ASSIGNMENT-1

Name: Tejaswi Narne

Reg no: 192372120

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:

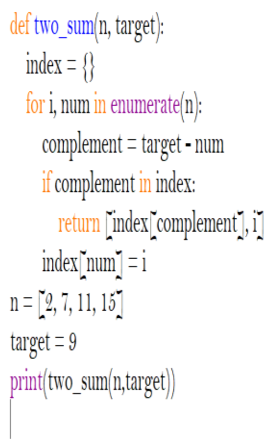
Input: nums = [2,7,11,15],

target = 9 Output: [0,1]

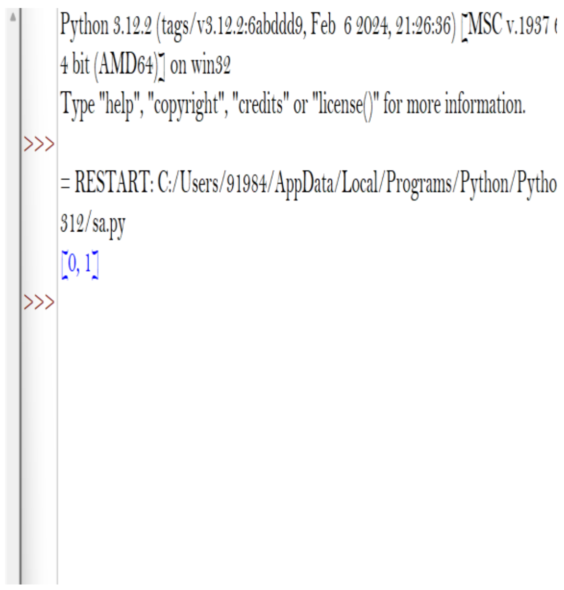
Explanation: Because nums[0] + nums[1] == 9,

we return [0, 1].

PROGRAM:



```python
def two_sum(n, target):
  index = {}
  for i, num in enumerate(n):
    complement = target - num
    if complement in index:
      return [index[complement], i]
    index[num] = i
n = [2, 7, 11, 15]
target = 9
print(two_sum(n,target))
```



```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937
4 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/91984/AppData/Local/Programs/Python/Pytho
312/sa.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.

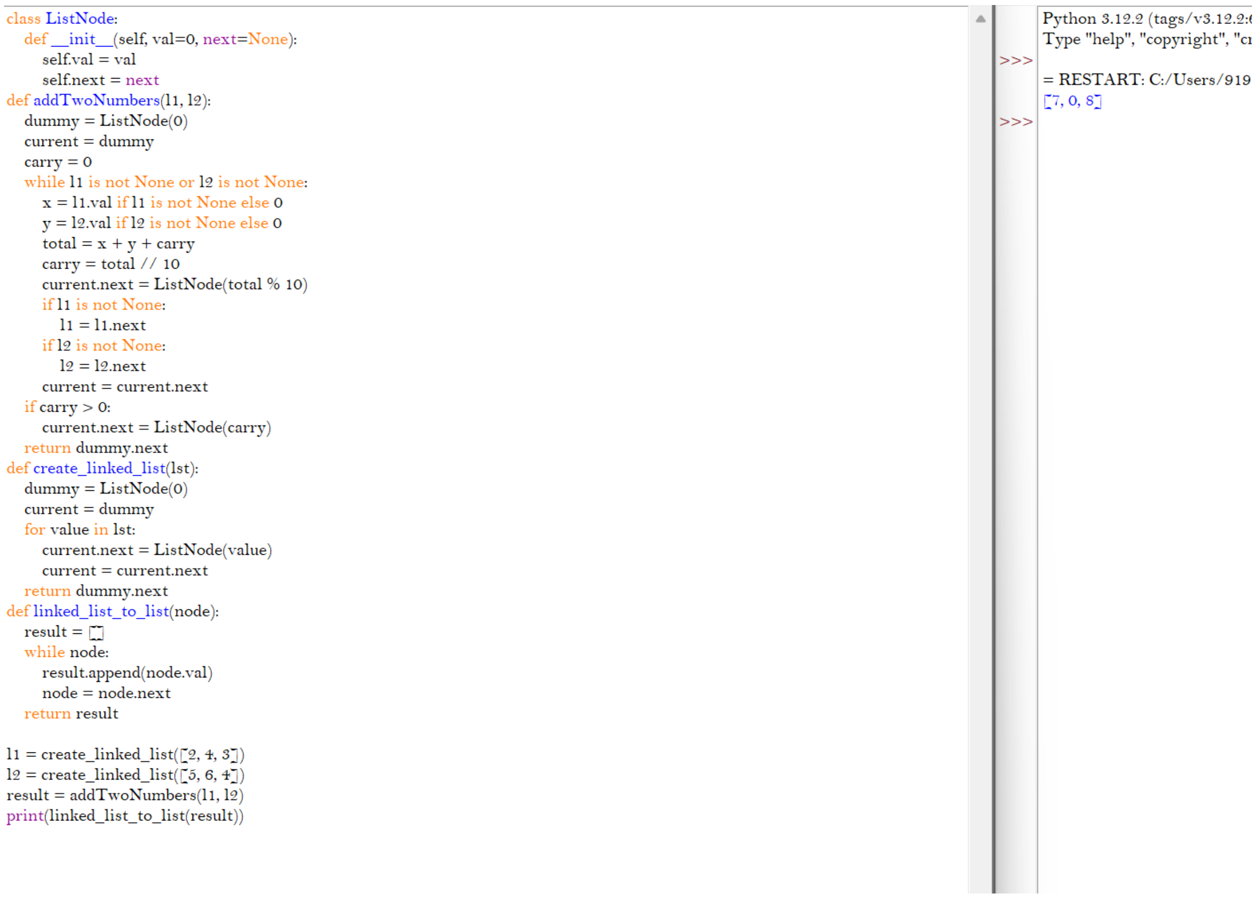
Example 1:

Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807

PROGRAM:



```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 is not None or l2 is not None:
        x = l1.val if l1 is not None else 0
        y = l2.val if l2 is not None else 0
        total = x + y + carry
        carry = total // 10
        current.next = ListNode(total % 10)
        if l1 is not None:
            l1 = l1.next
        if l2 is not None:
            l2 = l2.next
        current = current.next
    if carry > 0:
        current.next = ListNode(carry)
    return dummy.next
def create_linked_list(lst):
    dummy = ListNode(0)
    current = dummy
    for value in lst:
        current.next = ListNode(value)
        current = current.next
    return dummy.next
def linked_list_to_list(node):
    result = []
    while node:
        result.append(node.val)
        node = node.next
    return result

l1 = create_linked_list([2, 4, 3])
l2 = create_linked_list([5, 6, 4])
result = addTwoNumbers(l1, l2)
print(linked_list_to_list(result))
```

```
Python 3.12.2 (tags/v3.12.2:
Type "help", "copyright", "c
>>>
= RESTART: C:/Users/919
[7, 0, 8]
>>>
```

3. **Longest Substring without Repeating Characters**

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

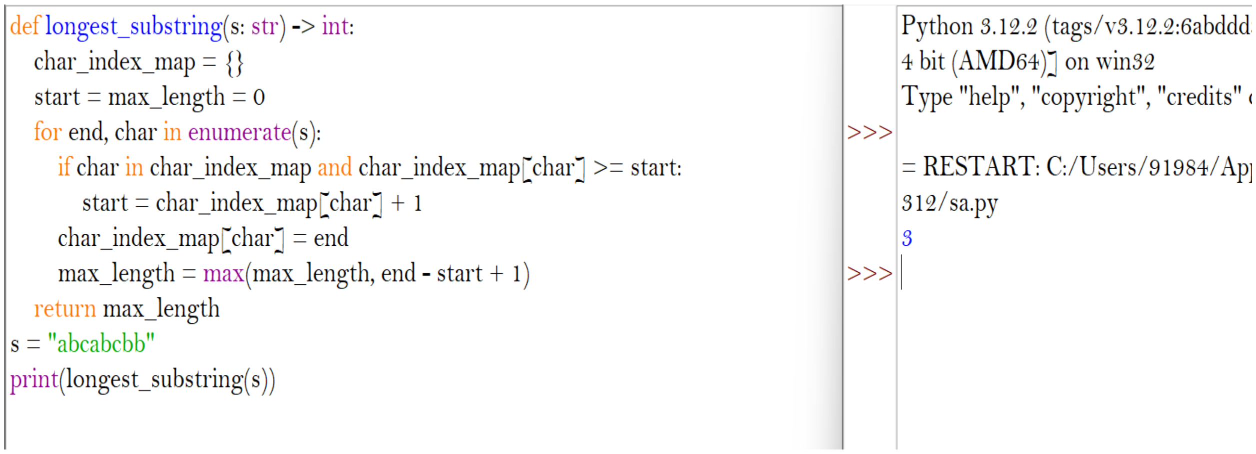
Example 1:

Input: s = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

PROGRAM:



```python
def longest_substring(s: str) -> int:
    char_index_map = {}
    start = max_length = 0
    for end, char in enumerate(s):
        if char in char_index_map and char_index_map[char] >= start:
            start = char_index_map[char] + 1
        char_index_map[char] = end
        max_length = max(max_length, end - start + 1)
    return max_length
s = "abcabcbb"
print(longest_substring(s))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd
4 bit (AMD64)] on win32
Type "help", "copyright", "credits" 
>>>
= RESTART: C:/Users/91984/Ap
312/sa.py
3
>>>
```

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)).

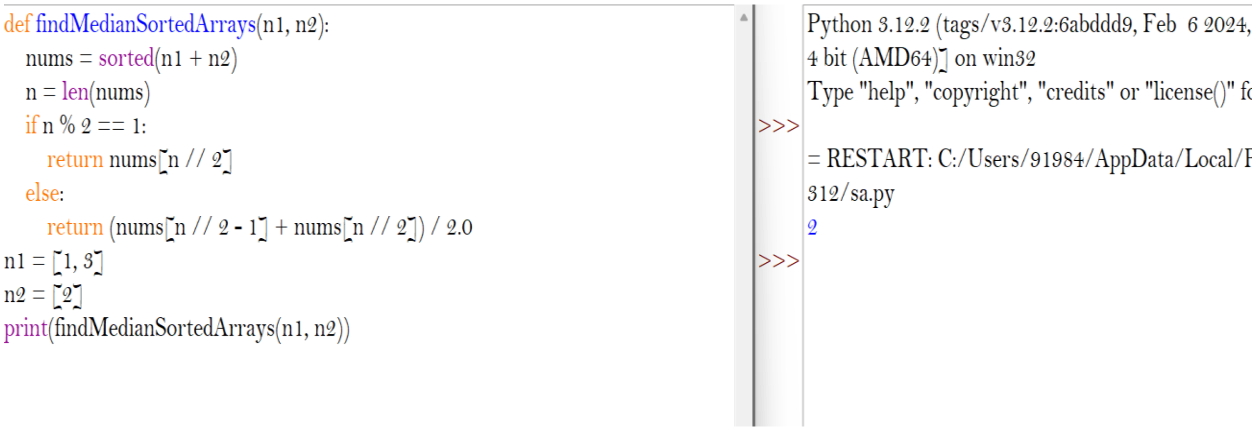
Example 1:

Input: nums1 = [1,3], nums2 = [2]

Output: 2.00000

Explanation: merged array = [1,2,3] and median is 2.

PROGRAM:



```
def findMedianSortedArrays(n1, n2):
    nums = sorted(n1 + n2)
    n = len(nums)
    if n % 2 == 1:
        return nums[n // 2]
    else:
        return (nums[n // 2 - 1] + nums[n // 2]) / 2.0
n1 = [1, 3]
n2 = [2]
print(findMedianSortedArrays(n1, n2))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024,
4 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" fo
>>>
= RESTART: C:/Users/91984/AppData/Local/P
312/sa.py
2
>>>
```

5. **Longest Palindromic Substring**

Given a string s, return the longest palindromic substring in s.

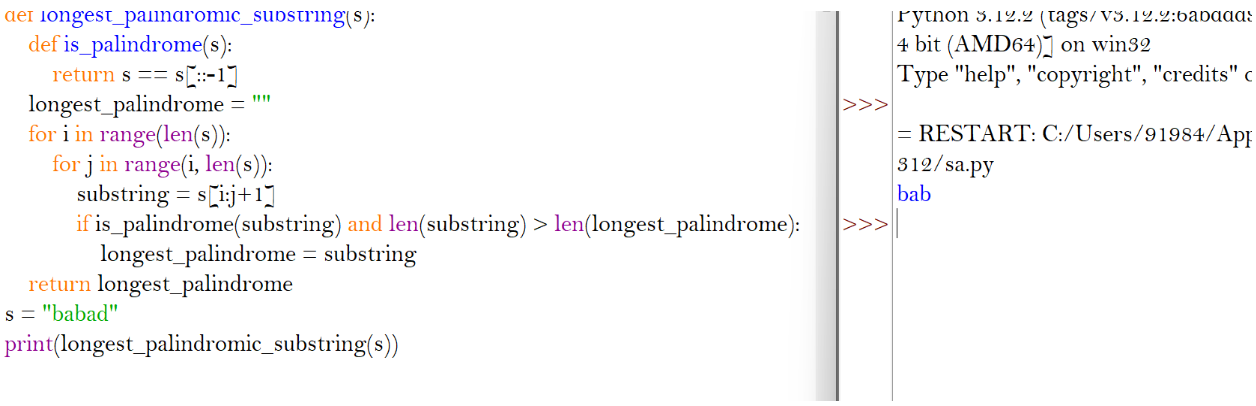
Example 1:

Input: s = "babad"

Output: "bab"

Explanation: "aba" is also a valid answer. Example 2: Input: s = "cbbd" Output: "bb"

PROGRAM:



```
def longest_palindromic_substring(s):
    def is_palindrome(s):
        return s == s[::-1]
    longest_palindrome = ""
    for i in range(len(s)):
        for j in range(i, len(s)):
            substring = s[i:j+1]
            if is_palindrome(substring) and len(substring) > len(longest_palindrome):
                longest_palindrome = substring
    return longest_palindrome
s = "babad"
print(longest_palindromic_substring(s))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd
4 bit (AMD64)] on win32
Type "help", "copyright", "credits" c
>>>
= RESTART: C:/Users/91984/App
312/sa.py
bab
>>>
```

6. **Zigzag Conversion**

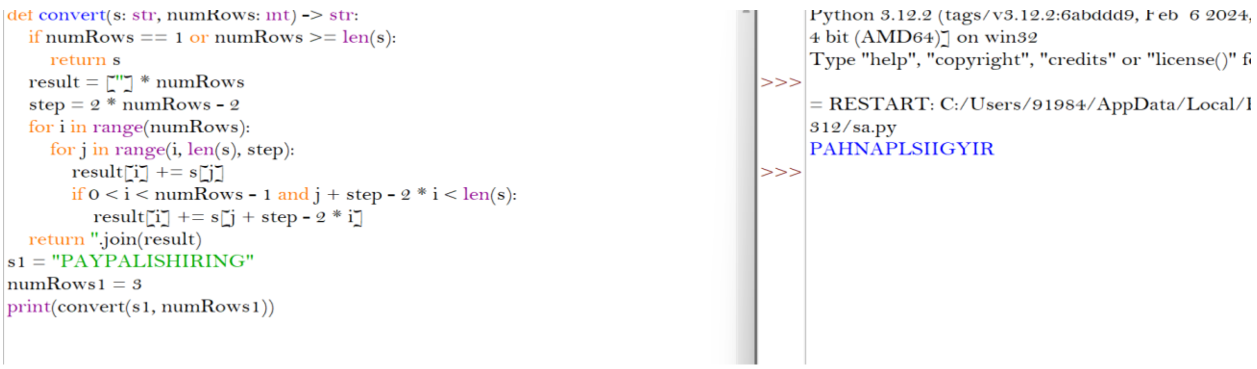
The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility) P A H N A P L S I I G Y I R And then read line by line: "PAHNAPLSIIGYIR" Write the code that will take a string and make this conversion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

Example 1:

Input: s = "PAYPALISHIRING", numRows = 3

Output: "PAHNAPLSIIGYIR

PROGRAM:



```
def convert(s: str, numRows: int) -> str:
    if numRows == 1 or numRows >= len(s):
        return s
    result = [""] * numRows
    step = 2 * numRows - 2
    for i in range(numRows):
        for j in range(i, len(s), step):
            result[i] += s[j]
            if 0 < i < numRows - 1 and j + step - 2 * i < len(s):
                result[i] += s[j + step - 2 * i]
    return ''.join(result)
s1 = "PAYPALISHIRING"
numRows1 = 3
print(convert(s1, numRows1))
```

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024,
4 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" f
>>>
= RESTART: C:/Users/91984/AppData/Local/P
312/sa.py
PAHNAPLSIIGYIR
>>>
```

7. **Reverse Integer**

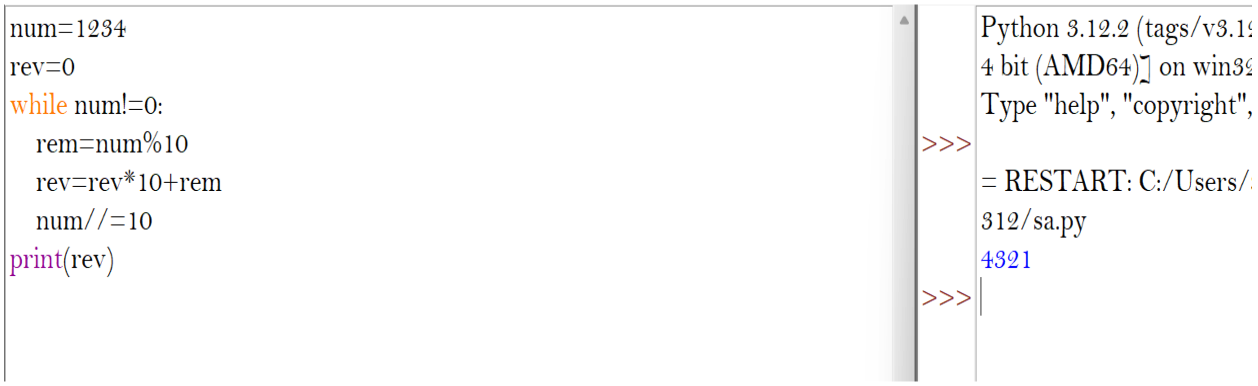
Given a signed 32-bit integer x, return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0. Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Example 1:

Input: x = 123

Output: 321

PROGRAM:



```
num=1234
rev=0
while num!=0:
    rem=num%10
    rev=rev*10+rem
    num//=10
print(rev)
```

```
Python 3.12.2 (tags/v3.1
4 bit (AMD64)] on win3
Type "help", "copyright",
>>>
= RESTART: C:/Users/
312/sa.py
4321
>>>
```

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. Note: ● Only the space character ' ' is considered a whitespace character. ● 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.

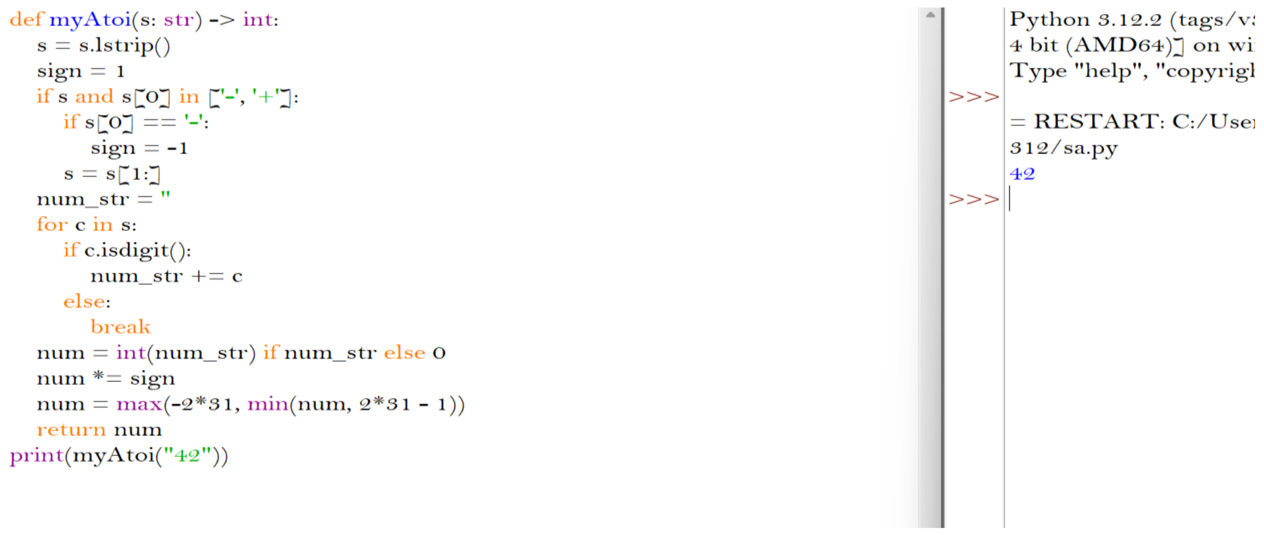
Step 1: "42" (no characters read because there is no leading whitespace) ^

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.

PROGRAM:



```python
def myAtoi(s: str) -> int:
    s = s.lstrip()
    sign = 1
    if s and s[0] in ['-', '+']:
        if s[0] == '-':
            sign = -1
        s = s[1:]
    num_str = ''
    for c in s:
        if c.isdigit():
            num_str += c
        else:
            break
    num = int(num_str) if num_str else 0
    num *= sign
    num = max(-2*31, min(num, 2*31 - 1))
    return num
print(myAtoi("42"))
```

```
    Python 3.12.2 (tags/v:
    4 bit (AMD64)] on wi
    Type "help", "copyrig
>>>
    = RESTART: C:/Use
    312/sa.py
    42
>>>
```

9. **Palindrome Number**

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

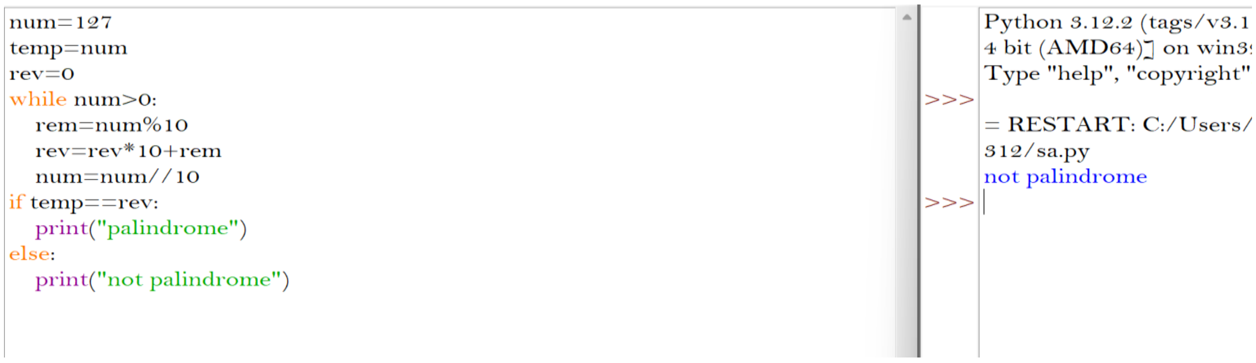
Example 1:

Input: x = 121

Output: true

Explanation: 121 reads as 121 from left to right and from right to left

PROGRAM:



```
num=127
temp=num
rev=0
while num>0:
    rem=num%10
    rev=rev*10+rem
    num=num//10
if temp==rev:
    print("palindrome")
else:
    print("not palindrome")
```

```
Python 3.12.2 (tags/v3.1
4 bit (AMD64)] on win3
Type "help", "copyright"
>>>
= RESTART: C:/Users/
312/sa.py
not palindrome
>>>
```

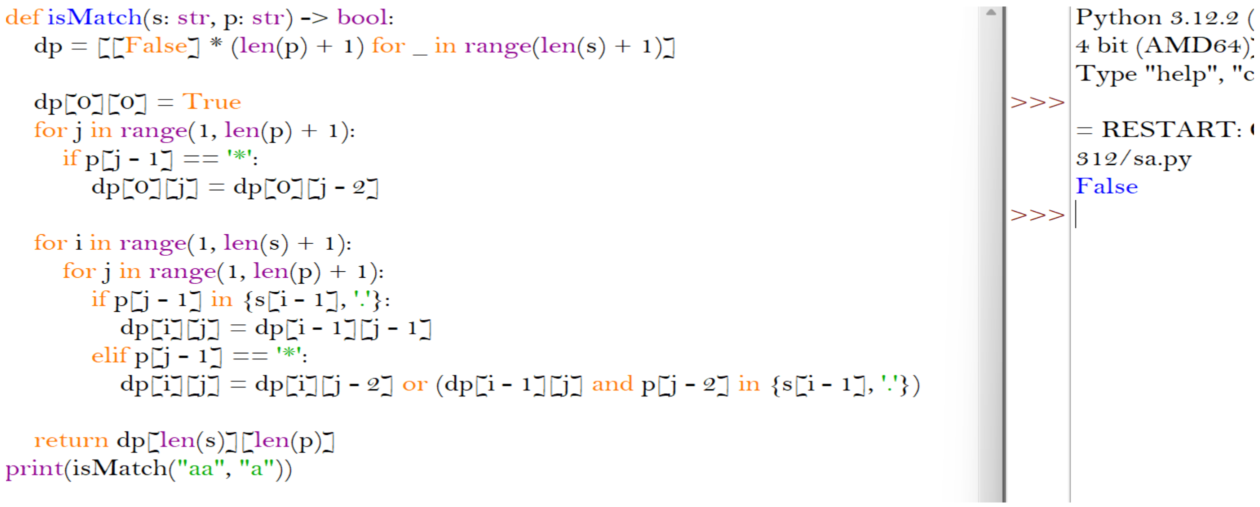
10. **Regular Expression Matching**

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"

PROGRAM:



```
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] in {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] in {s[i - 1], '.'})

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

```
Python 3.12.2 (
4 bit (AMD64)
Type "help", "c
>>>
= RESTART:
312/sa.py
False
>>>
```