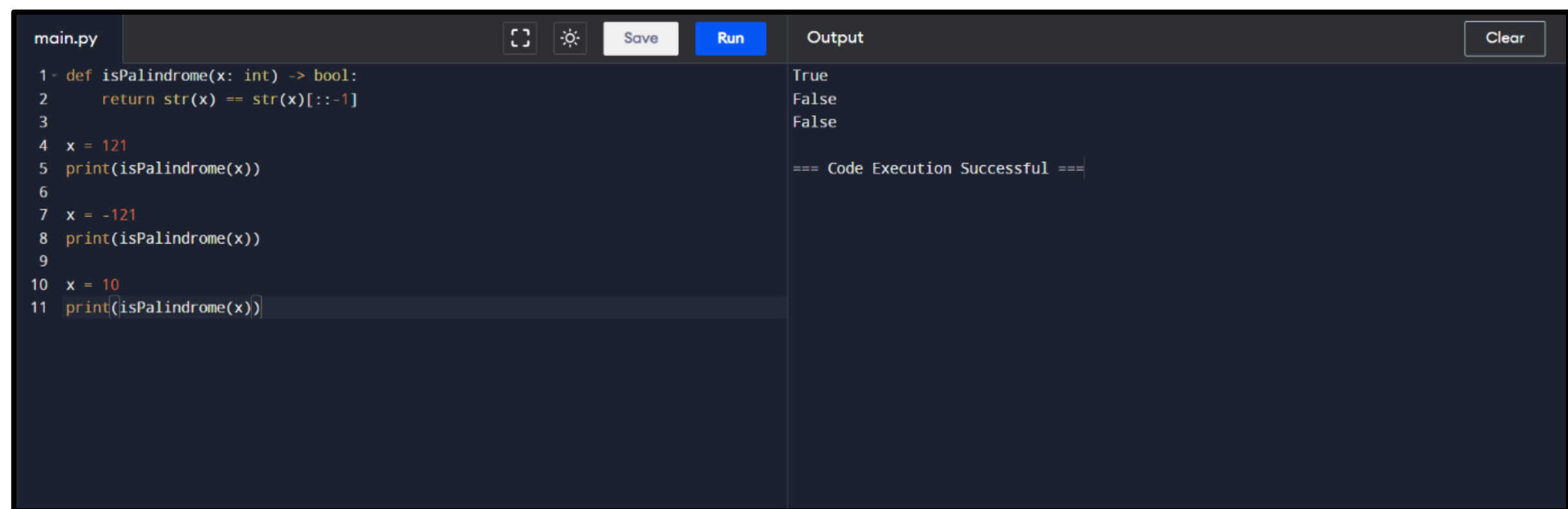
```
-42
4193
0
=== Code Execution Successful ===
```

9. Palindrome Number Given an integer x, return true if x is a palindrome, and false otherwise.

Example 1: Input: x = 121 Output: true Explanation: 121 reads as 121 from left to right and from right to left.

Example 2: Input: x = -121 Output: false Explanation: From left to right, it reads -121.

From right to left, it becomes 121-. Therefore it is not a palindrome.



The screenshot shows a code editor with a file named 'main.py'. The code defines a function `isPalindrome` that takes an integer `x` and returns a boolean. It converts `x` to a string and checks if it is equal to its reverse. The function is tested with `x = 121`, `x = -121`, and `x = 10`. The output shows that 121 is a palindrome (True), while -121 and 10 are not (False). The code execution is successful.

```
main.py
1- def isPalindrome(x: int) -> bool:
2-     return str(x) == str(x)[::-1]
3-
4- x = 121
5- print(isPalindrome(x))
6-
7- x = -121
8- print(isPalindrome(x))
9-
10 x = 10
11 print(isPalindrome(x))
```

Output

```
True
False
False

=== Code Execution Successful ===
```

10. Regular Expression Matching Given an input string `s` and a pattern `p`, implement regular expression matching with support for `'.'` and `'*'` where:

- `'.'` Matches any single character.
- `'*'` Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

Example 1: Input: `s = "aa"`, `p = "a"` Output: false Explanation: "a" does not match the entire string "aa".

main.py

Save

Run

Output

Clear

```
1- def isMatch(s: str, p: str) -> bool:
2   dp = [[False] * (len(p) + 1) for _ in range(len(s) + 1)]
3   dp[0][0] = True
4
5-   for j in range(1, len(p) + 1):
6-       if p[j - 1] == '*':
7-           dp[0][j] = dp[0][j - 2]
8
9-   for i in range(1, len(s) + 1):
10-       for j in range(1, len(p) + 1):
11-           if p[j - 1] == '.' or p[j - 1] == s[i - 1]:
12-               dp[i][j] = dp[i - 1][j - 1]
13-           elif p[j - 1] == '*':
14-               dp[i][j] = dp[i][j - 2] or (dp[i - 1][j] and (s[i - 1] == p[j - 2] or p[j - 2] == '.'))
15
16   return dp[-1][-1]
17 |
18 s = "aa"
19 p = "a*"
20 print(isMatch(s, p))
21
22 s = "mississippi"
23 p = "mis*is*p*."
24 print(isMatch(s, p))
```

True

False

=== Code Execution Successful ===