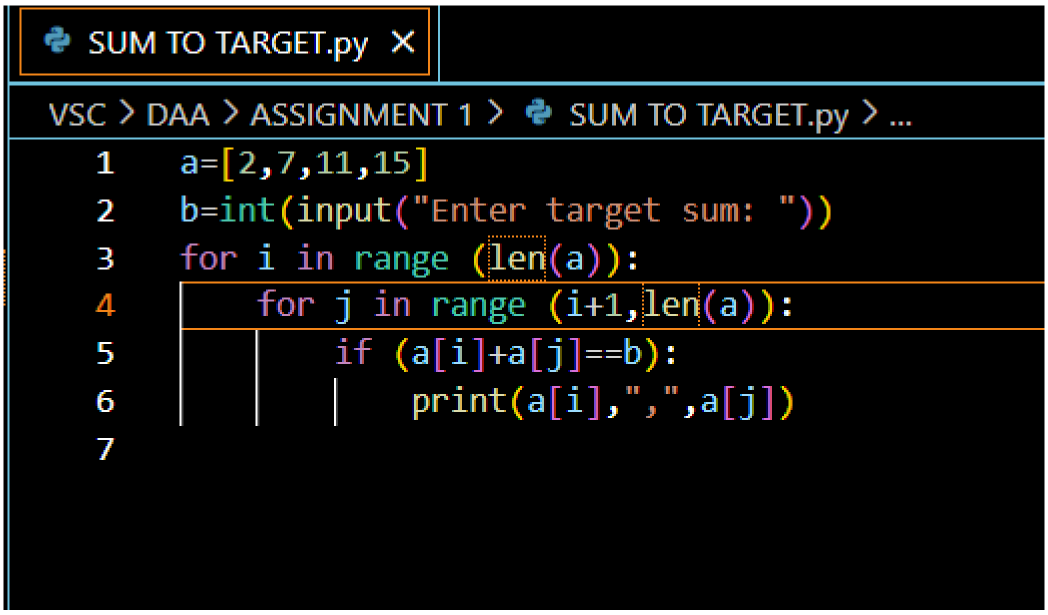
1.Two Sum Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order. Example 1:

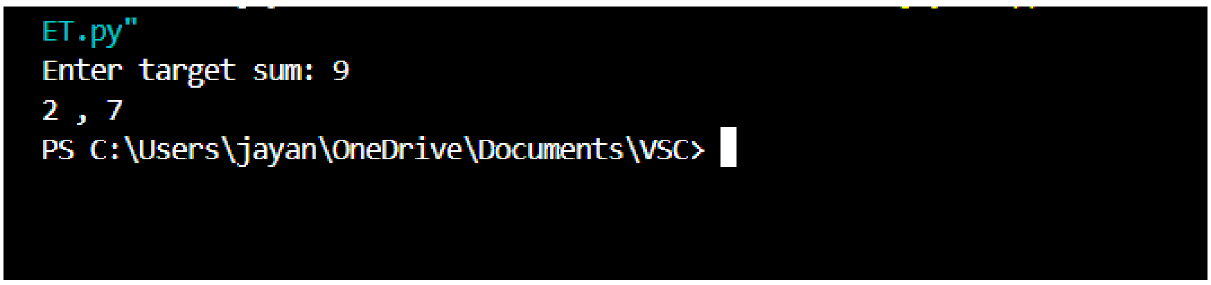
Input: nums = [2,7,11,15], target = 9

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]



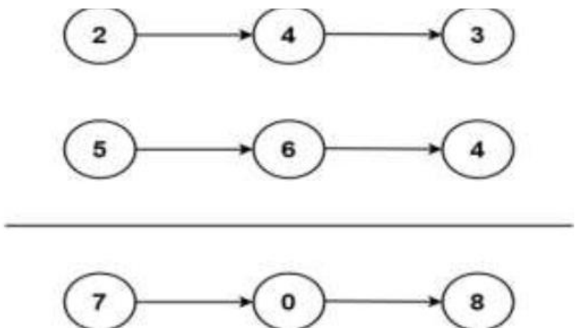
```python
a=[2,7,11,15]
b=int(input("Enter target sum: "))
for i in range (len(a)):
    for j in range (i+1,len(a)):
        if (a[i]+a[j]==b):
            print(a[i],",",a[j])

```



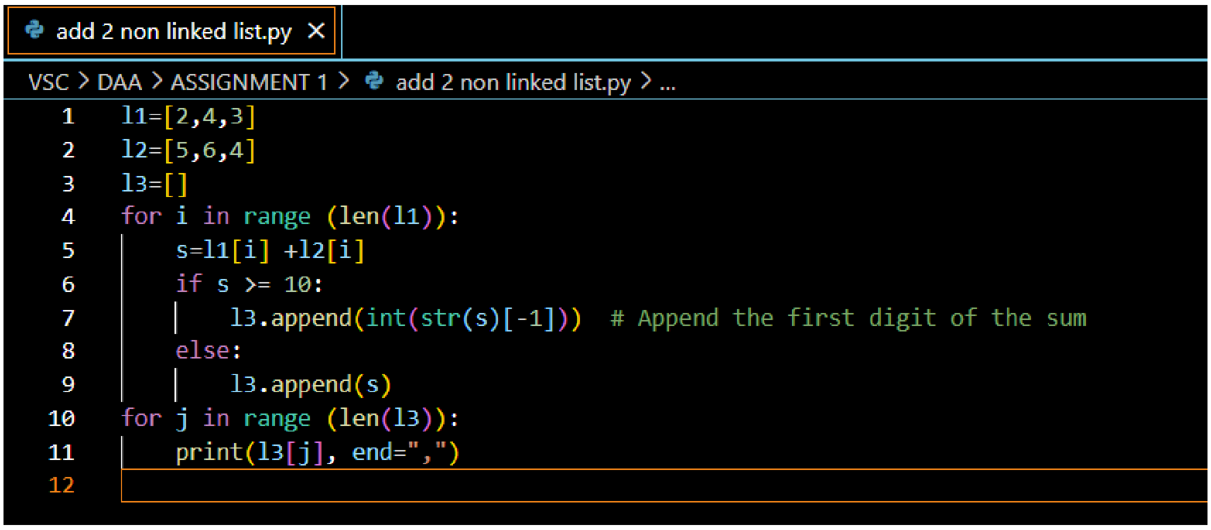
```
ET.py"
Enter target sum: 9
2 , 7
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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.

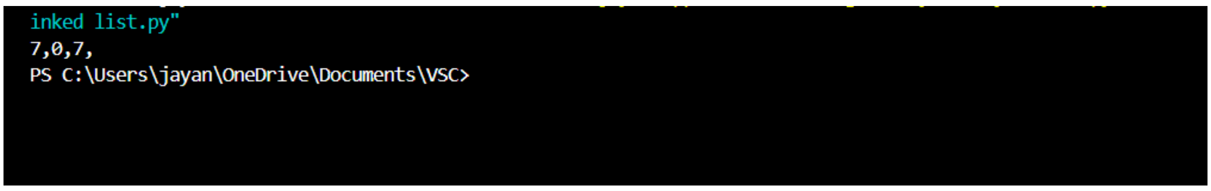
Add the two numbers and return the sum as a linked list. You may assume the two numbers do not contain any leading zero, except the number 0 itself.



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



```python
l1=[2,4,3]
l2=[5,6,4]
l3=[]
for i in range (len(l1)):
    s=l1[i] +l2[i]
    if s >= 10:
        l3.append(int(str(s)[-1]))  # Append the first digit of the sum
    else:
        l3.append(s)
for j in range (len(l3)):
    print(l3[j], end=",")

```

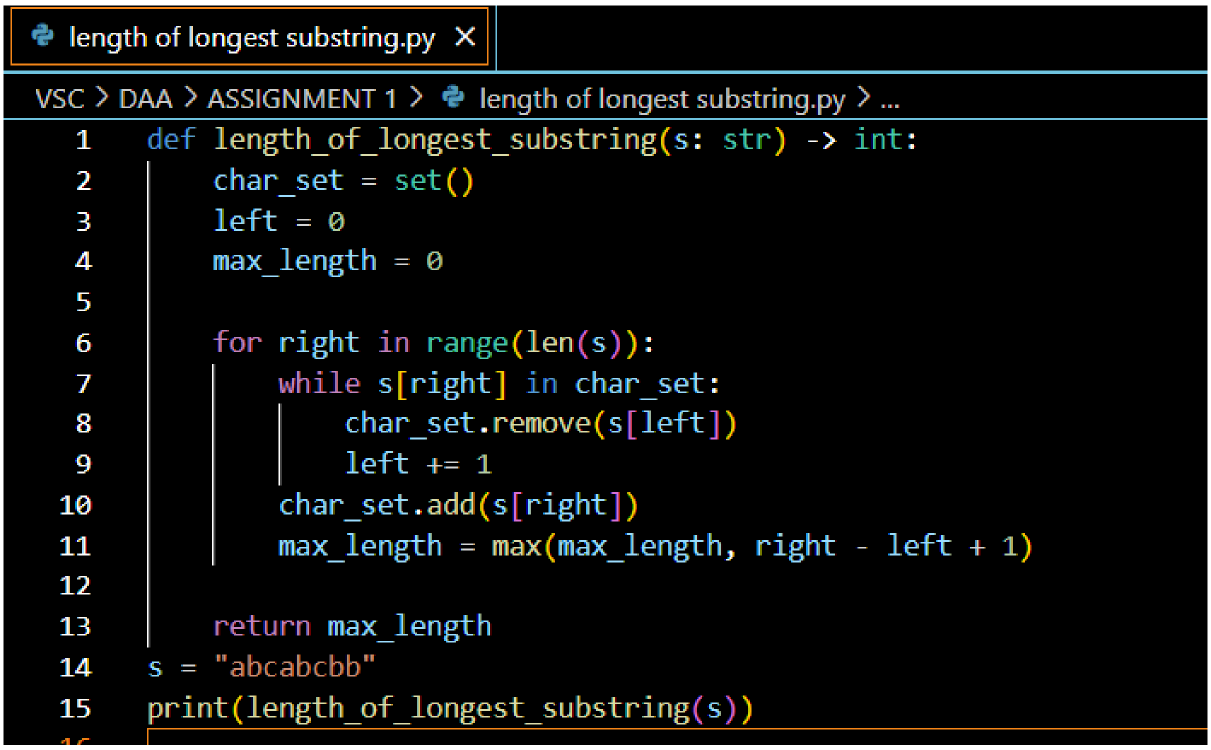


```
inked list.py"
7,0,7,
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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.

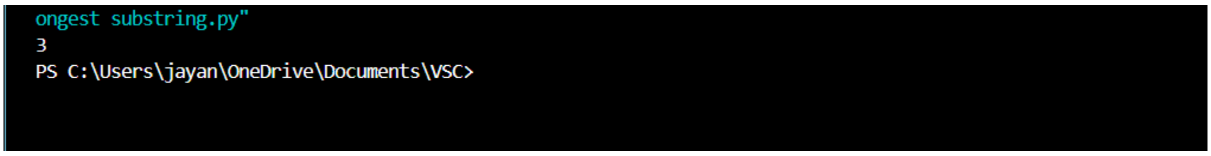
Example 2: Input: s = "bbbbb" Output: 1 Explanation: The answer is "b", with the length of 1



length of longest substring.py ×

VSC > DAA > ASSIGNMENT 1 > length of longest substring.py > ...

```python
def length_of_longest_substring(s: str) -> int:
    char_set = set()
    left = 0
    max_length = 0

    for right in range(len(s)):
        while s[right] in char_set:
            char_set.remove(s[left])
            left += 1
        char_set.add(s[right])
        max_length = max(max_length, right - left + 1)

    return max_length
s = "abcabcbb"
print(length_of_longest_substring(s))
```

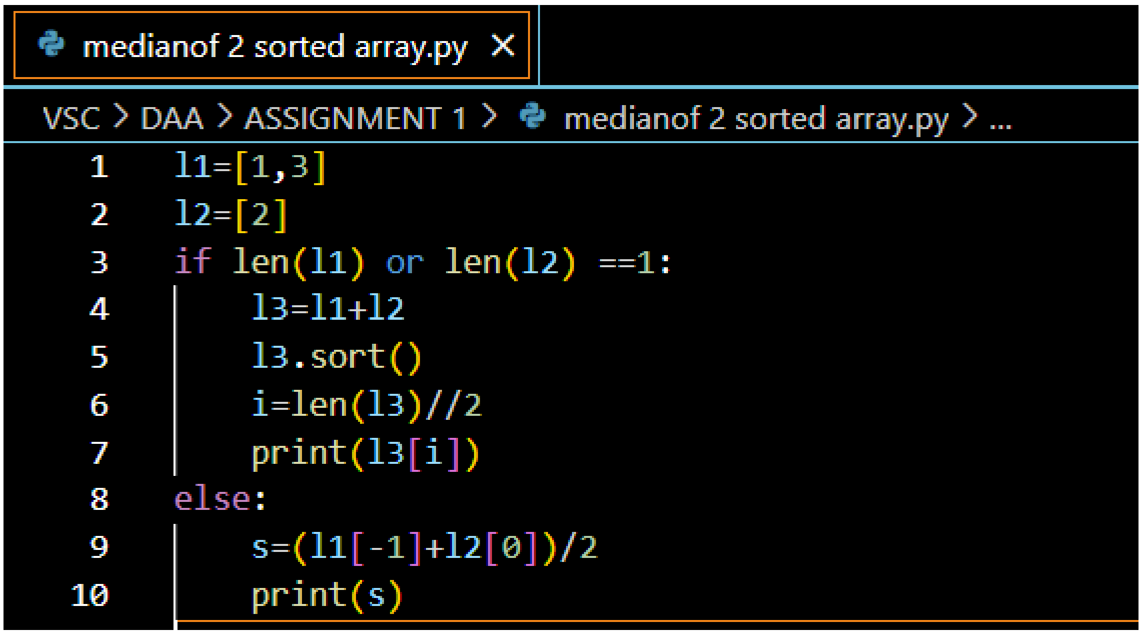


```
ongest substring.py"
3
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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.

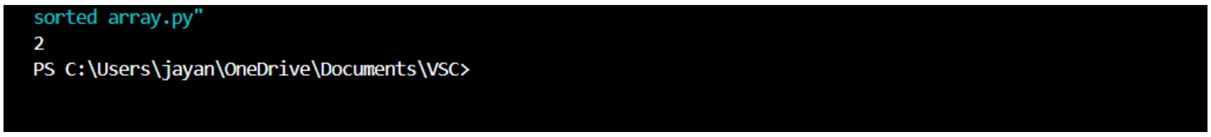
Example 2: Input: nums1 = [1,2], nums2 = [3,4] Output: 2.50000 Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5



medianof 2 sorted array.py ✕

VSC > DAA > ASSIGNMENT 1 > medianof 2 sorted array.py > ...

```python
l1=[1,3]
l2=[2]
if len(l1) or len(l2) ==1:
    l3=l1+l2
    l3.sort()
    i=len(l3)//2
    print(l3[i])
else:
    s=(l1[-1]+l2[0])/2
    print(s)
```

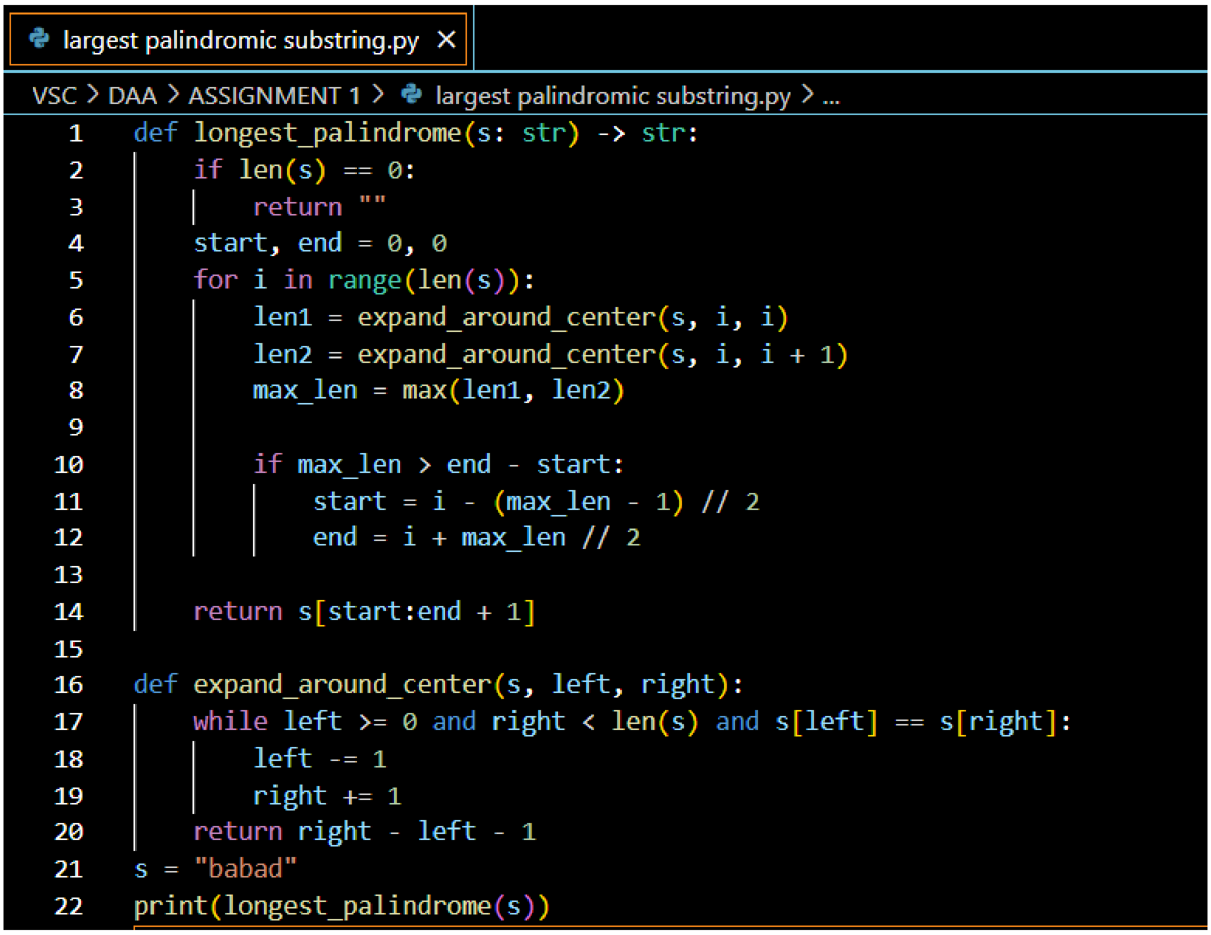


```
sorted array.py"
2
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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.

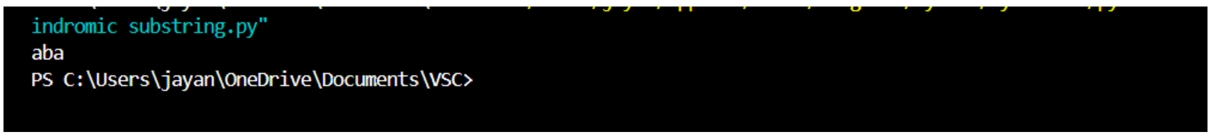
Example 2: Input: s = "cbbd" Output: "bb"



largest palindromic substring.py ×

VSC > DAA > ASSIGNMENT 1 > largest palindromic substring.py > ...

```python
def longest_palindrome(s: str) -> str:
    if len(s) == 0:
        return ""
    start, end = 0, 0
    for i in range(len(s)):
        len1 = expand_around_center(s, i, i)
        len2 = expand_around_center(s, i, i + 1)
        max_len = max(len1, len2)

        if max_len > end - start:
            start = i - (max_len - 1) // 2
            end = i + max_len // 2

    return s[start:end + 1]

def expand_around_center(s, left, right):
    while left >= 0 and right < len(s) and s[left] == s[right]:
        left -= 1
        right += 1
    return right - left - 1
s = "babad"
print(longest_palindrome(s))
```

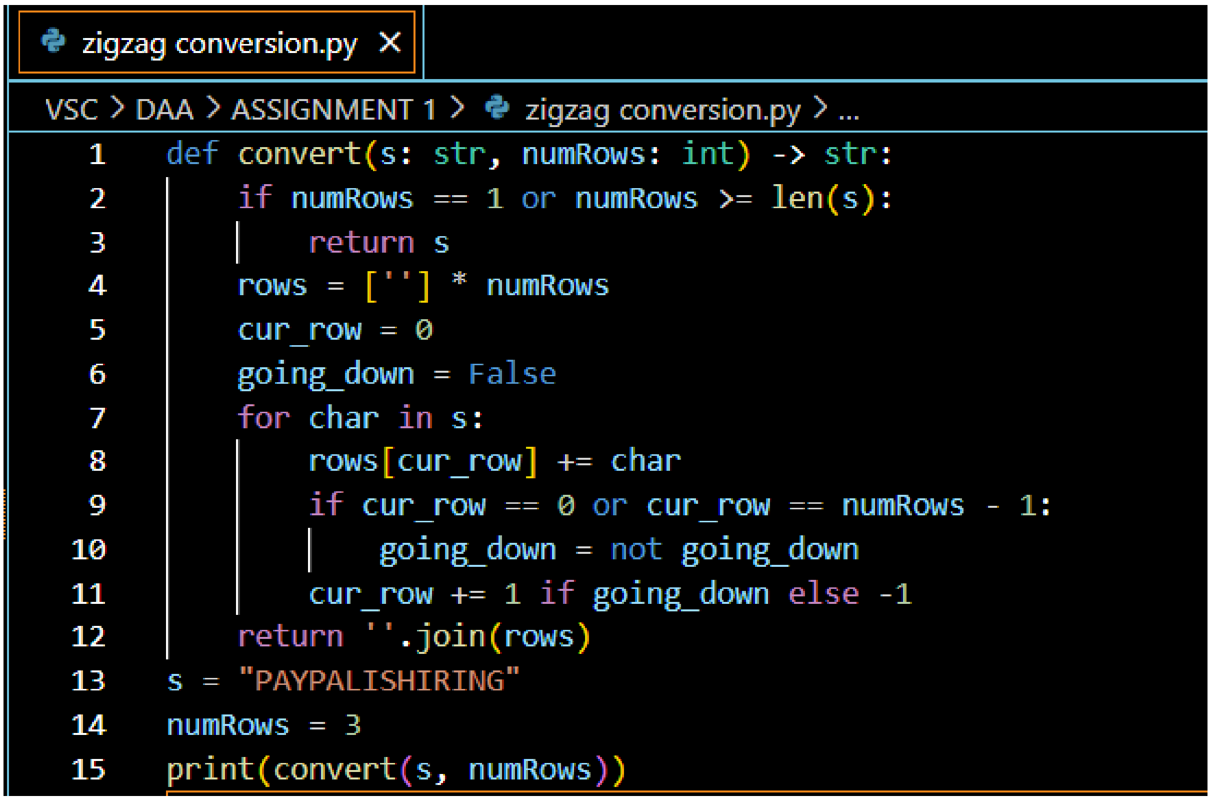


```
indromic substring.py"
aba
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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"

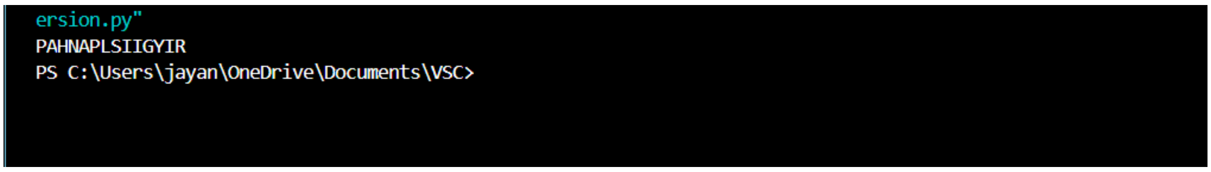
Example 2: Input: s = "PAYPALISHIRING", numRows = 4 Output: "PINALSIGYAHRPI"



zigzag conversion.py ×

VSC > DAA > ASSIGNMENT 1 > zigzag conversion.py > ...

```python
def convert(s: str, numRows: int) -> str:
    if numRows == 1 or numRows >= len(s):
        return s
    rows = [''] * numRows
    cur_row = 0
    going_down = False
    for char in s:
        rows[cur_row] += char
        if cur_row == 0 or cur_row == numRows - 1:
            going_down = not going_down
        cur_row += 1 if going_down else -1
    return ''.join(rows)
s = "PAYPALISHIRING"
numRows = 3
print(convert(s, numRows))
```

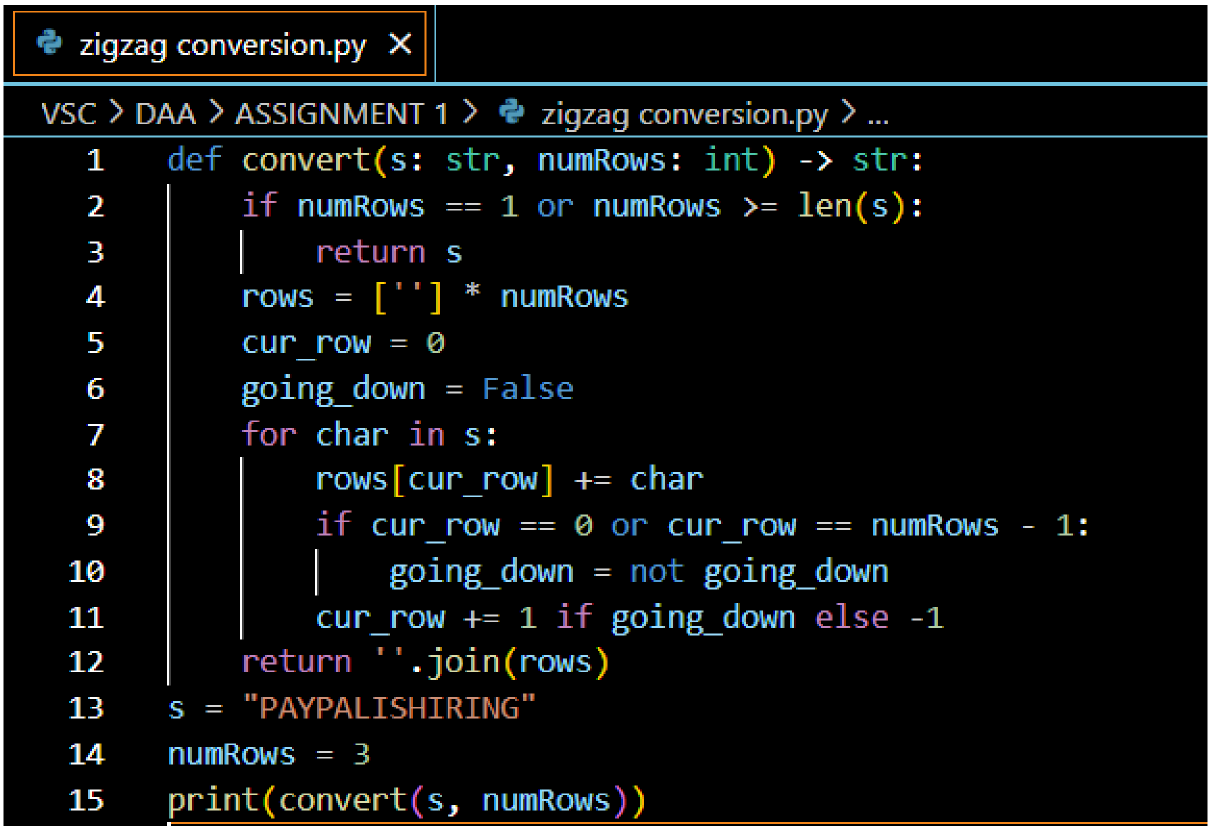


```
ersion.py"
PAHNAPLSIIGYIR
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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

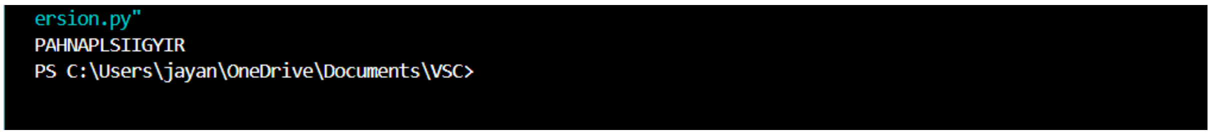
Example 2: Input: x =-123 Output:-321



zigzag conversion.py

VSC > DAA > ASSIGNMENT 1 > zigzag conversion.py > ...

```python
def convert(s: str, numRows: int) -> str:
    if numRows == 1 or numRows >= len(s):
        return s
    rows = [''] * numRows
    cur_row = 0
    going_down = False
    for char in s:
        rows[cur_row] += char
        if cur_row == 0 or cur_row == numRows - 1:
            going_down = not going_down
        cur_row += 1 if going_down else -1
    return ''.join(rows)
s = "PAYPALISHIRING"
numRows = 3
print(convert(s, numRows))
```



```
ersion.py"
PAHNAPLSIIGYIR
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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.

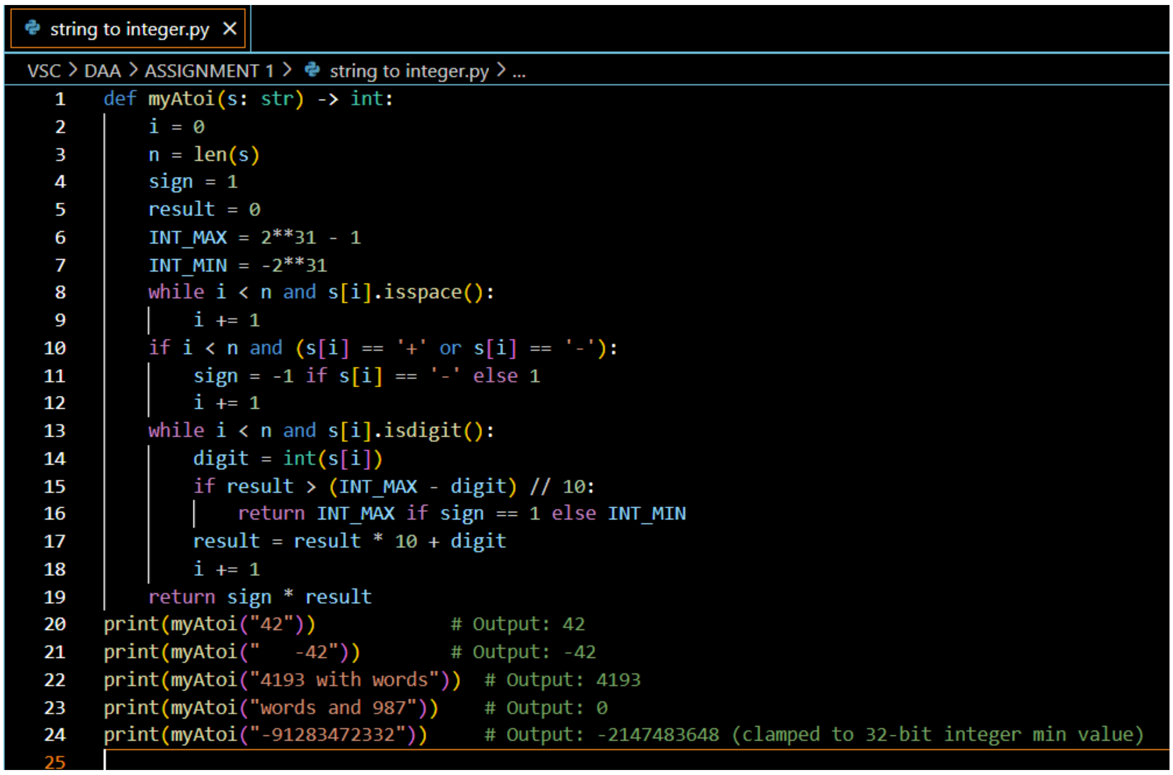
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.

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.

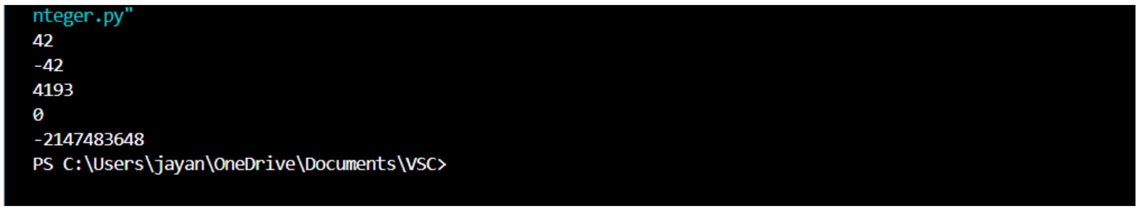
4. Convert these digits into an integer (i.e. "123"-> 123, "0032"-> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).

5. If the integer is out of the 32-bit signed integer range [-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.

6. Return the integer as the final result


string to integer.py
VSC > DAA > ASSIGNMENT 1 > string to integer.py > ...

```python
def myAtoi(s: str) -> int:
    i = 0
    n = len(s)
    sign = 1
    result = 0
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31
    while i < n and s[i].isspace():
        i += 1
    if i < n and (s[i] == '+' or s[i] == '-'):
        sign = -1 if s[i] == '-' else 1
        i += 1
    while i < n and s[i].isdigit():
        digit = int(s[i])
        if result > (INT_MAX - digit) // 10:
            return INT_MAX if sign == 1 else INT_MIN
        result = result * 10 + digit
        i += 1
    return sign * result
print(myAtoi("42"))            # Output: 42
print(myAtoi("   -42"))        # Output: -42
print(myAtoi("4193 with words"))  # Output: 4193
print(myAtoi("words and 987"))    # Output: 0
print(myAtoi("-91283472332"))     # Output: -2147483648 (clamped to 32-bit integer min value)
```

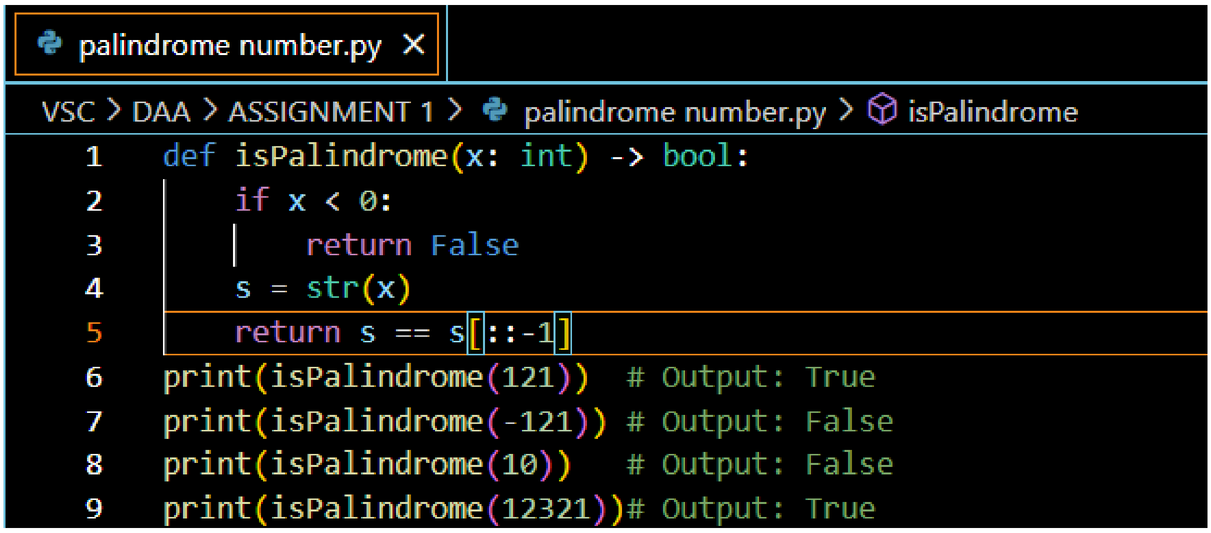


```
nteger.py"
42
-42
4193
0
-2147483648
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

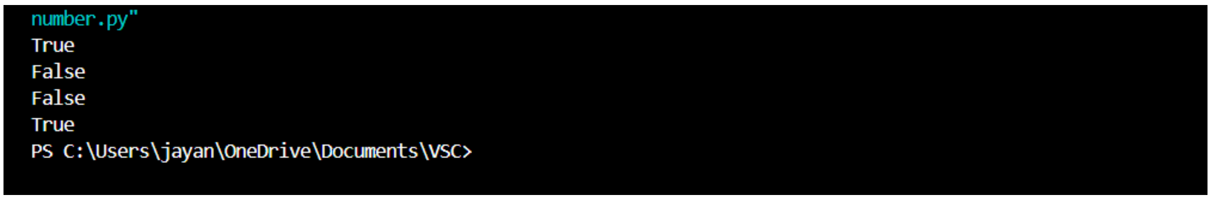
9. Palindrome Number Given an integer x, return true if x is a palindrome, and false otherwise.

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.



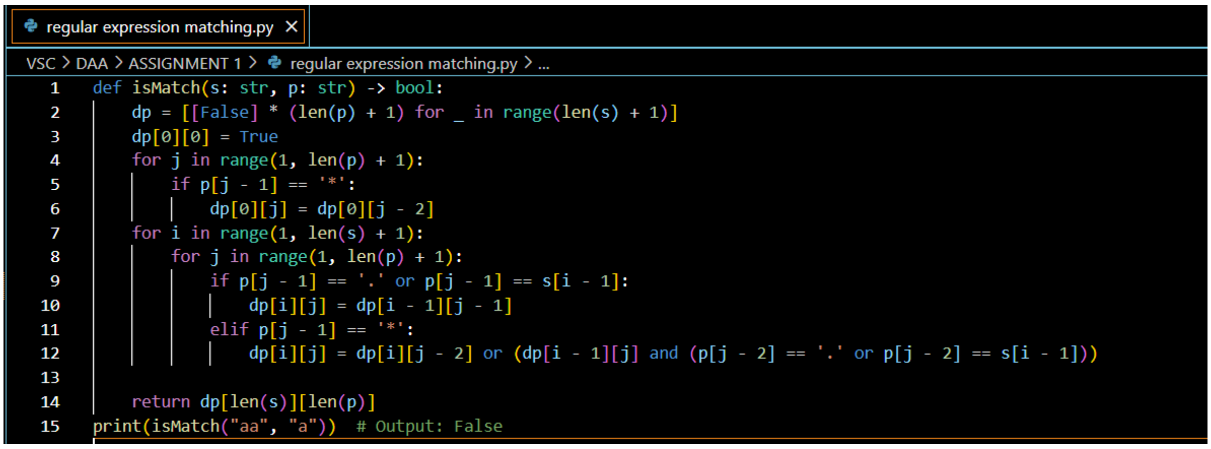
```
palindrome number.py ×
VSC > DAA > ASSIGNMENT 1 > palindrome number.py > isPalindrome
def isPalindrome(x: int) -> bool:
    if x < 0:
        return False
    s = str(x)
    return s == s[::-1]
print(isPalindrome(121))  # Output: True
print(isPalindrome(-121)) # Output: False
print(isPalindrome(10))   # Output: False
print(isPalindrome(12321))# Output: True
```



```
number.py"
True
False
False
True
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

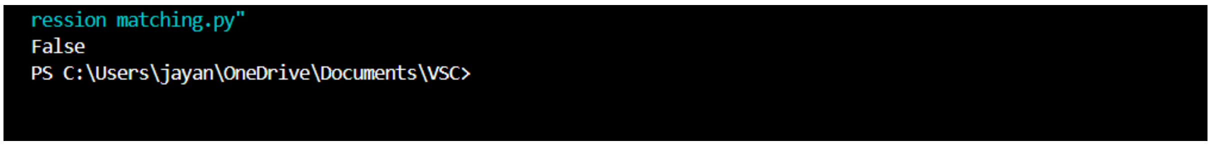
10. Regular Expression Matching Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where: ● '.' Matches any single character. ● '*' Matches zero or more of the preceding element. The matching should cover the entire input string (not partial).

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


regular expression matching.py ✕

VSC > DAA > ASSIGNMENT 1 > regular expression matching.py > ...

```python
def isMatch(s: str, p: str) -> bool:
    dp = [[False] * (len(p) + 1) for _ in range(len(s) + 1)]
    dp[0][0] = True
    for j in range(1, len(p) + 1):
        if p[j - 1] == '*':
            dp[0][j] = dp[0][j - 2]
    for i in range(1, len(s) + 1):
        for j in range(1, len(p) + 1):
            if p[j - 1] == '.' or p[j - 1] == s[i - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            elif p[j - 1] == '*':
                dp[i][j] = dp[i][j - 2] or (dp[i - 1][j] and (p[j - 2] == '.' or p[j - 2] == s[i - 1]))

    return dp[len(s)][len(p)]
print(isMatch("aa", "a"))  # Output: False
```



```
ression matching.py"
False
PS C:\Users\jayan\OneDrive\Documents\VSC>
```