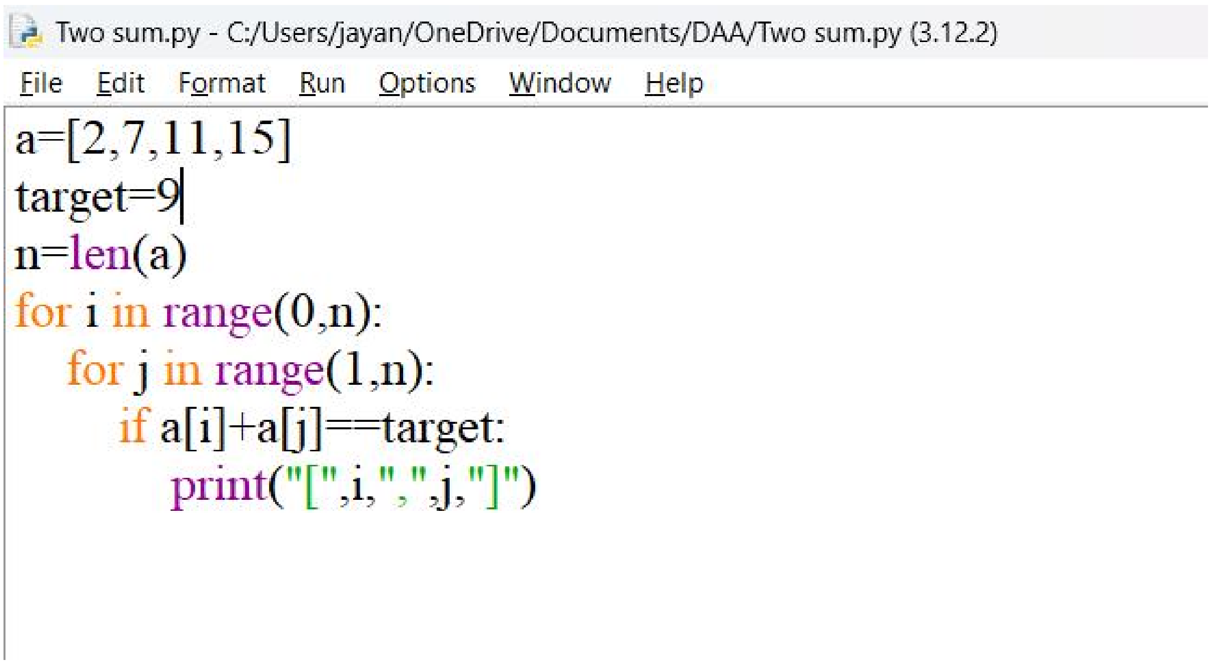
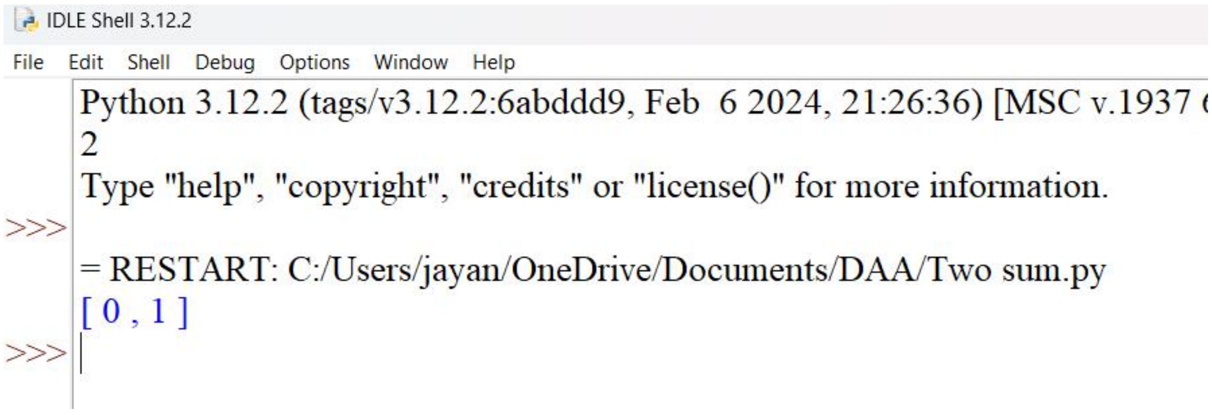
ASSIGNMENT-5

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].

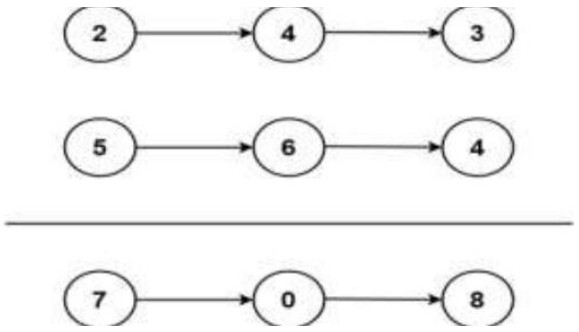

Two sum.py - C:/Users/jayan/OneDrive/Documents/DAA/Two sum.py (3.12.2)
File Edit Format Run Options Window Help

```
a=[2,7,11,15]
target=9
n=len(a)
for i in range(0,n):
    for j in range(1,n):
        if a[i]+a[j]==target:
            print("[",i,",",j,"]")
```

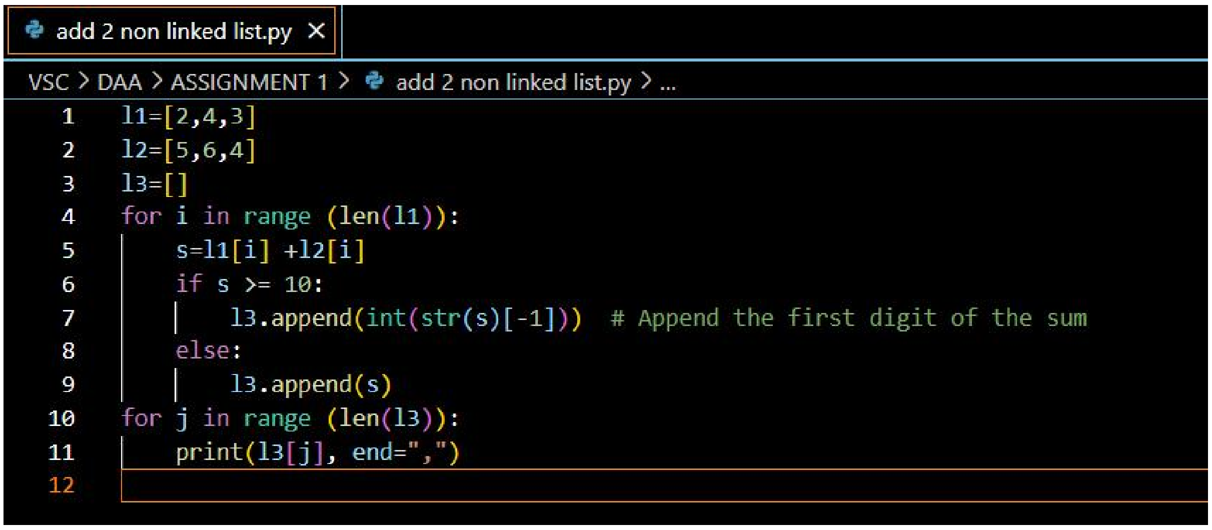


```
    Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 (
    2
    Type "help", "copyright", "credits" or "license()" for more information.
>>> 
    = RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Two sum.py
    [ 0 , 1 ]
>>> 
```

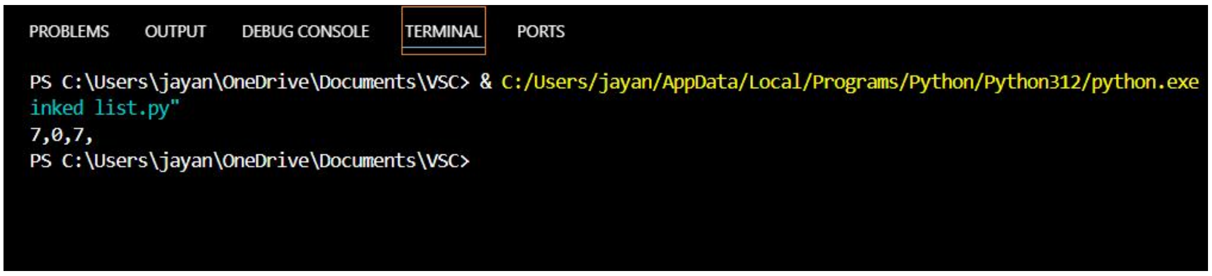
2. Add Two Numbers You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list. You may assume the two numbers do not contain any leading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

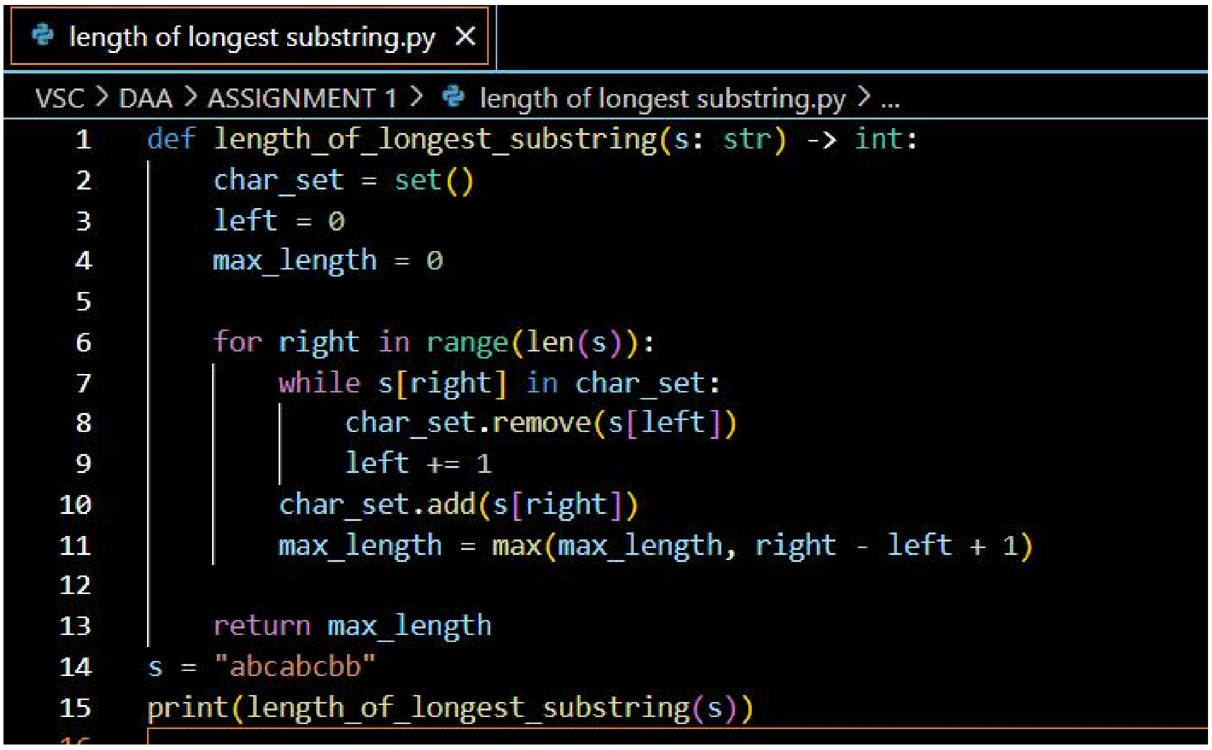


```
PS C:\Users\jayan\OneDrive\Documents\VSC> & C:/Users/jayan/AppData/Local/Programs/Python/Python312/python.exe
inked list.py"
7,0,7,
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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.

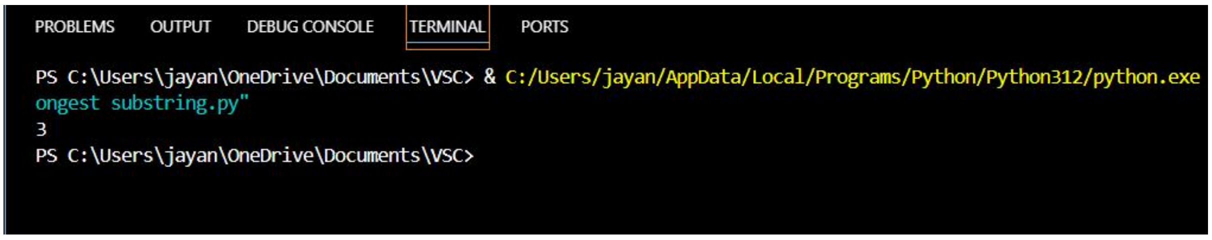
Example 2: Input: s = "bbbbb" Output: 1 Explanation: The answer is "b", with the length of 1



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))
```

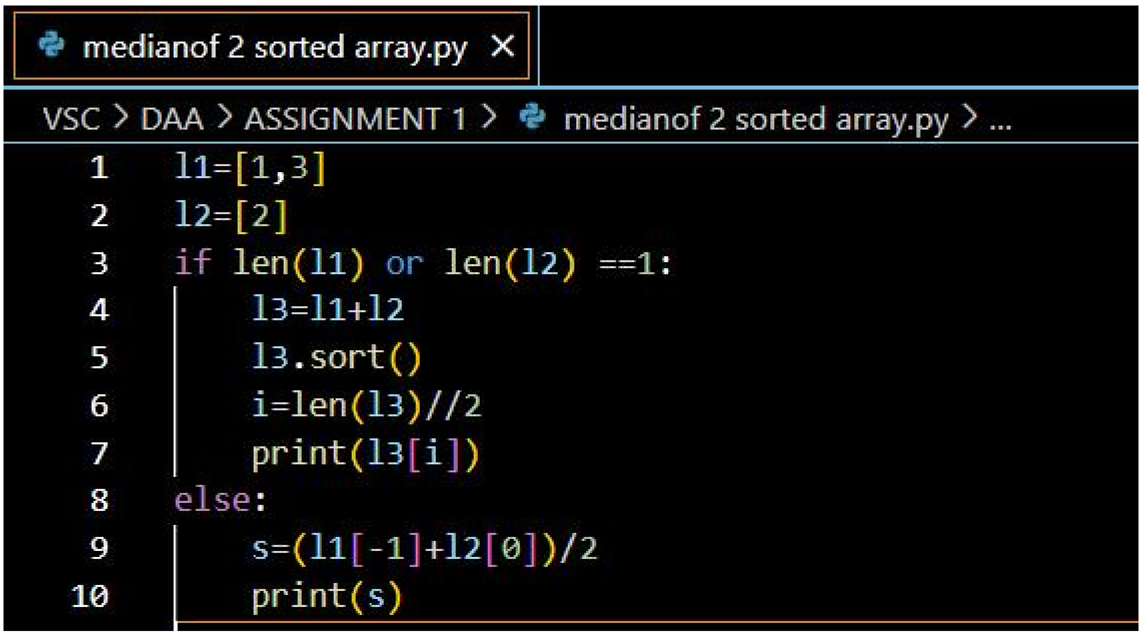


```
PS C:\Users\jayan\OneDrive\Documents\VSC> & C:/Users/jayan/AppData/Local/Programs/Python/Python312/python.exe
ongest substring.py"
3
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

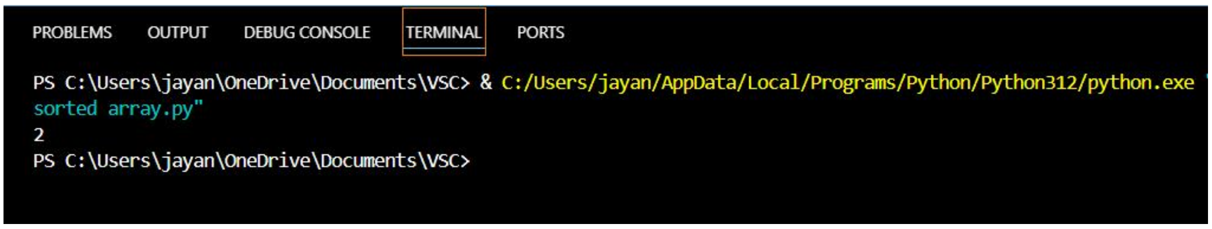
4. Median of Two Sorted Arrays Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

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)
```

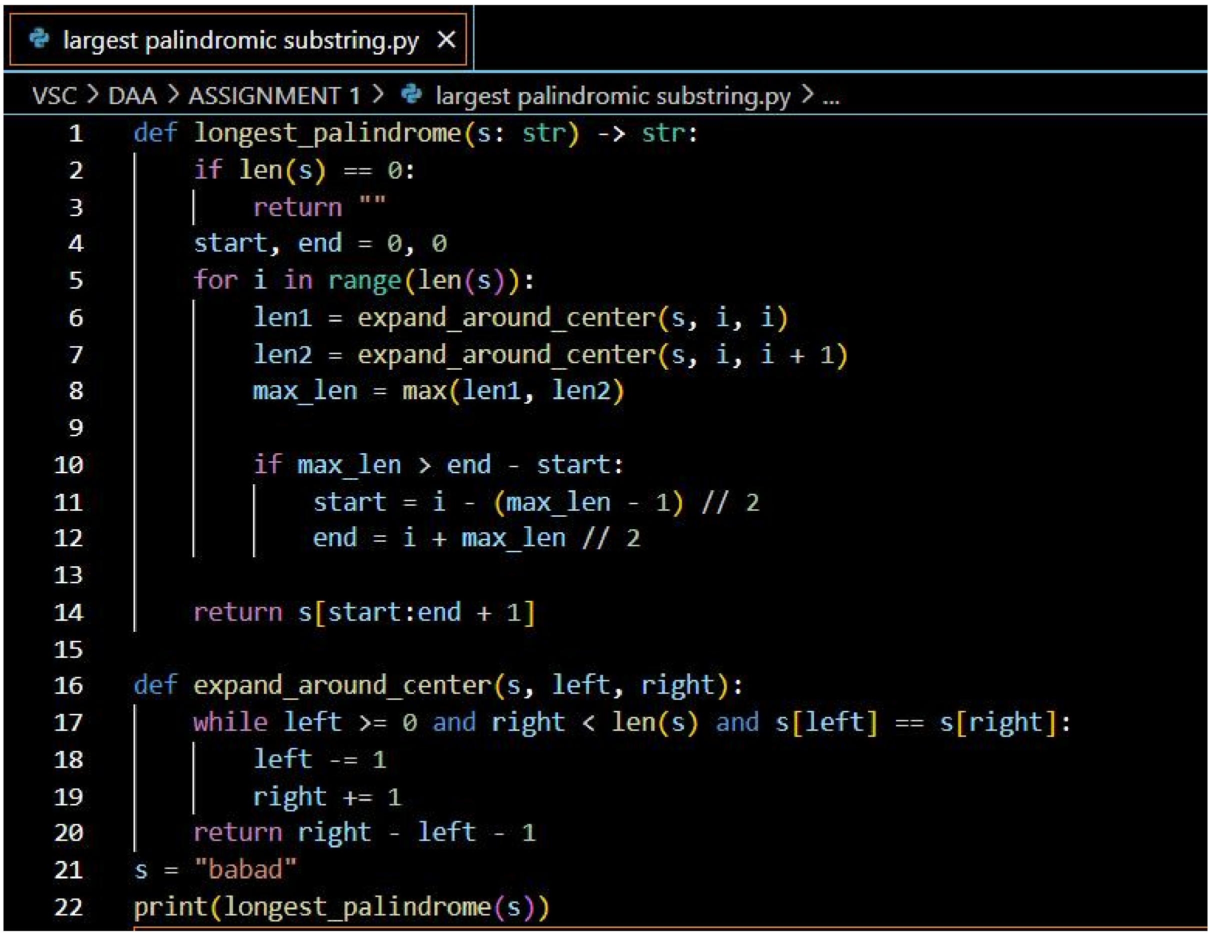


```
PS C:\Users\jayan\OneDrive\Documents\VSC> & C:/Users/jayan/AppData/Local/Programs/Python/Python312/python.exe
sorted array.py"
2
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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.

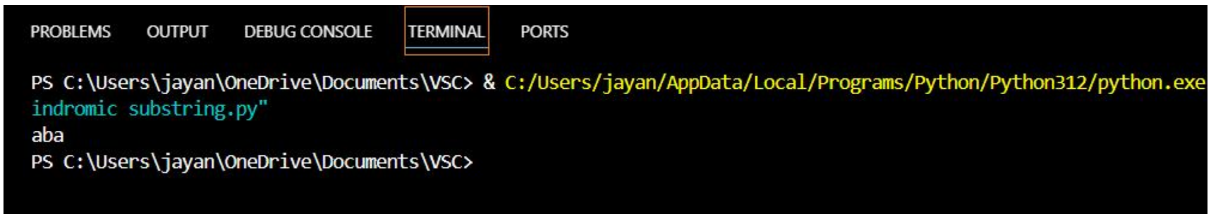
Example 2: Input: s = "cbbd" Output: "bb"



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))
```

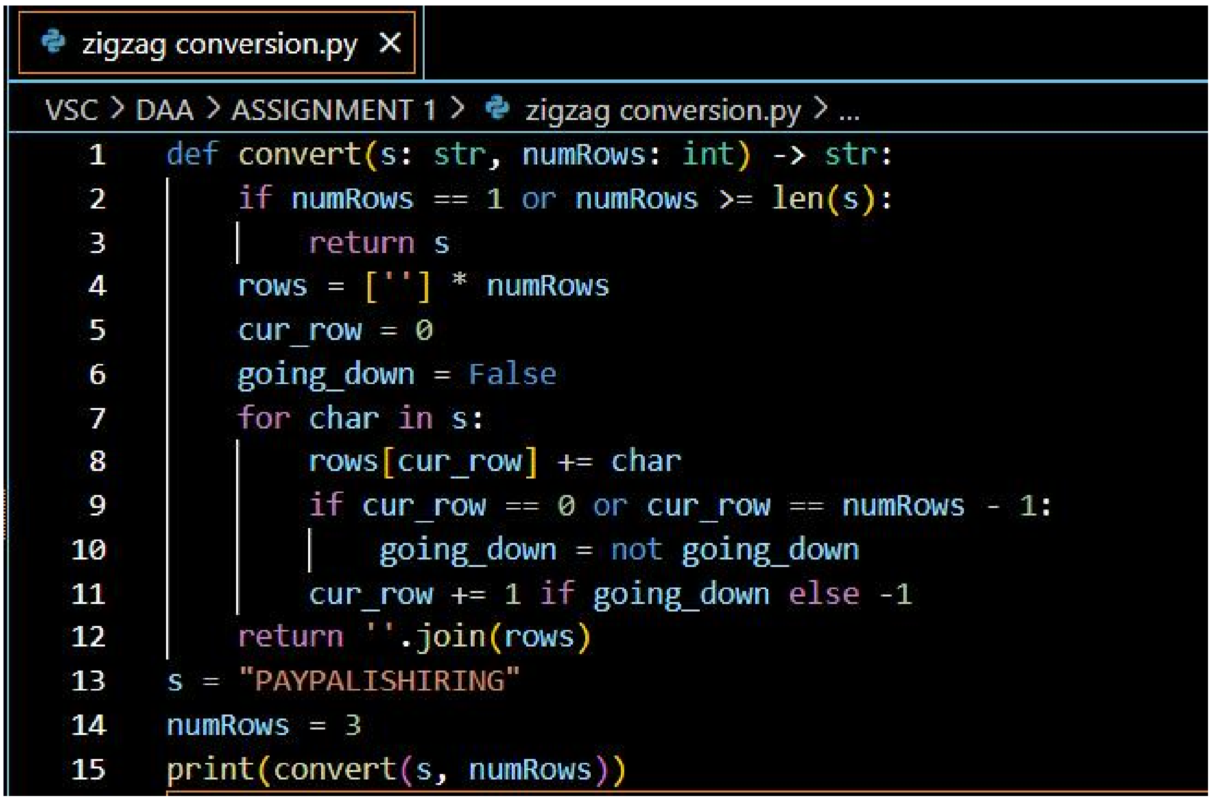


```
PS C:\Users\jayan\OneDrive\Documents\VSC> & C:/Users/jayan/AppData/Local/Programs/Python/Python312/python.exe
indromic substring.py"
aba
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

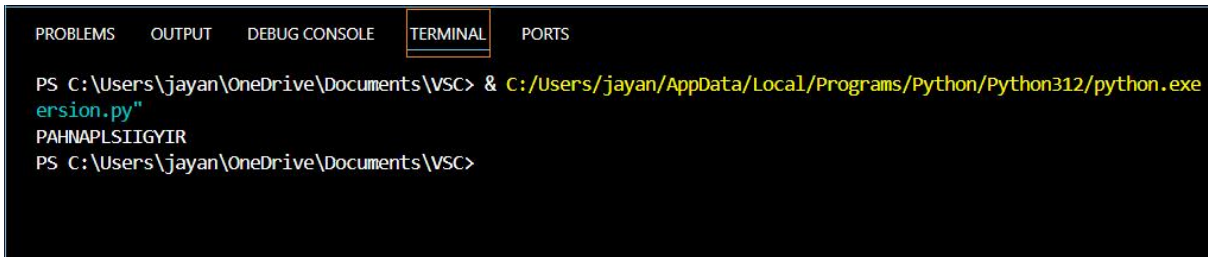
6. Zigzag Conversion The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility) P A H N A P L S I I G Y I R And then read line by line: "PAHNAPLSIIGYIR" Write the code that will take a string 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"



```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))
```

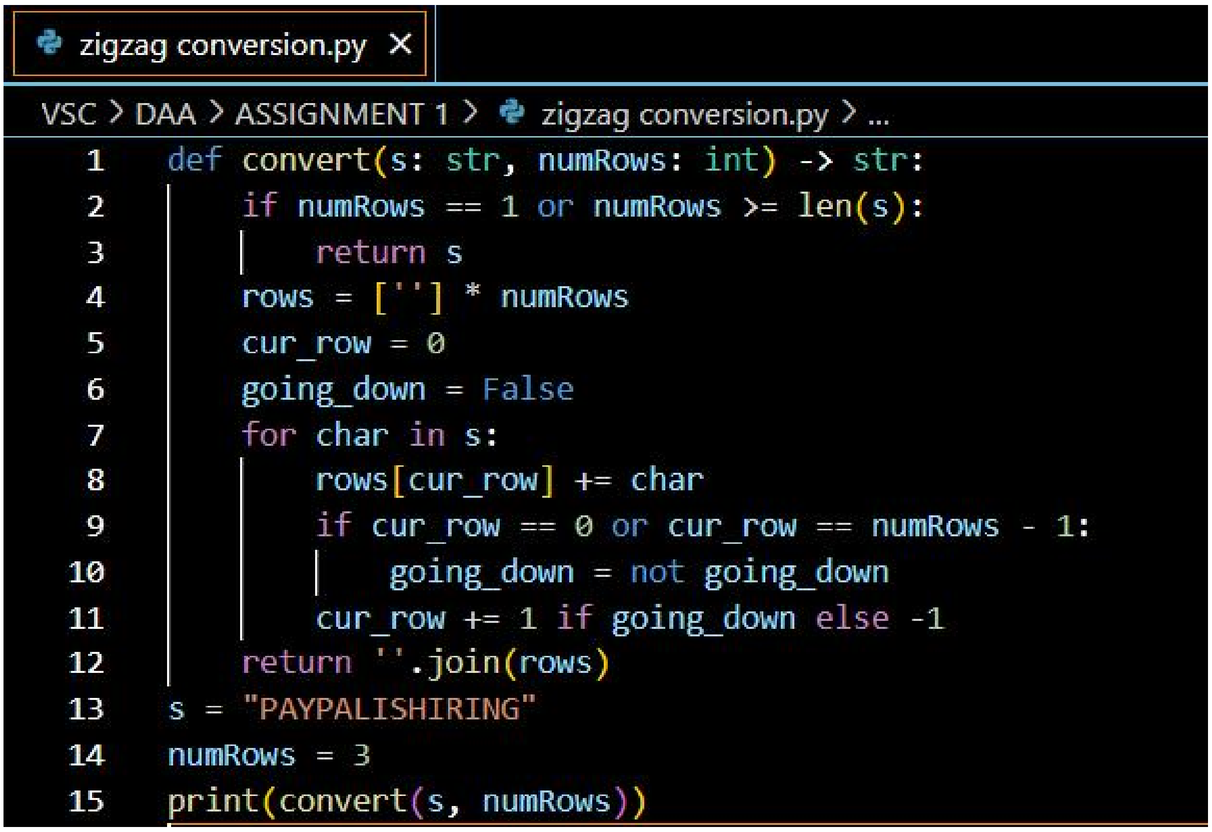


```
PS C:\Users\jayan\OneDrive\Documents\VSC> & C:/Users/jayan/AppData/Local/Programs/Python/Python312/python.exe
ersion.py"
PAHNAPLSIIGYIR
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

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

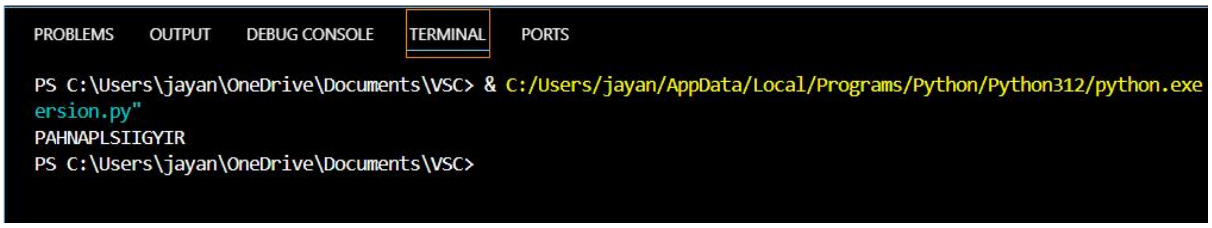
Example 2: Input: x = -123 Output: -321



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))
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\jayan\OneDrive\Documents\VSC> & C:/Users/jayan/AppData/Local/Programs/Python/Python312/python.exe
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.

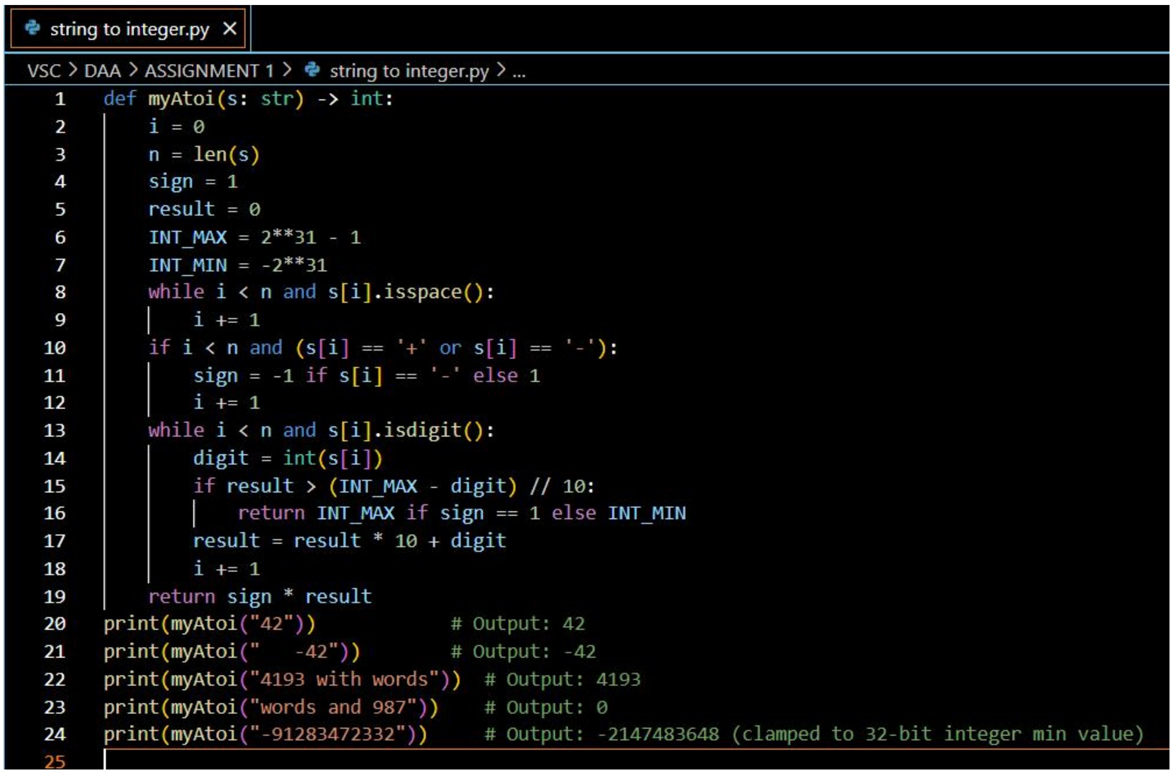
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.

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.

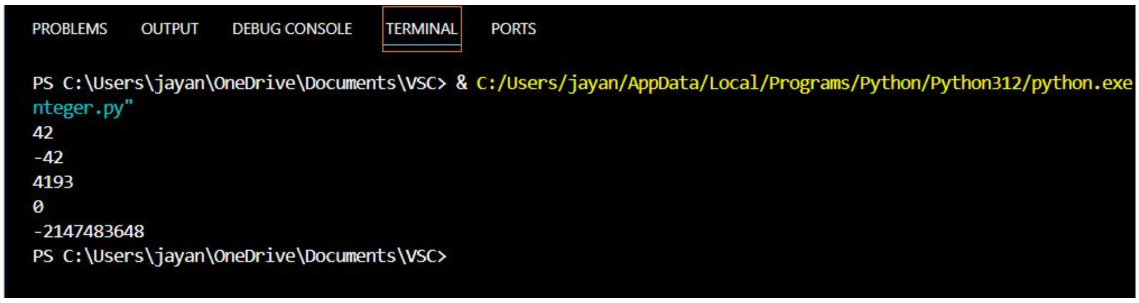
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).

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



```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)
```

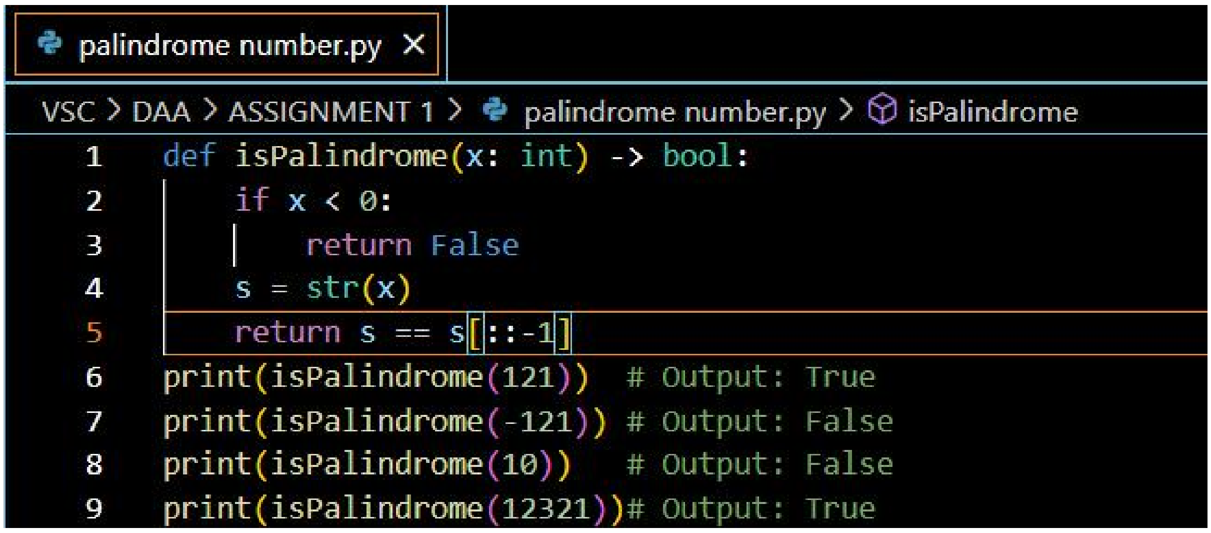


```
PS C:\Users\jayan\OneDrive\Documents\VSC> & C:/Users/jayan/AppData/Local/Programs/Python/Python312/python.exe
nteger.py"
42
-42
4193
0
-2147483648
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

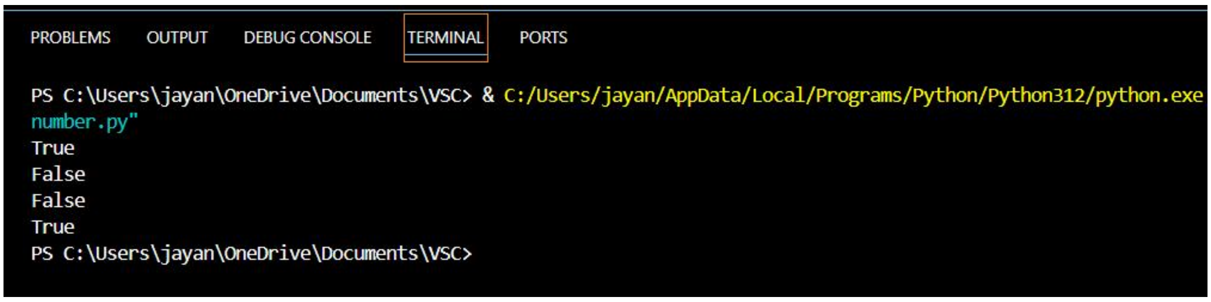
9. Palindrome Number Given an integer x, return true if x is a palindrome, and false otherwise.

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

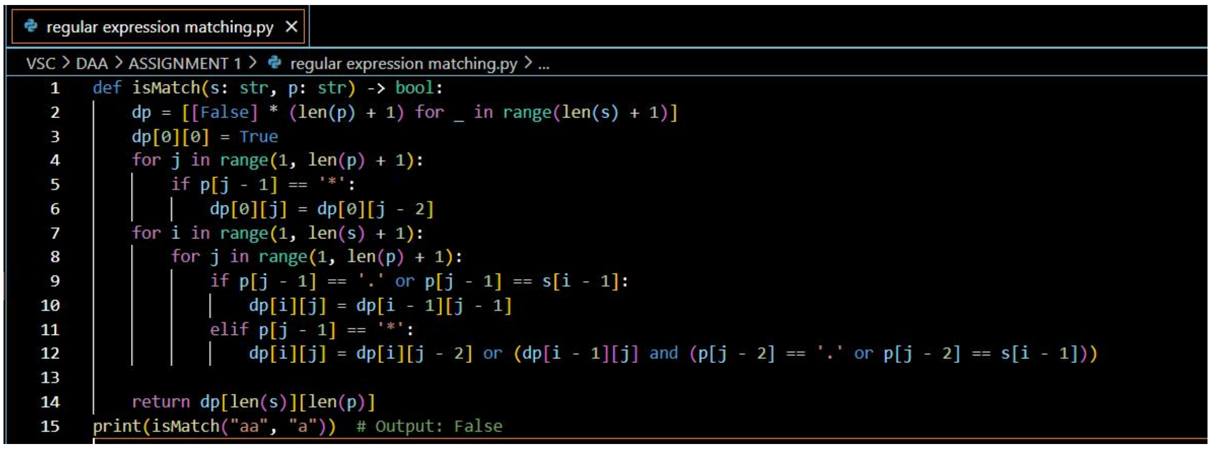
```python
def isPalindrome(x: int) -> bool:
    if x < 0:
        return False
    s = str(x)
    return s == s[::-1]
print(isPalindrome(121))  # Output: True
print(isPalindrome(-121)) # Output: False
print(isPalindrome(10))   # Output: False
print(isPalindrome(12321))# Output: True
```



```
PS C:\Users\jayan\OneDrive\Documents\VSC> & C:/Users/jayan/AppData/Local/Programs/Python/Python312/python.exe
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).

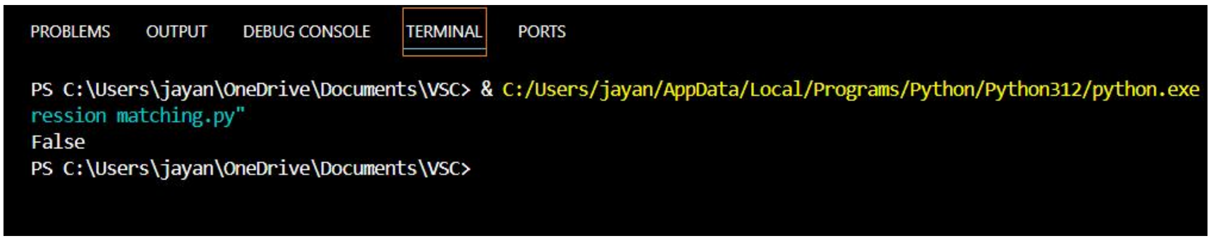
Example 1: Input: s = "aa", p = "a" Output: false Explanation: "a" does not match the entire string "aa".



regular expression matching.py

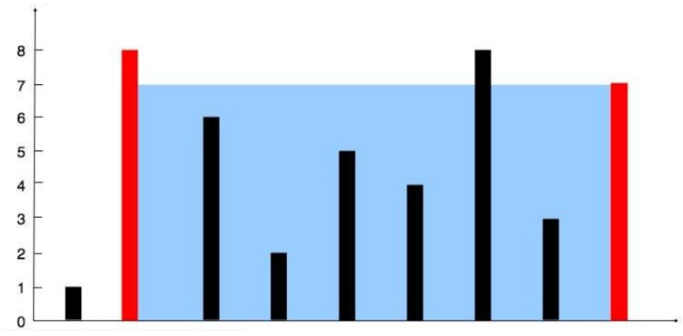
VSC > DAA > ASSIGNMENT 1 > regular expression matching.py > ...

```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
```



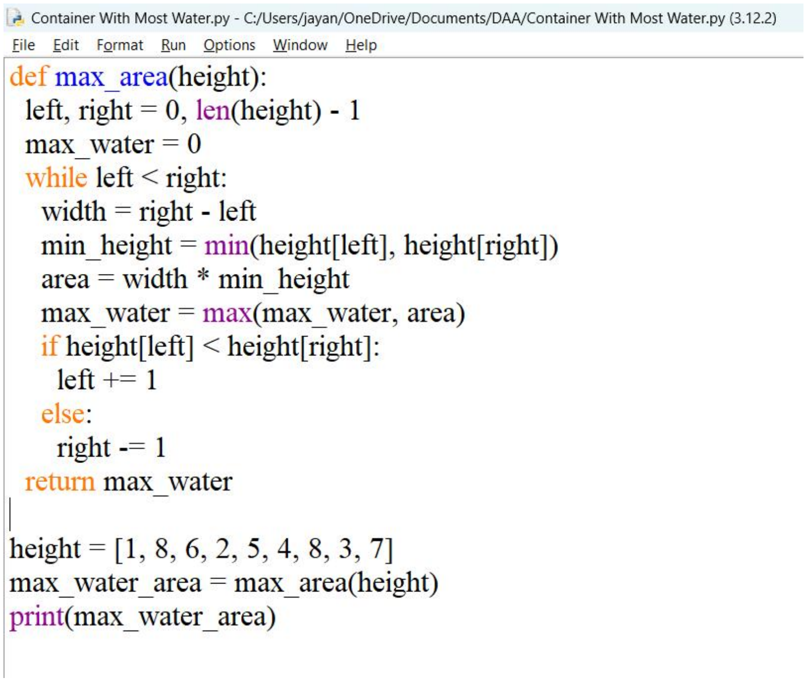
```
PS C:\Users\jayan\OneDrive\Documents\VSC> & C:/Users/jayan/AppData/Local/Programs/Python/Python312/python.exe
ression matching.py"
False
PS C:\Users\jayan\OneDrive\Documents\VSC>
```

11 .Container With Most Water You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]). Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store. Notice that you may not slant the container.

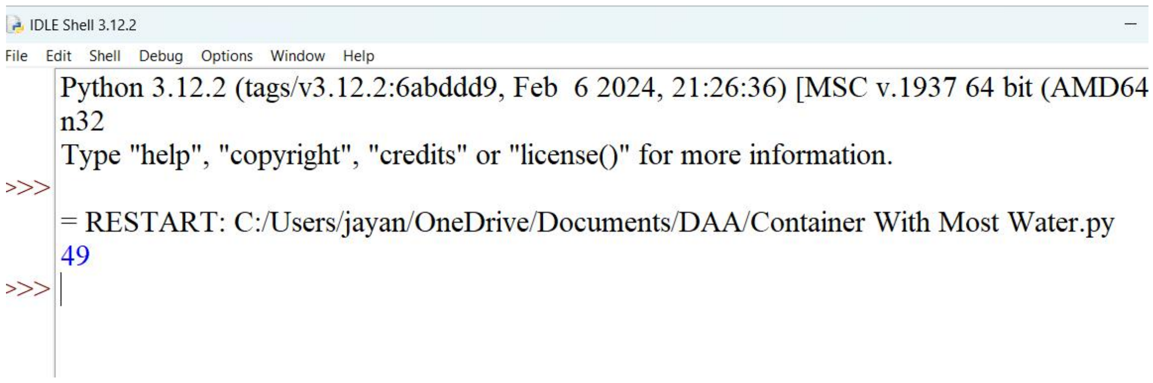


Example 2:

Input: height = [1,8,6,2,5,4,8,3,7] Output: 49 Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.


Container With Most Water.py - C:/Users/jayan/OneDrive/Documents/DAA/Container With Most Water.py (3.12.2)
File Edit Format Run Options Window Help

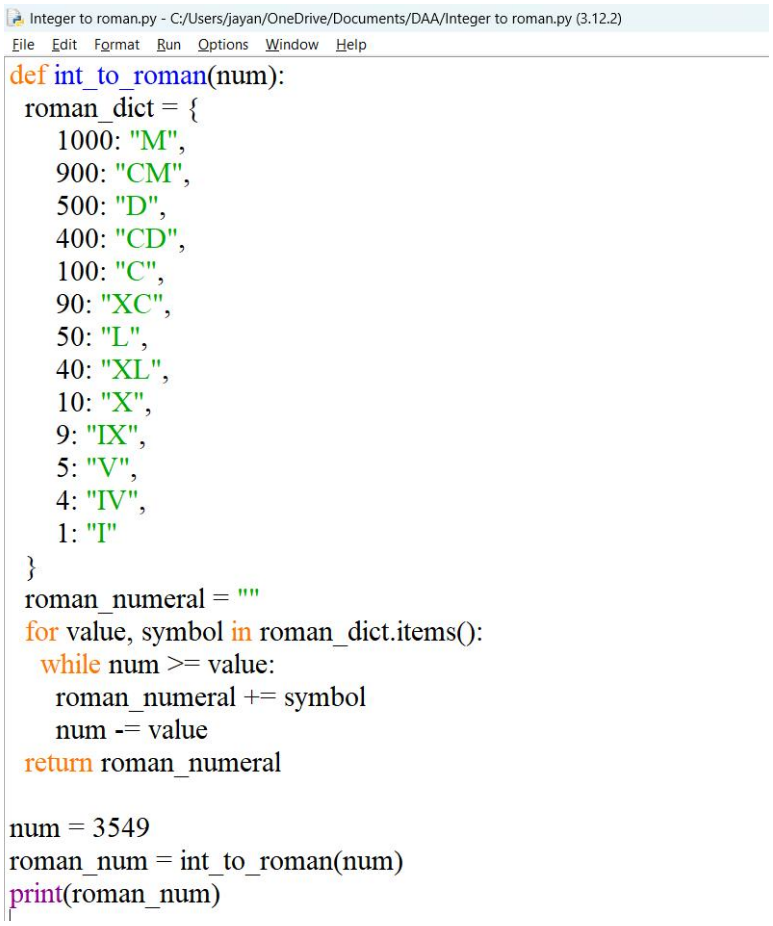
```python
def max_area(height):
  left, right = 0, len(height) - 1
  max_water = 0
  while left < right:
    width = right - left
    min_height = min(height[left], height[right])
    area = width * min_height
    max_water = max(max_water, area)
    if height[left] < height[right]:
      left += 1
    else:
      right -= 1
  return max_water

height = [1, 8, 6, 2, 5, 4, 8, 3, 7]
max_water_area = max_area(height)
print(max_water_area)
```



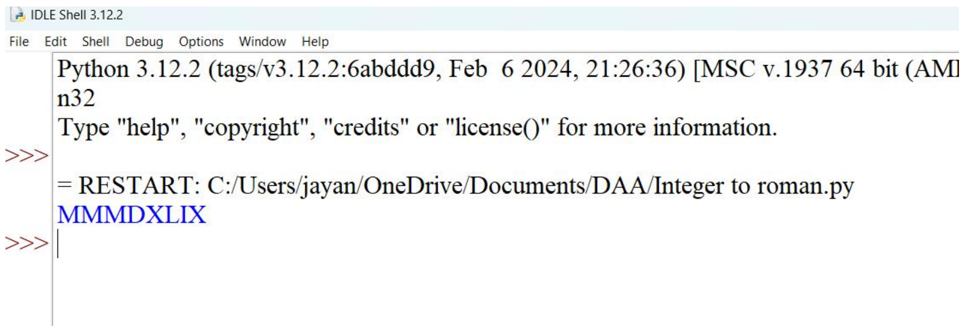
```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Container With Most Water.py
49
>>>
```

12. Integer to Roman Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Symbol Value I 1 V 5 X 10 L 50 C 100 D 500 M 1000 For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used: ● I can be placed before V (5) and X (10) to make 4 and 9. ● X can be placed before L (50) and C (100) to make 40 and 90. ● C can be placed before D (500) and M (1000) to make 400 and 900. Given an integer, convert it to a roman numeral.

Example 1: Input: num = 3 Output: "III" Explanation: 3 is represented as 3 ones.


Integer to roman.py - C:/Users/jayan/OneDrive/Documents/DAA/Integer to roman.py (3.12.2)
File Edit Format Run Options Window Help

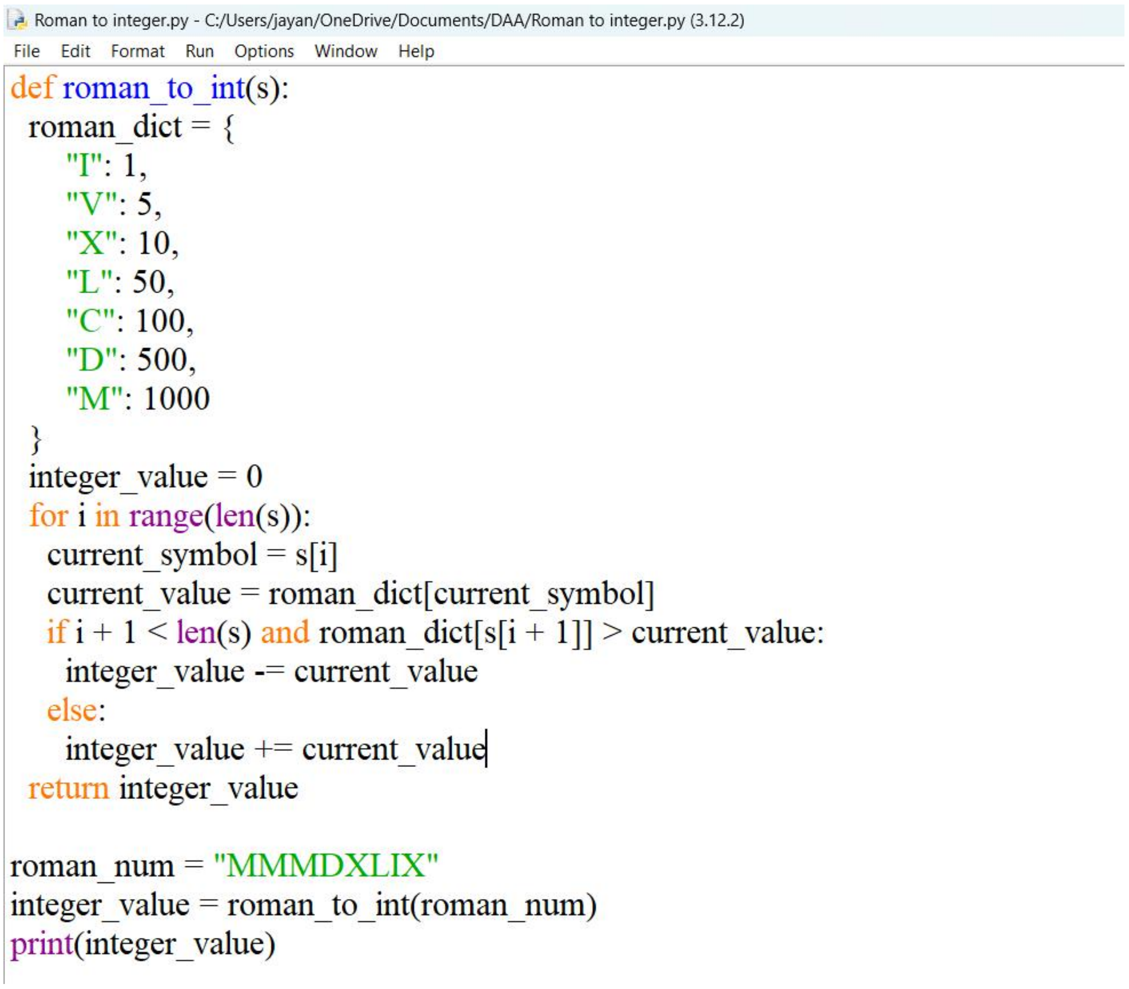
```
def int_to_roman(num):
  roman_dict = {
      1000: "M",
      900: "CM",
      500: "D",
      400: "CD",
      100: "C",
      90: "XC",
      50: "L",
      40: "XL",
      10: "X",
      9: "IX",
      5: "V",
      4: "IV",
      1: "I"
  }
  roman_numeral = ""
  for value, symbol in roman_dict.items():
    while num >= value:
      roman_numeral += symbol
      num -= value
  return roman_numeral

num = 3549
roman_num = int_to_roman(num)
print(roman_num)
```



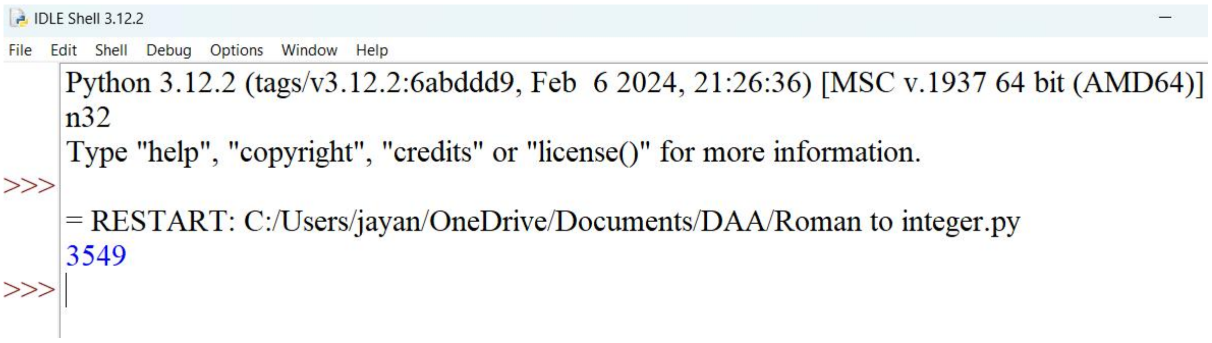
```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AM
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Integer to roman.py
MMMDXLIX
>>>
```

13. Roman to Integer Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Symbol Value I 1 V 5 X 10 L 50 C 100 D 500 M 1000 For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used: ● I can be placed before V (5) and X (10) to make 4 and 9. ● X can be placed before L (50) and C (100) to make 40 and 90. ● C can be placed before D (500) and M (1000) to make 400 and 900. Given a roman numeral, convert it to an integer.

Example 1: Input: s = "III" Output: 3 Explanation: III = 3.


Roman to integer.py - C:/Users/jayan/OneDrive/Documents/DAA/Roman to integer.py (3.12.2)
File Edit Format Run Options Window Help

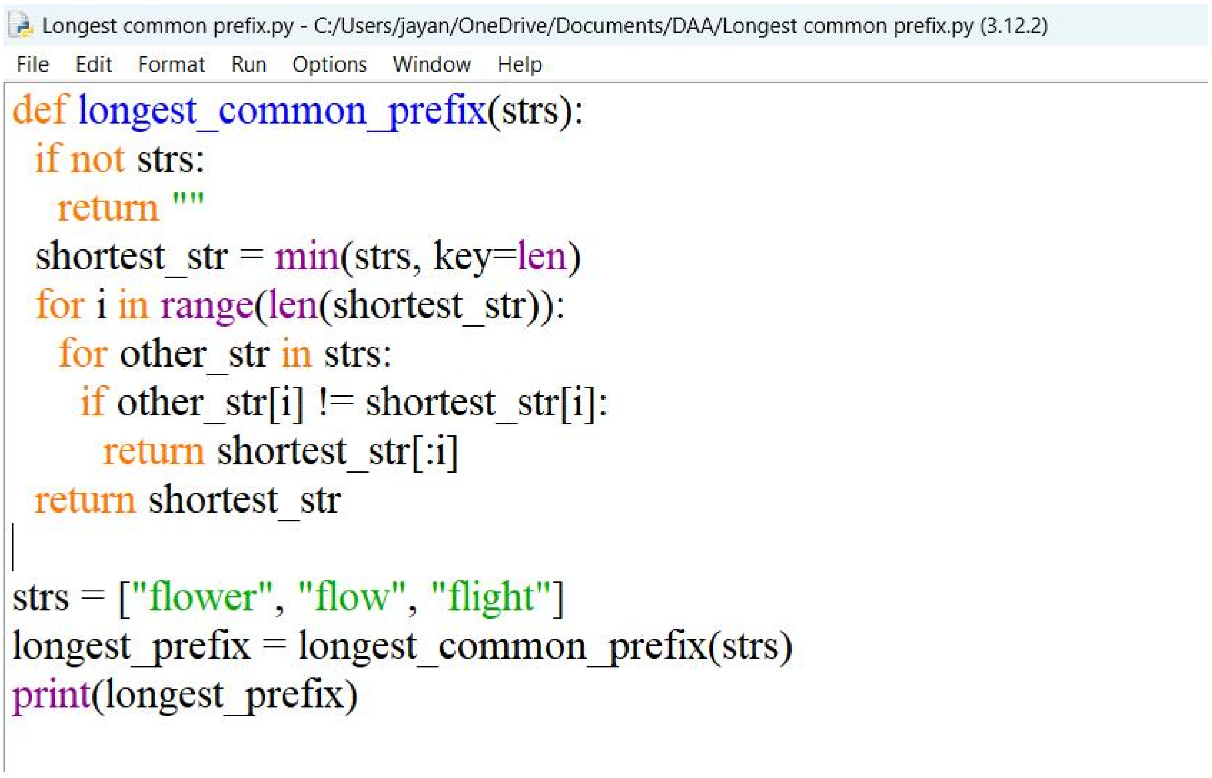
```python
def roman_to_int(s):
  roman_dict = {
      "I": 1,
      "V": 5,
      "X": 10,
      "L": 50,
      "C": 100,
      "D": 500,
      "M": 1000
  }
  integer_value = 0
  for i in range(len(s)):
    current_symbol = s[i]
    current_value = roman_dict[current_symbol]
    if i + 1 < len(s) and roman_dict[s[i + 1]] > current_value:
      integer_value -= current_value
    else:
      integer_value += current_value
  return integer_value

roman_num = "MMMDXLIX"
integer_value = roman_to_int(roman_num)
print(integer_value)
```



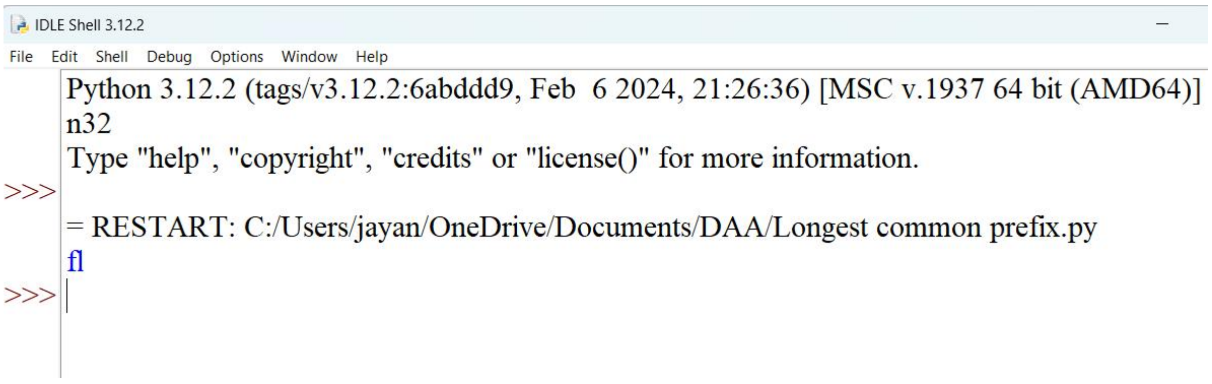
```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Roman to integer.py
3549
>>>
```

14. Longest Common Prefix Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".

Example 1: Input: strs = ["flower","flow","flight"] Output: "fl"


Longest common prefix.py - C:/Users/jayan/OneDrive/Documents/DAA/Longest common prefix.py (3.12.2)
File Edit Format Run Options Window Help

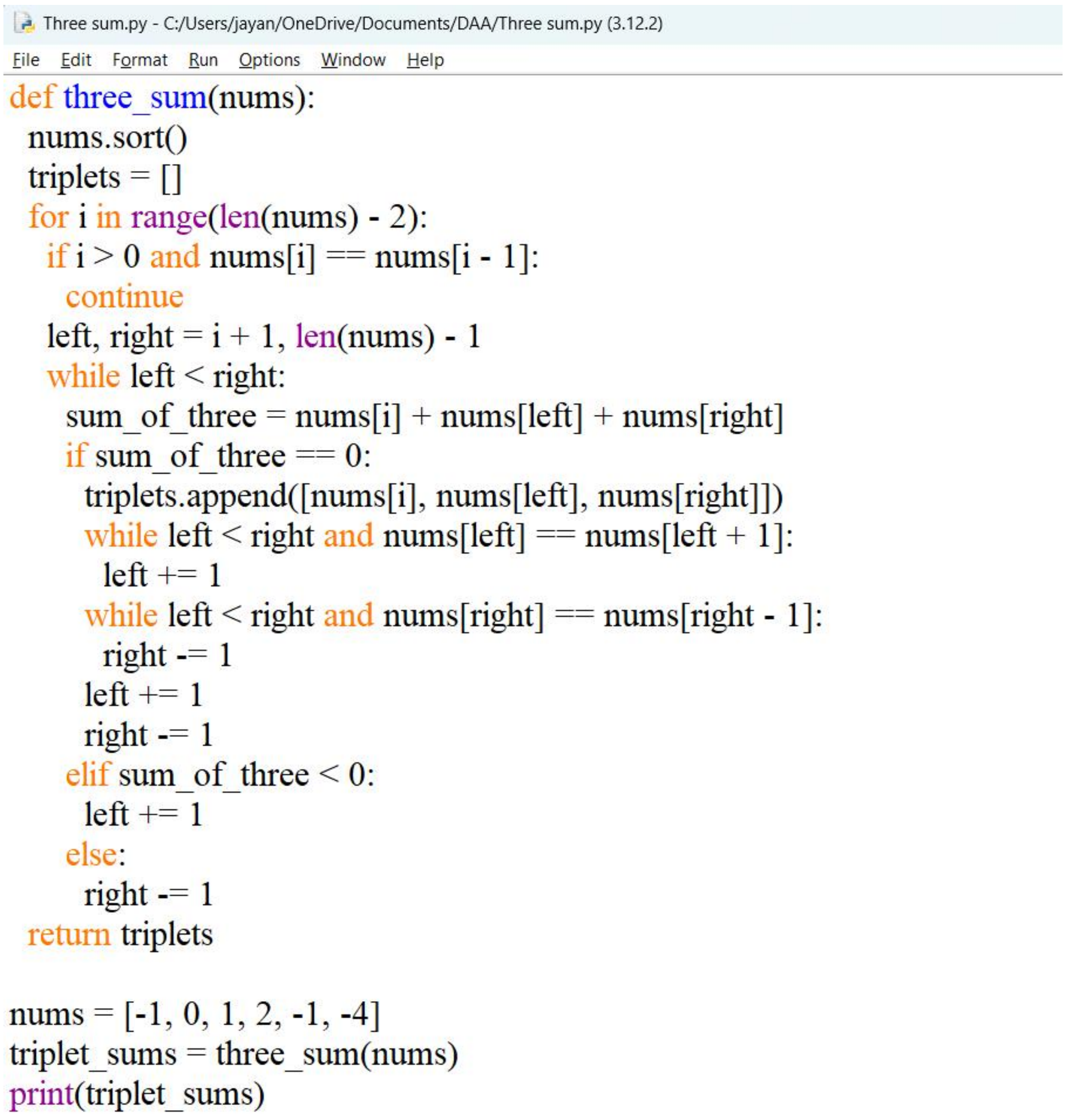
```python
def longest_common_prefix(strs):
  if not strs:
    return ""
  shortest_str = min(strs, key=len)
  for i in range(len(shortest_str)):
    for other_str in strs:
      if other_str[i] != shortest_str[i]:
        return shortest_str[:i]
  return shortest_str

strs = ["flower", "flow", "flight"]
longest_prefix = longest_common_prefix(strs)
print(longest_prefix)
```



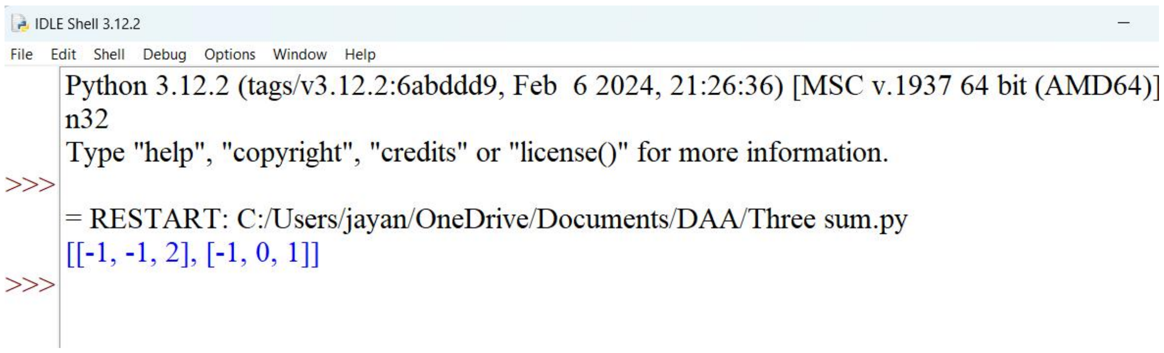
```
    Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
    n32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    = RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Longest common prefix.py
    fl
>>>
```

15. 3 Sum Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0. Notice that the solution set must not contain duplicate triplets.

Example 1: Input: nums = [-1,0,1,2,-1,-4] Output: [[-1,-1,2],[-1,0,1]] Explanation: nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0. nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0. nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0. The distinct triplets are [-1,0,1] and [-1,-1,2]. Notice that the order of the output and the order of the triplets does not matter


Three sum.py - C:/Users/jayan/OneDrive/Documents/DAA/Three sum.py (3.12.2)
File Edit Format Run Options Window Help

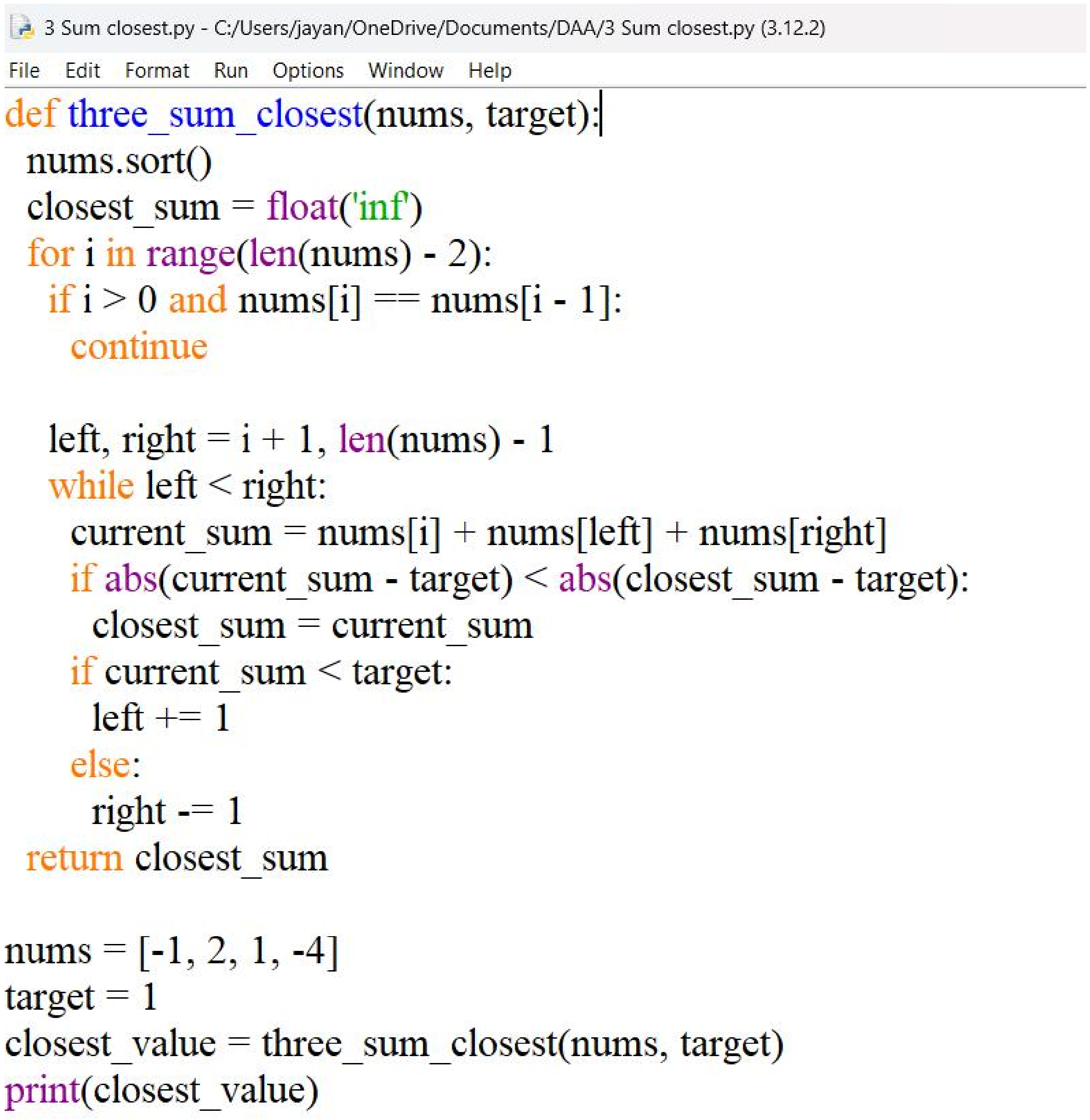
```python
def three_sum(nums):
  nums.sort()
  triplets = []
  for i in range(len(nums) - 2):
    if i > 0 and nums[i] == nums[i - 1]:
      continue
    left, right = i + 1, len(nums) - 1
    while left < right:
      sum_of_three = nums[i] + nums[left] + nums[right]
      if sum_of_three == 0:
        triplets.append([nums[i], nums[left], nums[right]])
        while left < right and nums[left] == nums[left + 1]:
          left += 1
        while left < right and nums[right] == nums[right - 1]:
          right -= 1
        left += 1
        right -= 1
      elif sum_of_three < 0:
        left += 1
      else:
        right -= 1
  return triplets

nums = [-1, 0, 1, 2, -1, -4]
triplet_sums = three_sum(nums)
print(triplet_sums)
```



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Three sum.py
[[-1, -1, 2], [-1, 0, 1]]
>>>
```

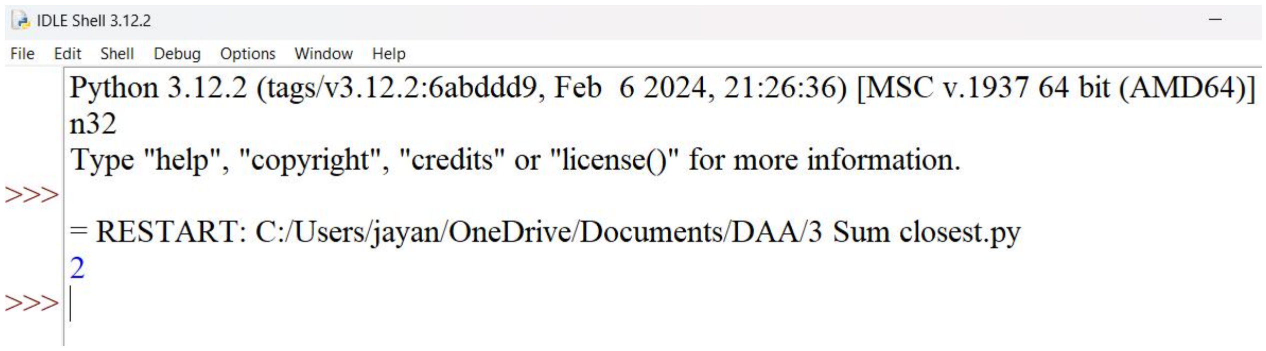
16. 3 Sum Closest Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

Example 1: Input: nums = [-1,2,1,-4], target = 1 Output: 2 Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).



3 Sum closest.py - C:/Users/jayan/OneDrive/Documents/DAA/3 Sum closest.py (3.12.2)

```
def three_sum_closest(nums, target):
  nums.sort()
  closest_sum = float('inf')
  for i in range(len(nums) - 2):
    if i > 0 and nums[i] == nums[i - 1]:
      continue

    left, right = i + 1, len(nums) - 1
    while left < right:
      current_sum = nums[i] + nums[left] + nums[right]
      if abs(current_sum - target) < abs(closest_sum - target):
        closest_sum = current_sum
      if current_sum < target:
        left += 1
      else:
        right -= 1
  return closest_sum

nums = [-1, 2, 1, -4]
target = 1
closest_value = three_sum_closest(nums, target)
print(closest_value)
```

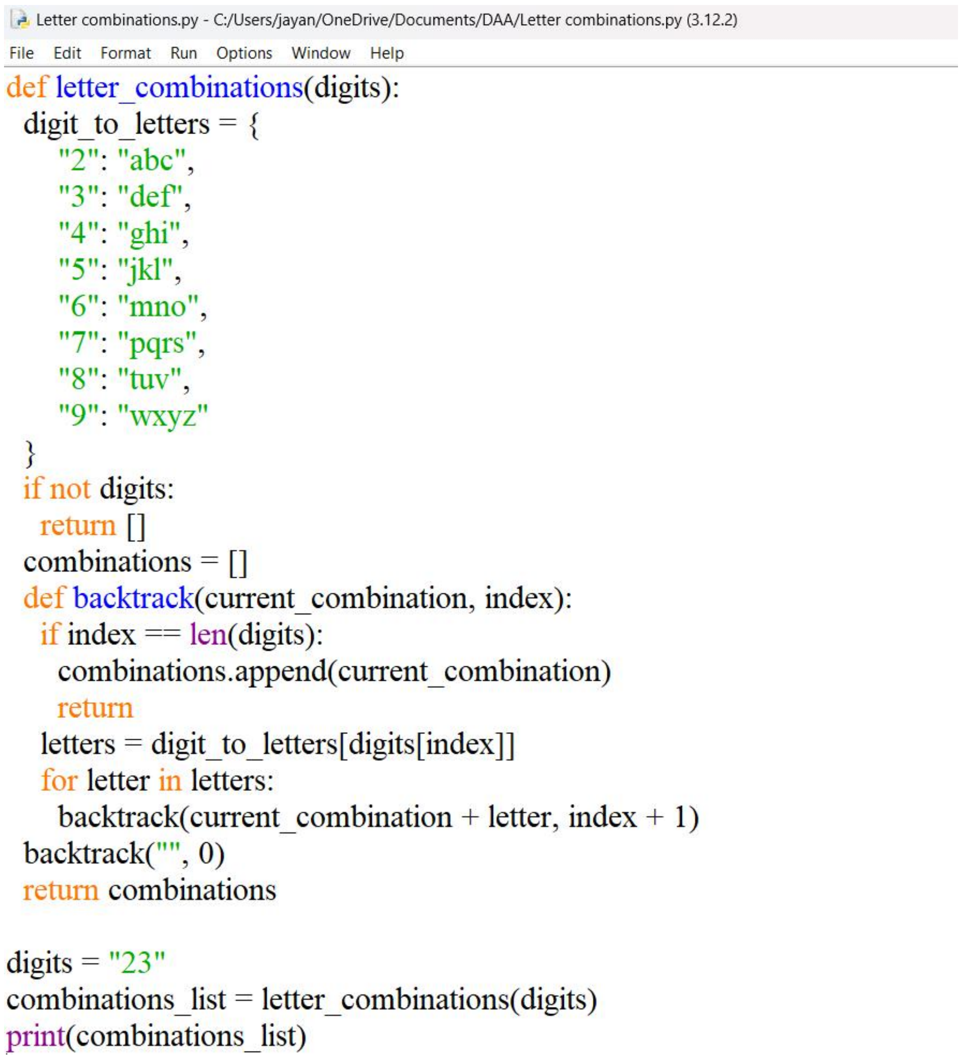


IDLE Shell 3.12.2

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/jayan/OneDrive/Documents/DAA/3 Sum closest.py
2
>>>
```

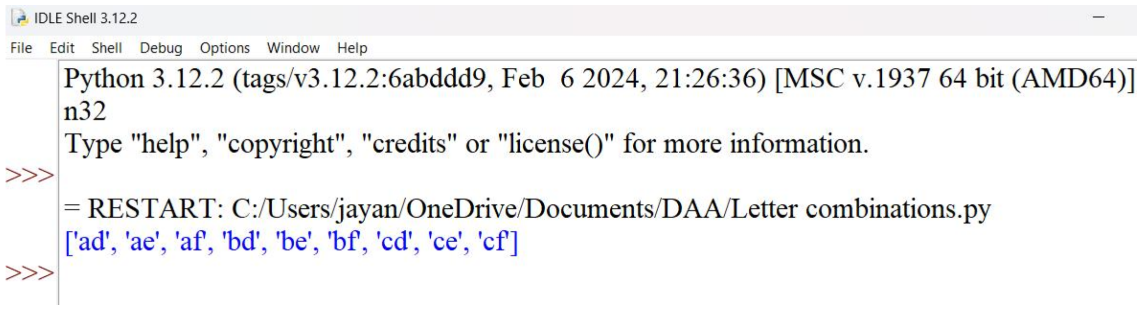
17. Letter Combinations of a Phone Number Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order. A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1: Input: digits = "23" Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]



Letter combinations.py - C:/Users/jayan/OneDrive/Documents/DAA/Letter combinations.py (3.12.2)
File Edit Format Run Options Window Help

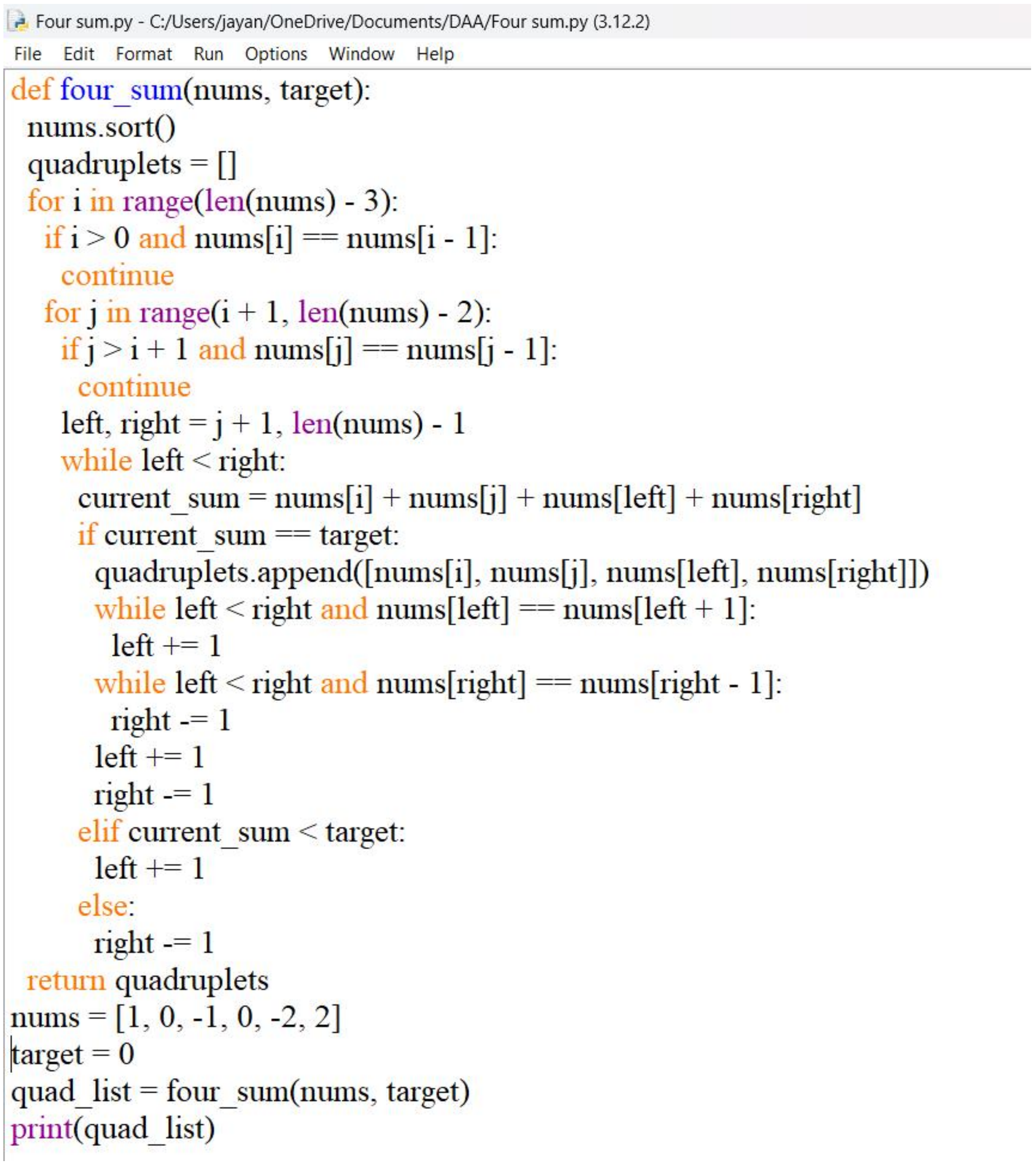
```
def letter_combinations(digits):
 digit_to_letters = {
    "2": "abc",
    "3": "def",
    "4": "ghi",
    "5": "jkl",
    "6": "mno",
    "7": "pqrs",
    "8": "tuv",
    "9": "wxyz"
 }
 if not digits:
  return []
 combinations = []
 def backtrack(current_combination, index):
  if index == len(digits):
    combinations.append(current_combination)
    return
  letters = digit_to_letters[digits[index]]
  for letter in letters:
    backtrack(current_combination + letter, index + 1)
 backtrack("", 0)
 return combinations

digits = "23"
combinations_list = letter_combinations(digits)
print(combinations_list)
```



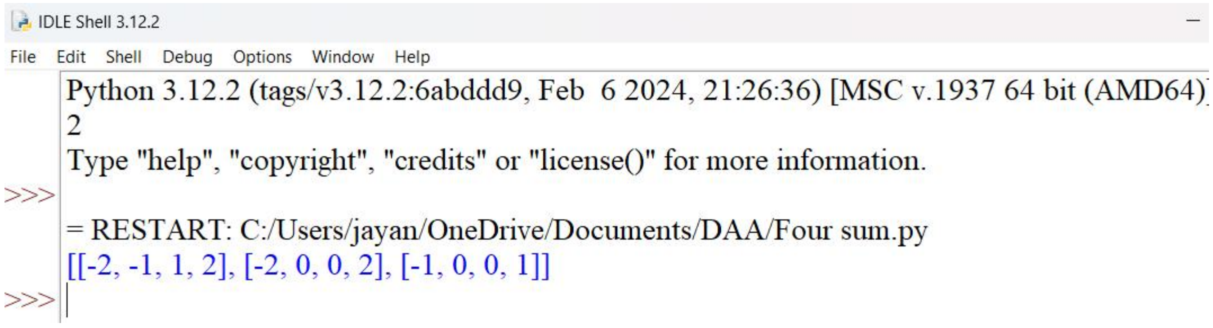
```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
n32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Letter combinations.py
['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']
>>>
```

18. 4 Sum Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that: ● 0 <= a, b, c, d < n ● a, b, c, and d are distinct. ● nums[a] + nums[b] + nums[c] + nums[d] == target You may return the answer in any order.

Example 1: Input: nums = [1,0,-1,0,-2,2], target = 0 Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]


Four sum.py - C:/Users/jayan/OneDrive/Documents/DAA/Four sum.py (3.12.2)
File Edit Format Run Options Window Help

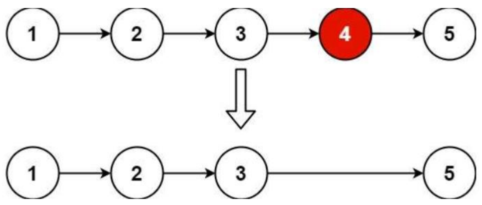
```
def four_sum(nums, target):
 nums.sort()
 quadruplets = []
 for i in range(len(nums) - 3):
  if i > 0 and nums[i] == nums[i - 1]:
   continue
  for j in range(i + 1, len(nums) - 2):
   if j > i + 1 and nums[j] == nums[j - 1]:
    continue
   left, right = j + 1, len(nums) - 1
   while left < right:
    current_sum = nums[i] + nums[j] + nums[left] + nums[right]
    if current_sum == target:
     quadruplets.append([nums[i], nums[j], nums[left], nums[right]])
     while left < right and nums[left] == nums[left + 1]:
      left += 1
     while left < right and nums[right] == nums[right - 1]:
      right -= 1
     left += 1
     right -= 1
    elif current_sum < target:
     left += 1
    else:
     right -= 1
 return quadruplets
nums = [1, 0, -1, 0, -2, 2]
target = 0
quad_list = four_sum(nums, target)
print(quad_list)
```

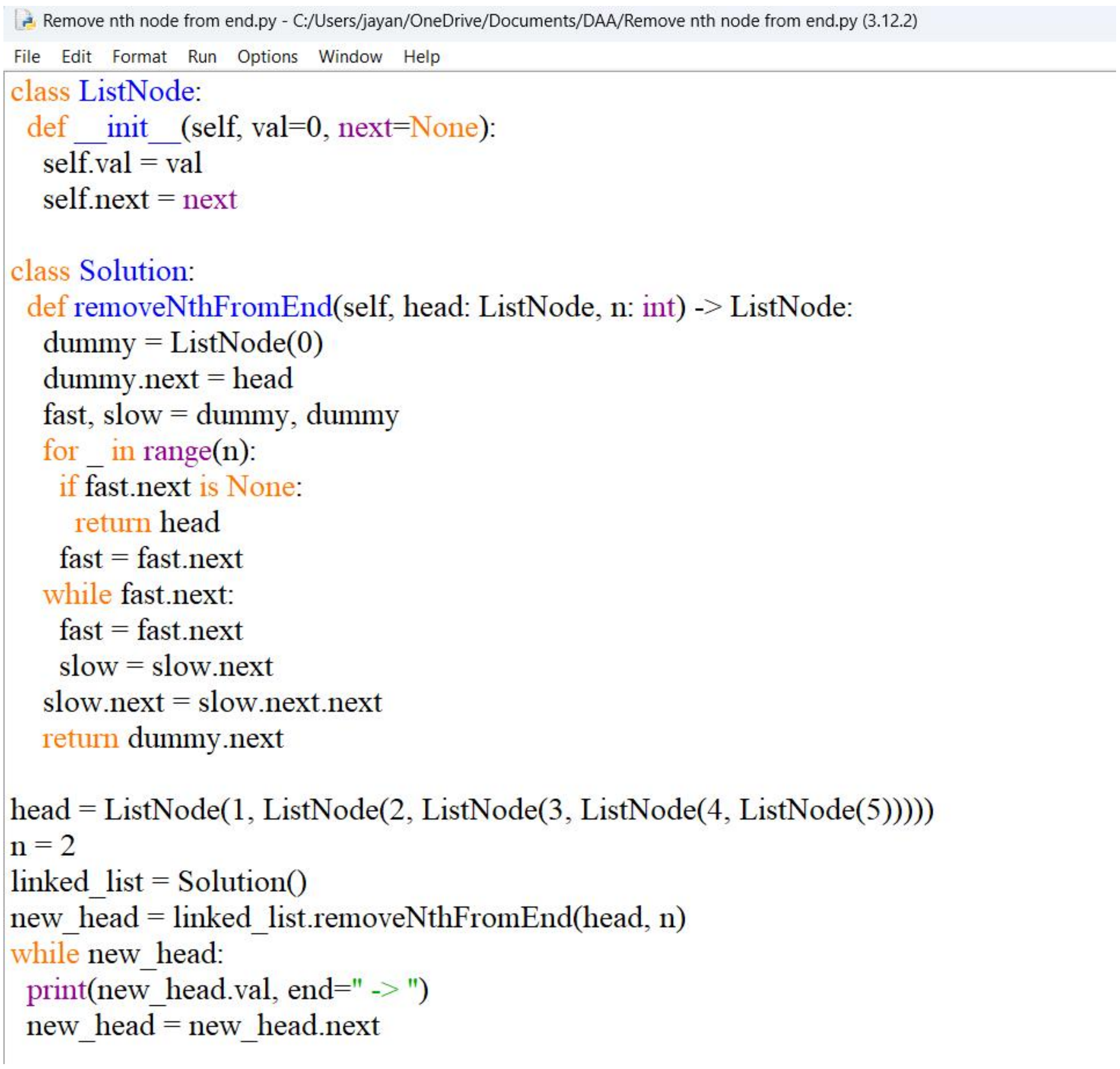


```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)
2
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Four sum.py
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
>>>
```

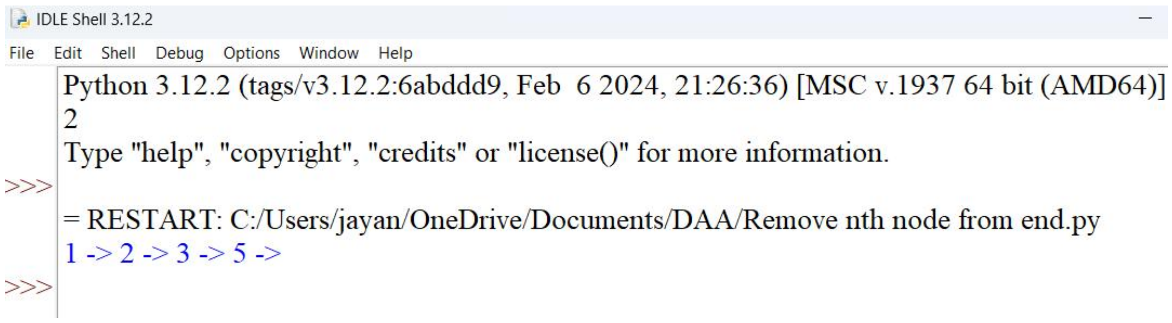
19. Remove Nth Node From End of List Given the head of a linked list, remove the nth node from the end of the list and return its head.

Example 1: Input: head = [1,2,3,4,5], n = 2 Output: [1,2,3,5]




Remove nth node from end.py - C:/Users/jayan/OneDrive/Documents/DAA/Remove nth node from end.py (3.12.2)

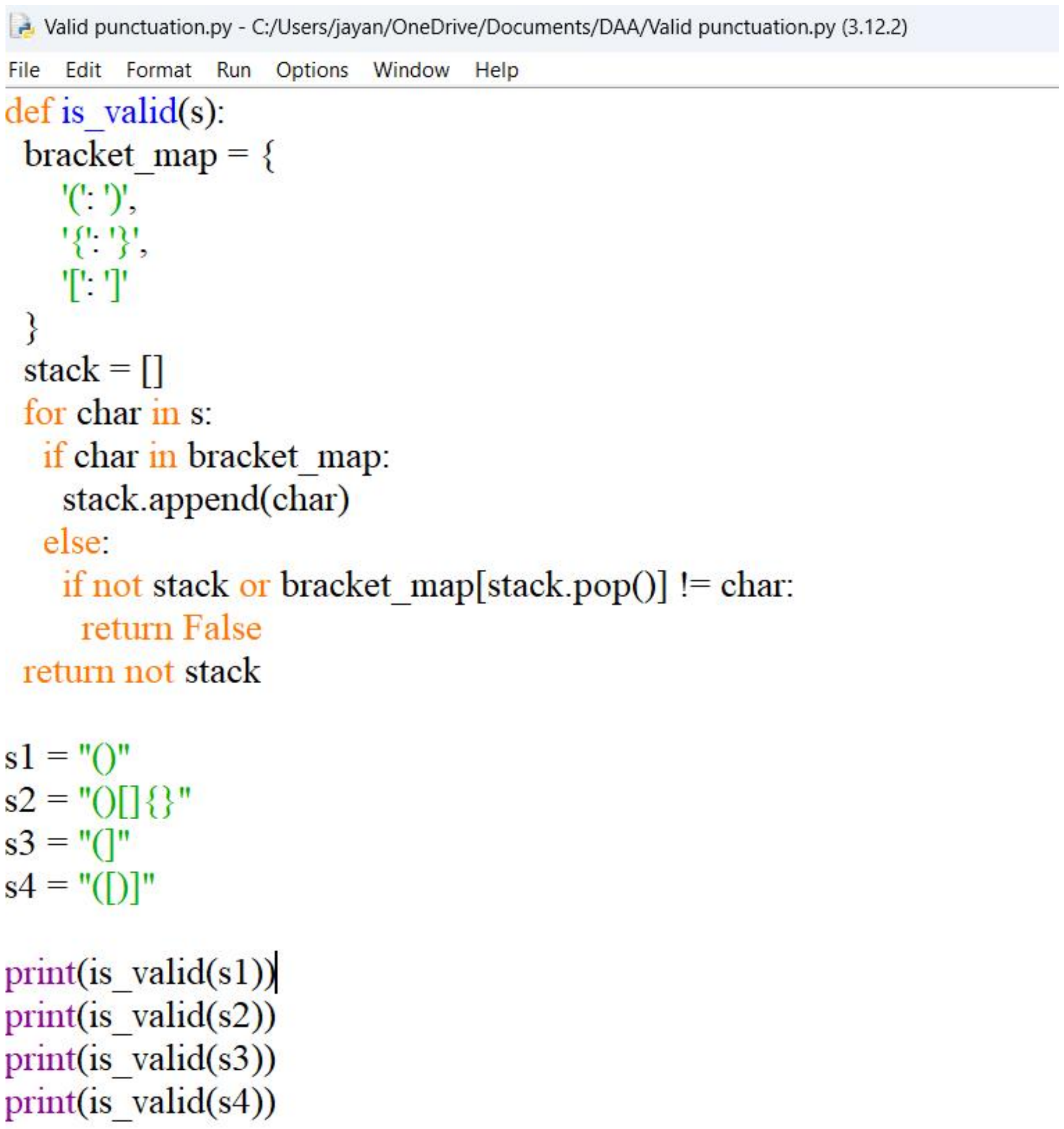
```python
class ListNode:
  def __init__(self, val=0, next=None):
    self.val = val
    self.next = next

class Solution:
  def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
    dummy = ListNode(0)
    dummy.next = head
    fast, slow = dummy, dummy
    for _ in range(n):
      if fast.next is None:
        return head
      fast = fast.next
    while fast.next:
      fast = fast.next
      slow = slow.next
    slow.next = slow.next.next
    return dummy.next

head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))
n = 2
linked_list = Solution()
new_head = linked_list.removeNthFromEnd(head, n)
while new_head:
  print(new_head.val, end=" -> ")
  new_head = new_head.next
```


IDLE Shell 3.12.2

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
2
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Remove nth node from end.py
1 -> 2 -> 3 -> 5 ->
>>>
```

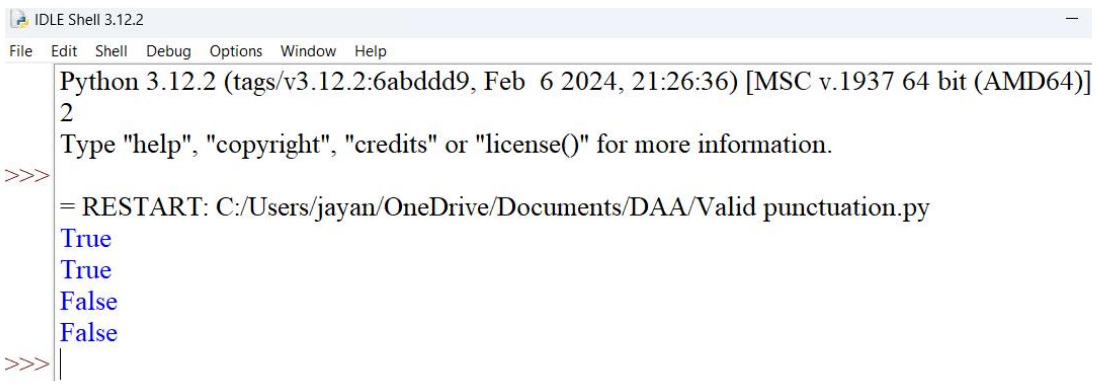
20. Valid Parentheses Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if: 1. Open brackets must be closed by the same type of brackets. 2. Open brackets must be closed in the correct order. 3. Every close bracket has a corresponding open bracket of the same type.

Example 1: Input: s = "()" Output: true



Valid punctuation.py - C:/Users/jayan/OneDrive/Documents/DAA/Valid punctuation.py (3.12.2)
File Edit Format Run Options Window Help

```
def is_valid(s):
 bracket_map = {
    '(': ')',
    '{': '}',
    '[': ']'
 }
 stack = []
 for char in s:
  if char in bracket_map:
    stack.append(char)
  else:
    if not stack or bracket_map[stack.pop()] != char:
     return False
 return not stack

s1 = "()"
s2 = "()[]{}"
s3 = "(]"
s4 = "([)]"

print(is_valid(s1))
print(is_valid(s2))
print(is_valid(s3))
print(is_valid(s4))
```



```
    Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
    2
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    = RESTART: C:/Users/jayan/OneDrive/Documents/DAA/Valid punctuation.py
    True
    True
    False
    False
>>>
```