ASSIGNMENT-1

1. Two Sum

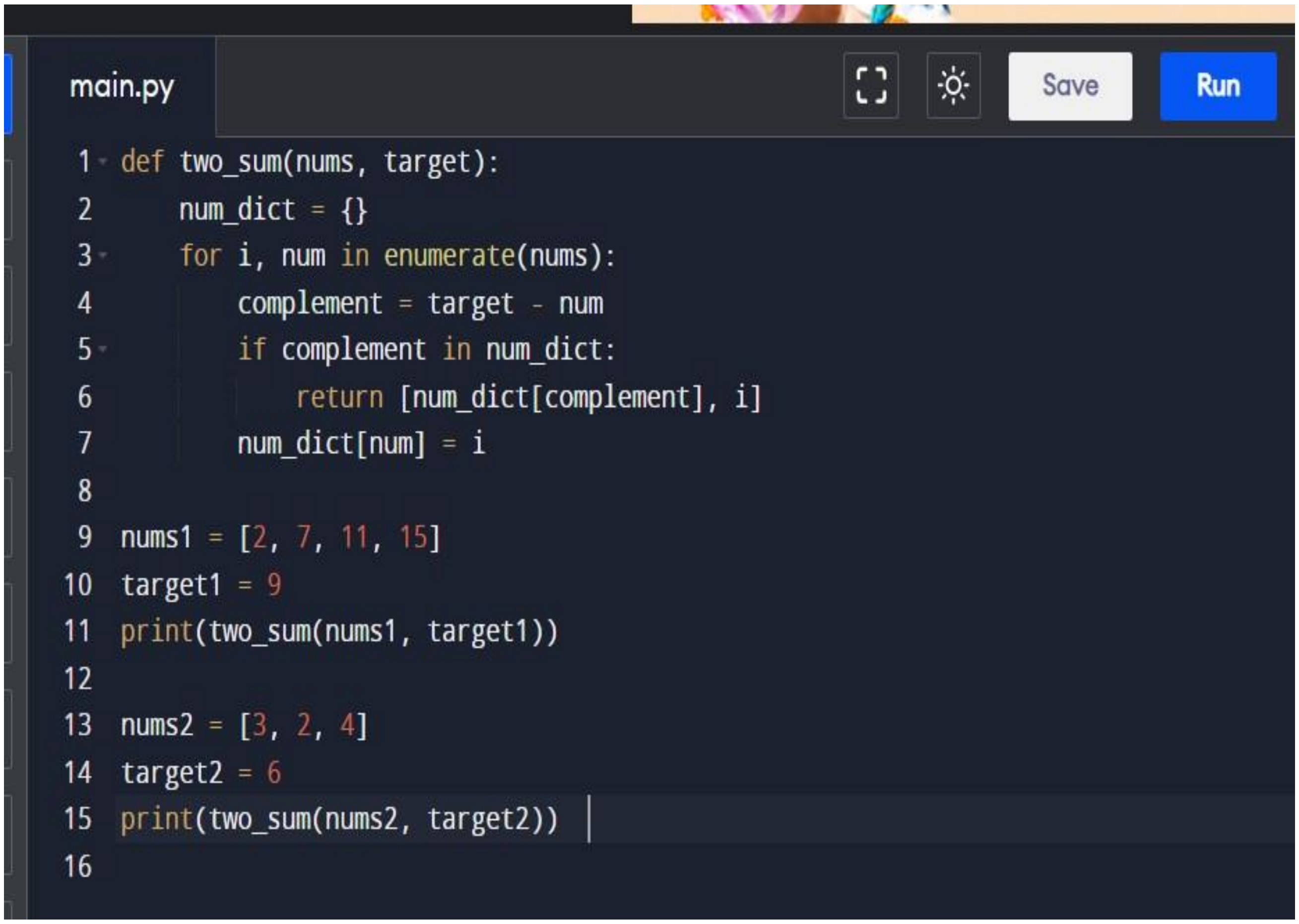
Given an array of integer nums and an integer target, return *indices of the two numbers such that they add up to the target*.

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]

Program:

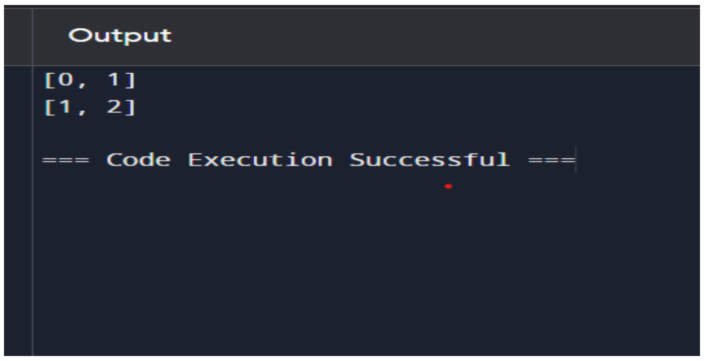


```python
def two_sum(nums, target):
    num_dict = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in num_dict:
            return [num_dict[complement], i]
        num_dict[num] = i

nums1 = [2, 7, 11, 15]
target1 = 9
print(two_sum(nums1, target1))

nums2 = [3, 2, 4]
target2 = 6
print(two_sum(nums2, target2))
```

the output of the program:

Output: [0, 1]

Output: [1,2]



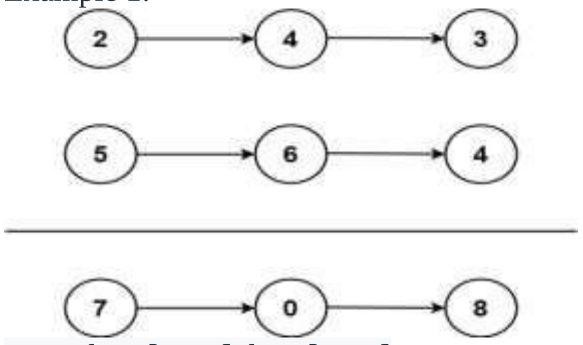
```
[0, 1]
[1, 2]

=== 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.

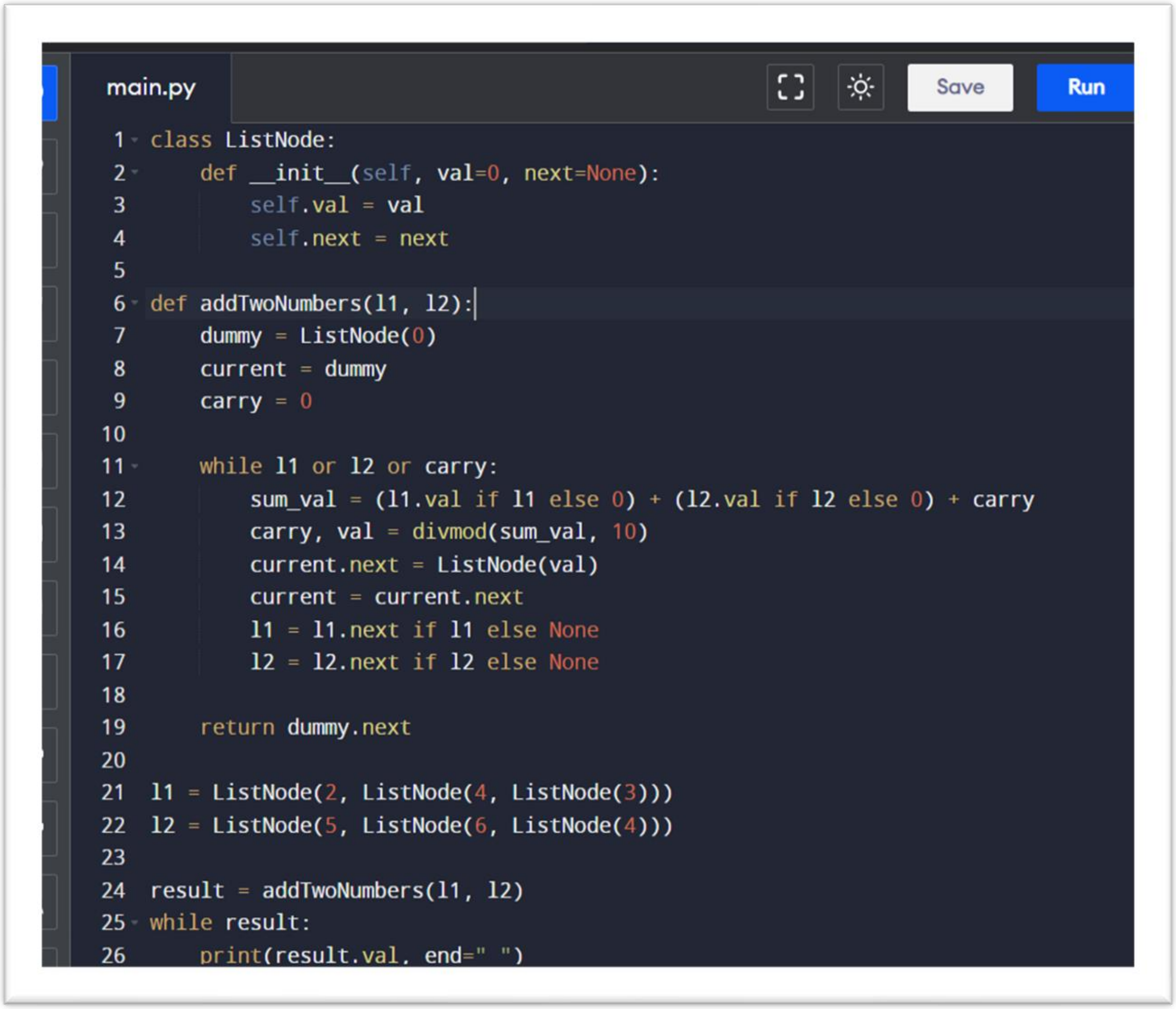
Example 1:



Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 80

Input: l1 = [0], l2 = [0]
Output: [0]

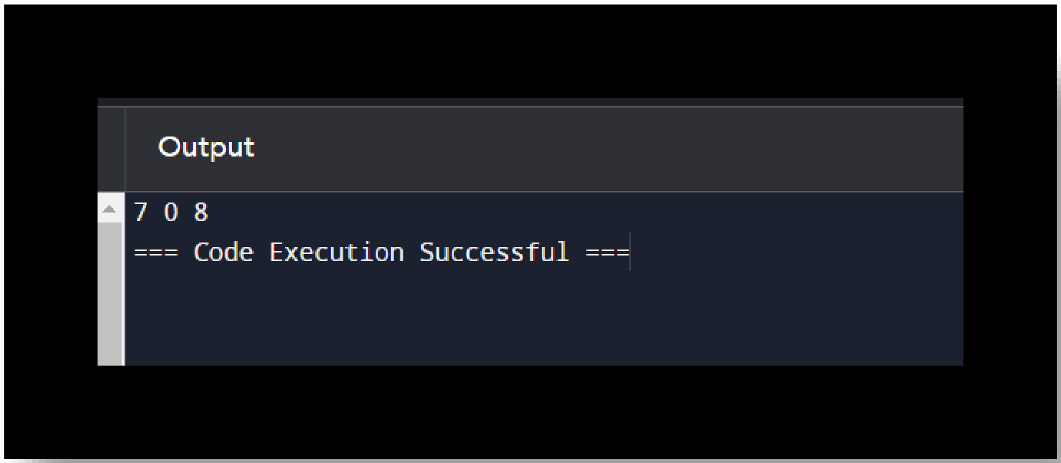
Example 3:
Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
Output: [8,9,9,9,0,0,0,1]



```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def addTwoNumbers(l1, l2):
    dummy = ListNode(0)
    current = dummy
    carry = 0

    while l1 or l2 or carry:
        sum_val = (l1.val if l1 else 0) + (l2.val if l2 else 0) + carry
        carry, val = divmod(sum_val, 10)
        current.next = ListNode(val)
        current = current.next
        l1 = l1.next if l1 else None
        l2 = l2.next if l2 else None

    return dummy.next

l1 = ListNode(2, ListNode(4, ListNode(3)))
l2 = ListNode(5, ListNode(6, ListNode(4)))

result = addTwoNumbers(l1, l2)
while result:
    print(result.val, end=" ")
```

the output is:

7 0 8



```
7 0 8
=== Code Execution Successful ===
```

3.Longest Substring without Repeating Characters

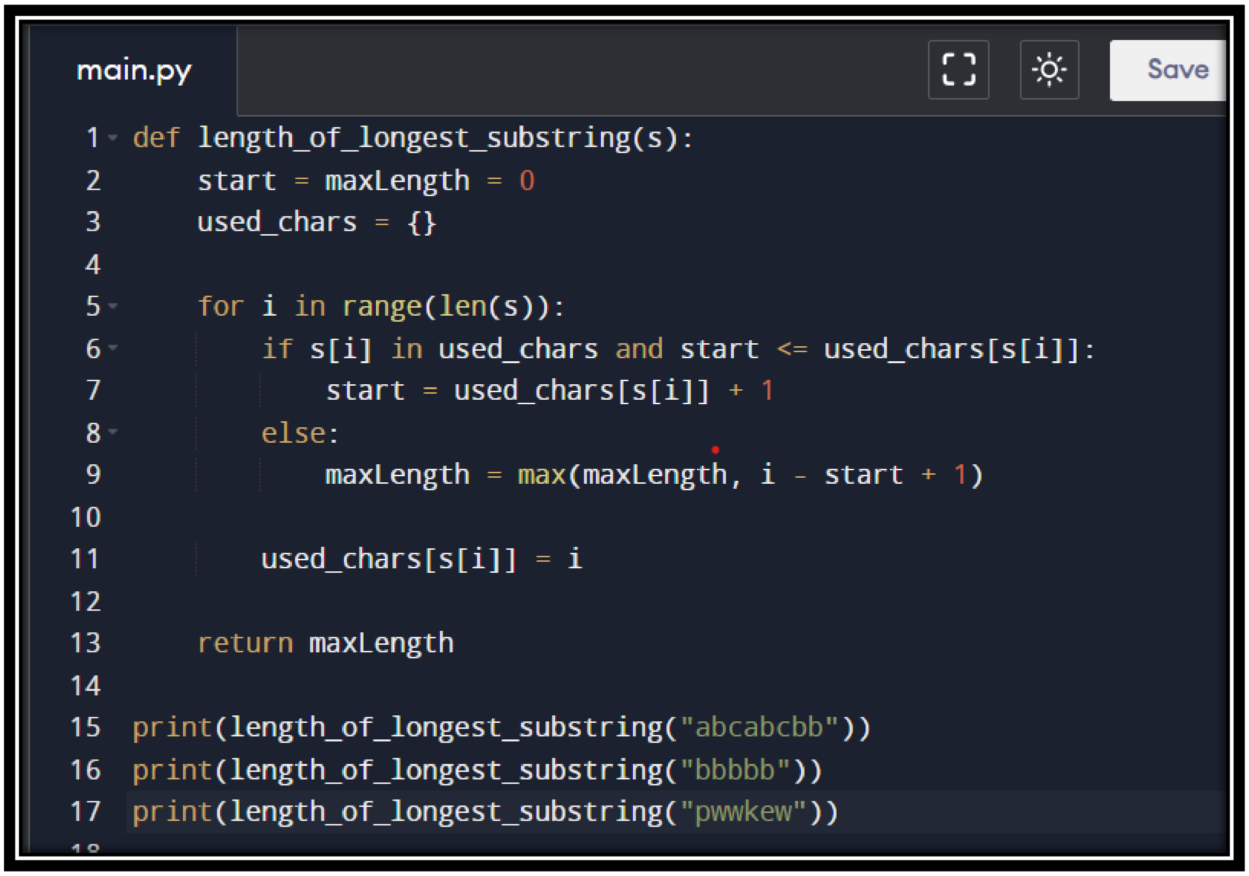
Given a string s, find the length of the longest substring without repeating characters.

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

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

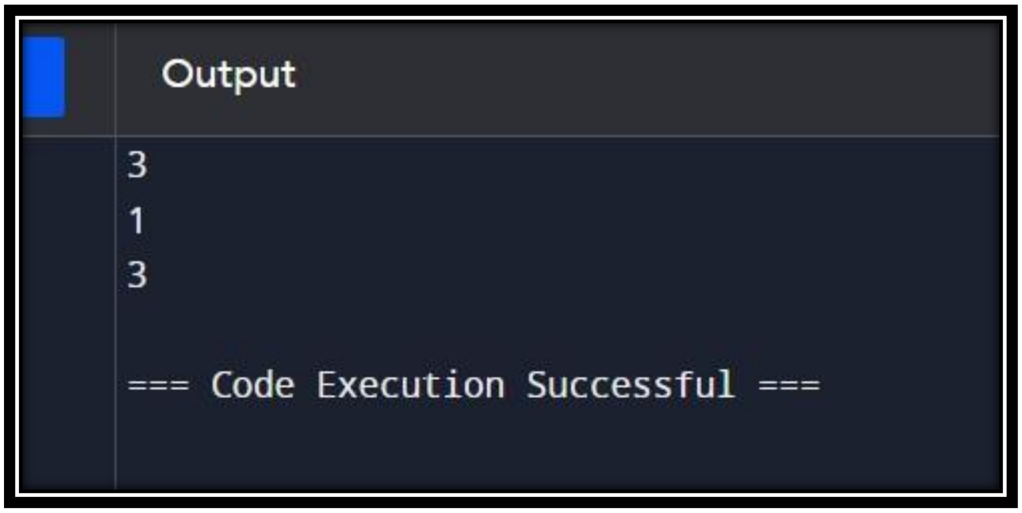
Example 3:
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Program:



```python
def length_of_longest_substring(s):
    start = maxLength = 0
    used_chars = {}

    for i in range(len(s)):
        if s[i] in used_chars and start <= used_chars[s[i]]:
            start = used_chars[s[i]] + 1
        else:
            maxLength = max(maxLength, i - start + 1)

        used_chars[s[i]] = i

    return maxLength

print(length_of_longest_substring("abcabcbb"))
print(length_of_longest_substring("bbbbb"))
print(length_of_longest_substring("pwwkew"))
```

The output is:



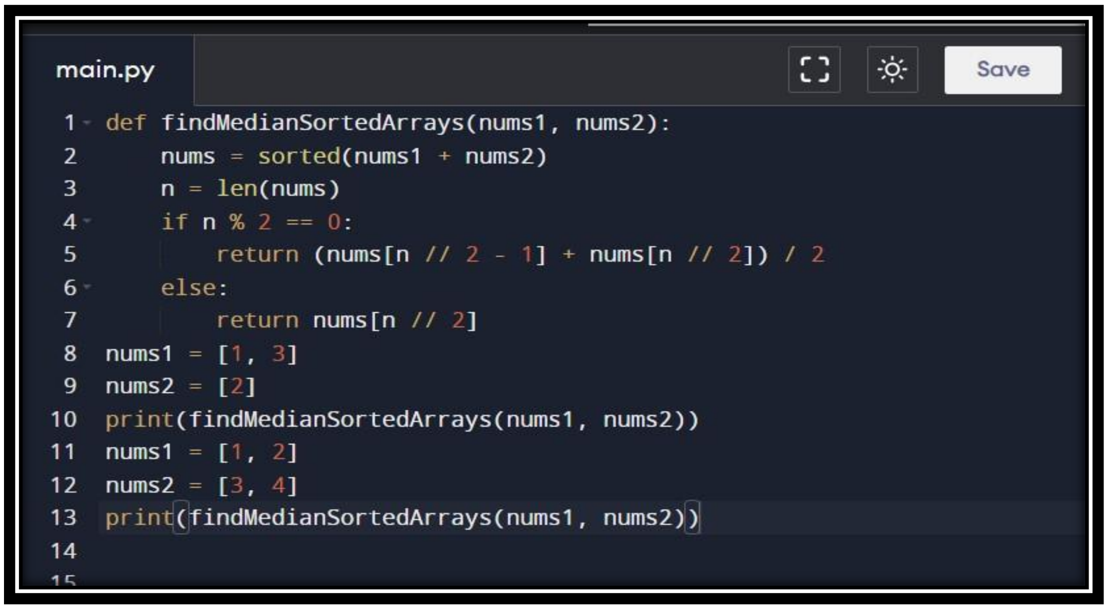
```
3
1
3

=== 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)).

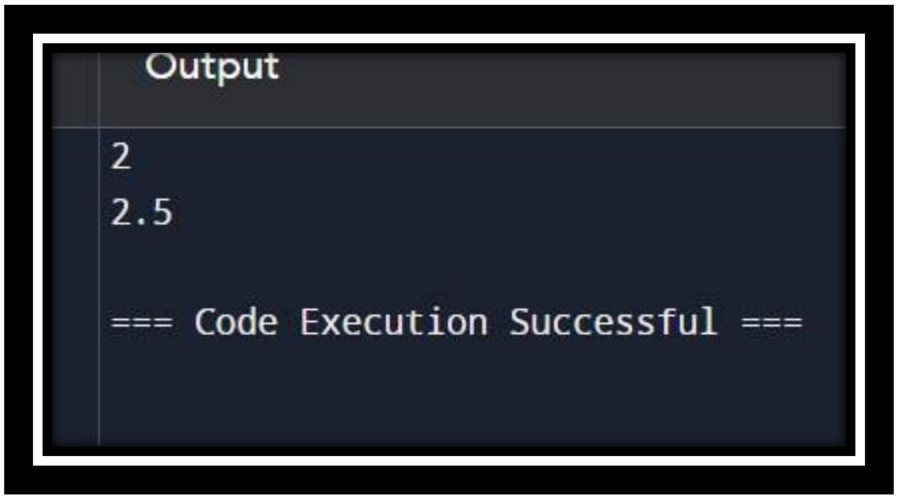
Example 1:

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



```python
def findMedianSortedArrays(nums1, nums2):
    nums = sorted(nums1 + nums2)
    n = len(nums)
    if n % 2 == 0:
        return (nums[n // 2 - 1] + nums[n // 2]) / 2
    else:
        return nums[n // 2]
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))
nums1 = [1, 2]
nums2 = [3, 4]
print(findMedianSortedArrays(nums1, nums2))
```

Output:

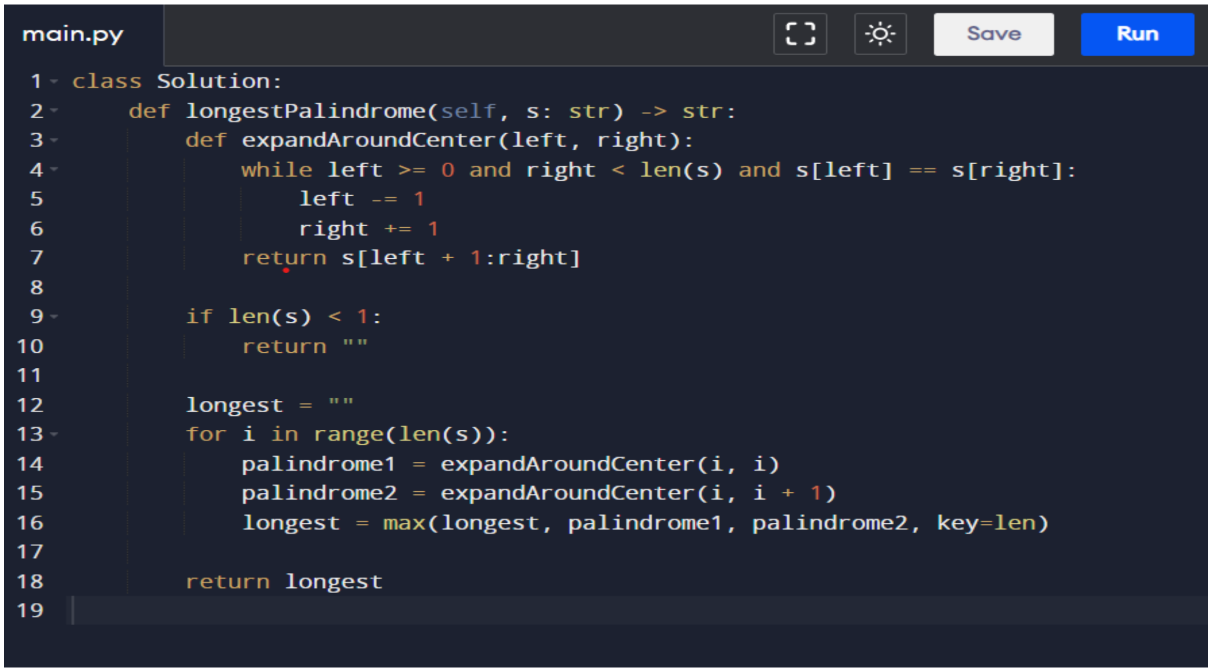

Output
```
2
2.5

=== 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"
Program:



```python
class Solution:
    def longestPalindrome(self, s: str) -> str:
        def expandAroundCenter(left, right):
            while left >= 0 and right < len(s) and s[left] == s[right]:
                left -= 1
                right += 1
            return s[left + 1:right]

        if len(s) < 1:
            return ""

        longest = ""
        for i in range(len(s)):
            palindrome1 = expandAroundCenter(i, i)
            palindrome2 = expandAroundCenter(i, i + 1)
            longest = max(longest, palindrome1, palindrome2, key=len)

        return longest

```

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);

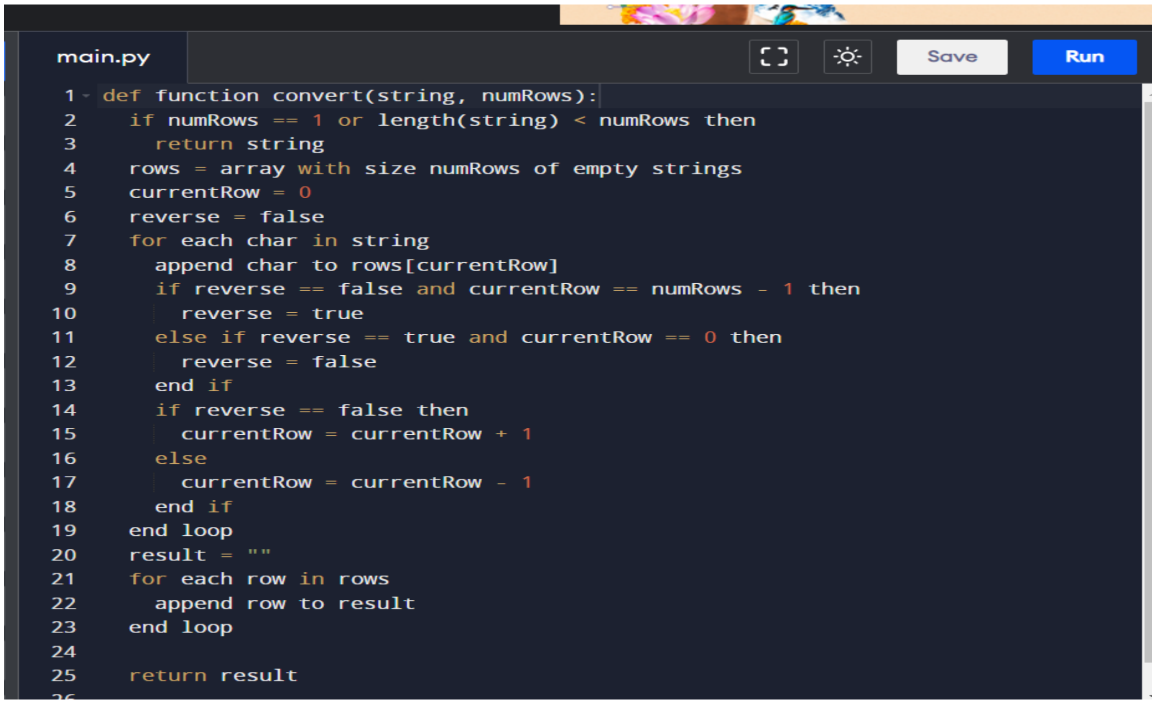
Example 1:

Input: s = "PAYPALISHIRING", numRows = 3 Output: "PAHNAPLSIIGYIR"

Example 2:

Input: s =
"PAYPALISHIRING
", numRows = 4
Output:
"PINALSIGYAHRPI

Program:



```
def function convert(string, numRows):
  if numRows == 1 or length(string) < numRows then
    return string
  rows = array with size numRows of empty strings
  currentRow = 0
  reverse = false
  for each char in string
    append char to rows[currentRow]
    if reverse == false and currentRow == numRows - 1 then
      reverse = true
    else if reverse == true and currentRow == 0 then
      reverse = false
    end if
    if reverse == false then
      currentRow = currentRow + 1
    else
      currentRow = currentRow - 1
    end if
  end loop
  result = ""
  for each row in rows
    append row to result
  end loop

  return result
```

7. Reverse Integer

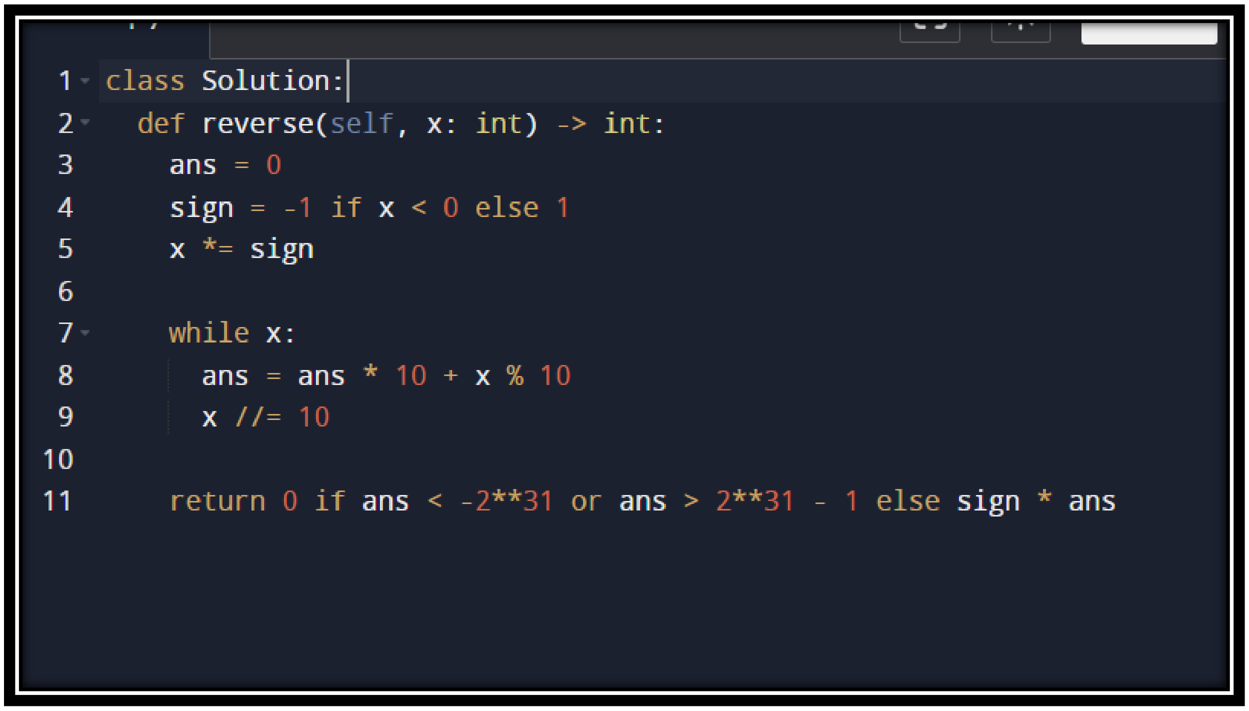
Given a signed 32-bit integer x, return x *with its digits reversed*. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0.
Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Example 1:

Input: x = 123
Output: 321

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

Example 3:
Input: x = 120
Output: 21
Program:



```python
class Solution:
  def reverse(self, x: int) -> int:
    ans = 0
    sign = -1 if x < 0 else 1
    x *= sign

    while x:
      ans = ans * 10 + x % 10
      x //= 10

    return 0 if ans < -2**31 or ans > 2**31 - 1 else sign * ans
```

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:

a. Read in and ignore any leading whitespace.
  b. 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.

  c. 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.
  d. 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).
  e. If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in the range. Specifically, integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1.

f. Return the integer as the final result.

Note:

  i. Only the space character ' ' is considered a whitespace character.
  ii. Do not ignore any characters other than the leading whitespace or the rest of the string after the digits.

Example 1:
Input: s = "42"
Output: 42
Explanation: The underlined characters are what is read in, the caret is the current reader position.
Step 1: "42" (no characters read because there is no leading whitespace)
         ^

Step 2: "42" (no characters read because there is neither a '-' nor '+')
         ^

Step 3: "42" ("42" is read in)
           ^

The parsed integer is 42.
Since 42 is in the range [-231, 231 - 1], the final result is 42.

Example 2:
Input: s = "   -42"
Output: -42
Explanation:
Step 1: "__-42" (leading whitespace is read and ignored)
            ^

Step 2: "   -42" ('-' is read, so the result should be negative)
             ^

Step 3: "   -42" ("42" is read in)
               ^

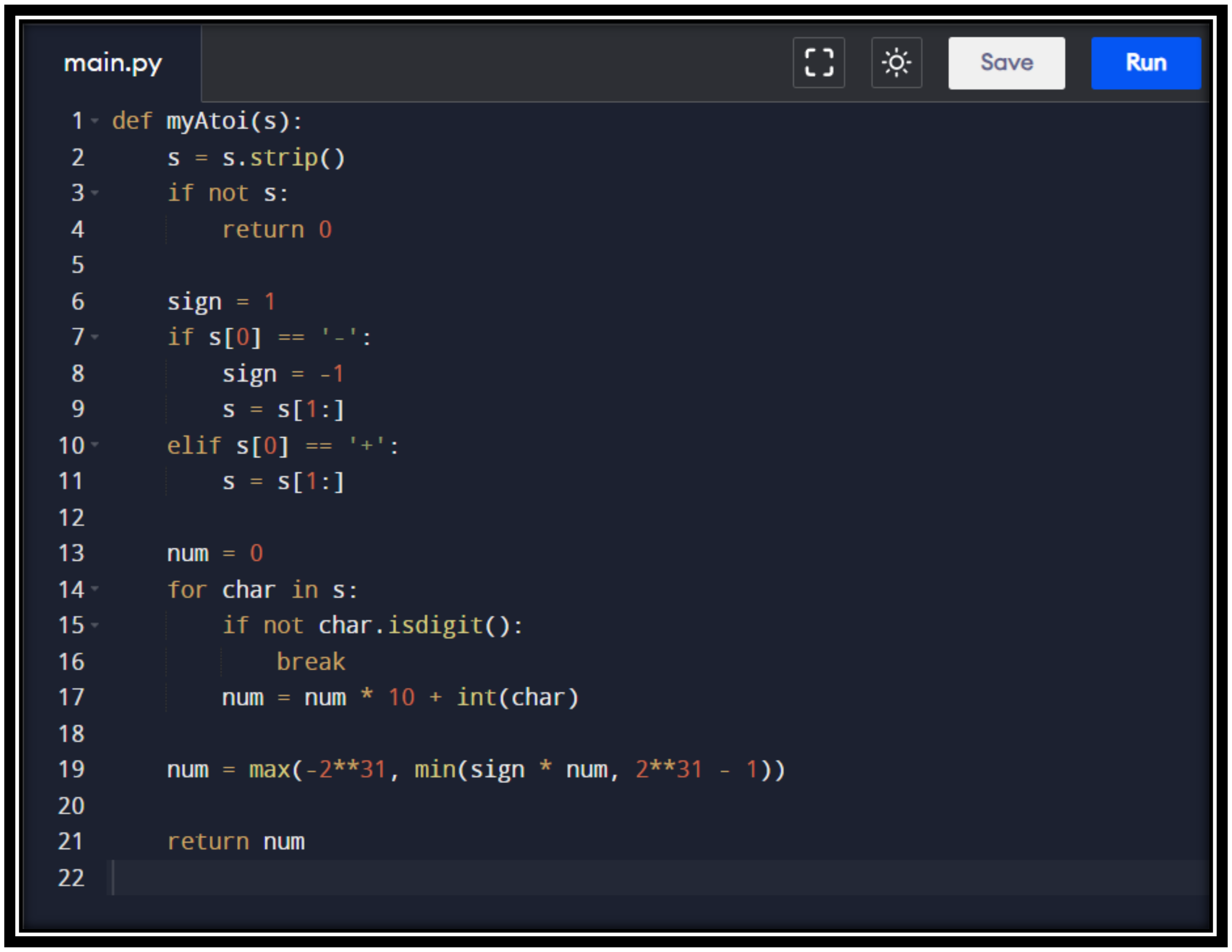
The parsed integer is -42.
Since -42 is in the range [-231, 231 - 1], the final result is -42.

Example 3:
Input: s = "4193 with words"
Output: 4193
Explanation:
Step 1: "4193 with words" (no characters read because there is no leading whitespace)
         ^

Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')
^

Step 3: "4193 with words" ("4193" is read in; reading stops because the next character is a non-digit)
^

The parsed integer is 4193.
Since 4193 is in the range [-231, 231 - 1], the final result is 4193.
Program:



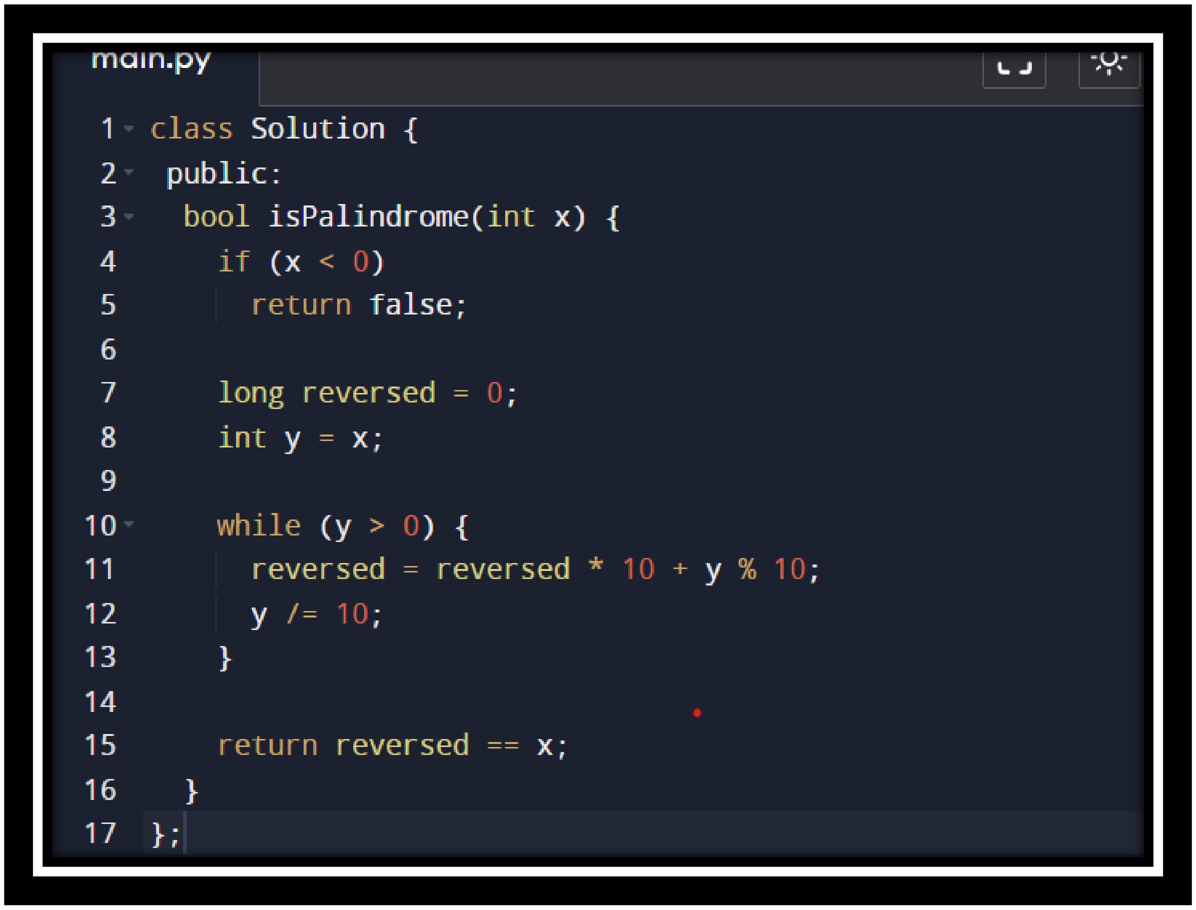
```python
def myAtoi(s):
    s = s.strip()
    if not s:
        return 0

    sign = 1
    if s[0] == '-':
        sign = -1
        s = s[1:]
    elif s[0] == '+':
        s = s[1:]

    num = 0
    for char in s:
        if not char.isdigit():
            break
        num = num * 10 + int(char)

    num = max(-2**31, min(sign * num, 2**31 - 1))

    return num
```

1. 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.

Example 3:
Input: x = 10
Output: false
Explanation: Reads 01 from right to left. Therefore it is not a palindrome.
Program:



main.py

```
class Solution {
 public:
  bool isPalindrome(int x) {
    if (x < 0)
      return false;

    long reversed = 0;
    int y = x;

    while (y > 0) {
      reversed = reversed * 10 + y % 10;
      y /= 10;
    }

    return reversed == x;
  }
};
```

9.

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.

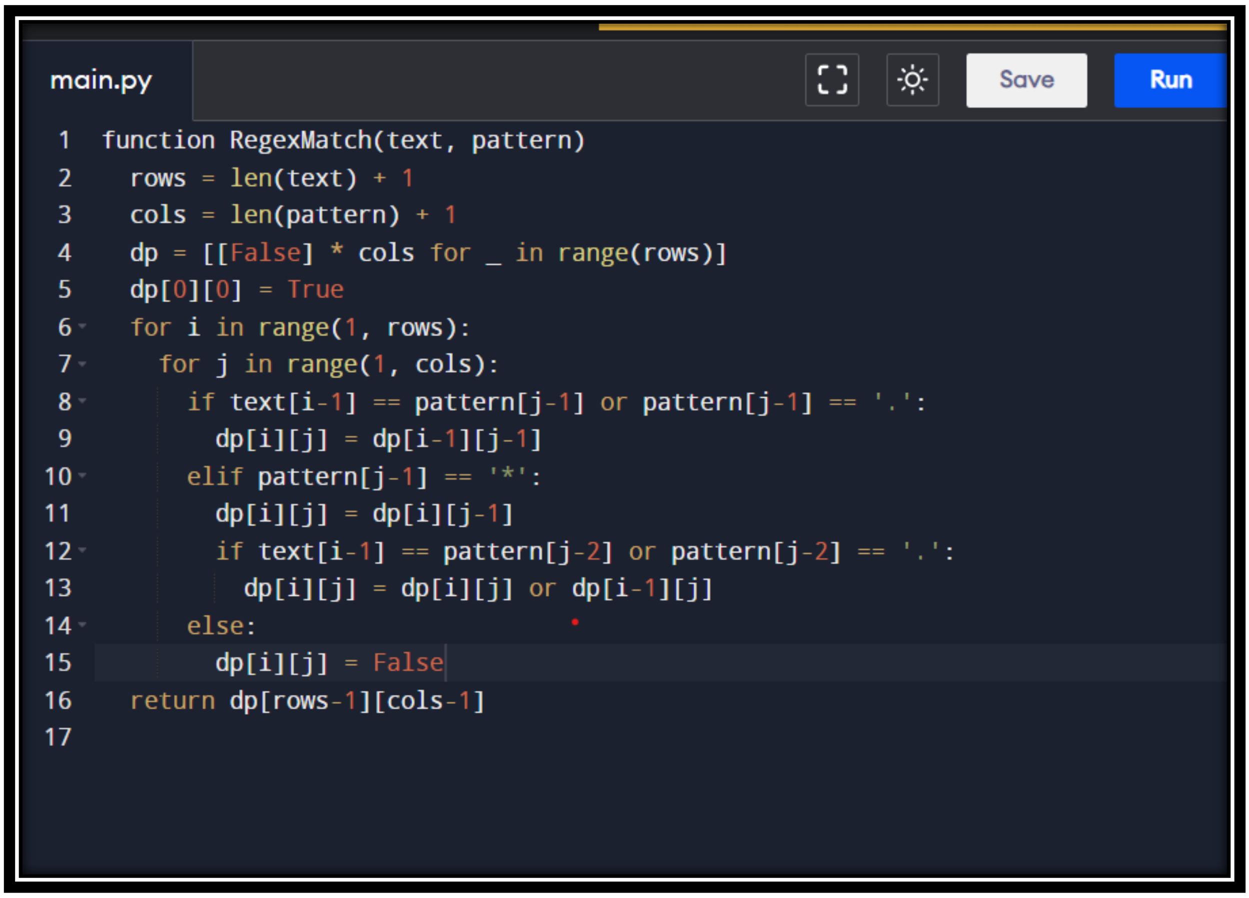
The matching should cover the entire input string (not partial).

Example 1:
Input: s = "aa", p = "a"

Output: false
Explanation: "a" does not match the entire string "aa".

Example 2:
Input: s = "aa", p = "a*"
Output: true
Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

Example 3:
Input: s = "ab", p = ".*"
Output: true
Explanation: ".*" means "zero or more (*) of any character (.)".
Program:



```
function RegexMatch(text, pattern)
  rows = len(text) + 1
  cols = len(pattern) + 1
  dp = [[False] * cols for _ in range(rows)]
  dp[0][0] = True
  for i in range(1, rows):
    for j in range(1, cols):
      if text[i-1] == pattern[j-1] or pattern[j-1] == '.':
        dp[i][j] = dp[i-1][j-1]
      elif pattern[j-1] == '*':
        dp[i][j] = dp[i][j-1]
        if text[i-1] == pattern[j-2] or pattern[j-2] == '.':
          dp[i][j] = dp[i][j] or dp[i-1][j]
      else:
        dp[i][j] = False
  return dp[rows-1][cols-1]
```