

1. 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:

Output: `def two_sum(nums, target):`

`seen = {}`

`for i, num in enumerate(nums):`

`complement = target - num`

`if complement in seen:`

`return [seen[complement], i]`

`seen[num] = i`

`return []`

`def main():`

`nums = [2, 7, 11, 15]`

`target = 9`

`result = two_sum(nums, target)`

`print("Output:", result)`

`if __name__ == "__main__":`

`main()`

```
PS C:\Users\karth> & c:/Users/karth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/karth/OneDrive/Desktop/daa.py
Output: []
PS C:\Users\karth>
```

2. 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.

Program:

```

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

def addTwoNumbers(l1, l2):
    dummy_head = ListNode(0)
    current = dummy_head
    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 = carry + x + y
        carry = total // 10
        current.next = ListNode(total % 10)
        current = current.next

        if l1 is not None: l1 = l1.next
        if l2 is not None: l2 = l2.next

    if carry > 0:
        current.next = ListNode(carry)
    return dummy_head.next

def create_linked_list(lst):
    dummy_head = ListNode(0)
    current = dummy_head
    for number in lst:
        current.next = ListNode(number)
        current = current.next
    return dummy_head.next

def print_linked_list(node):
    while node is not None:
        print(node.val, end=" ")
        node = node.next

```

```

print()

l1 = create_linked_list([2, 4, 3])
l2 = create_linked_list([5, 6, 4])

result = addTwoNumbers(l1, l2)

```

```

print("Resultant linked list:")
print_linked_list(result)

```

Output:

```

PS C:\Users\karth> & C:/Users/karth/AppData/Local/Programs/Python/Python31
Resultant linked list:
4 5 7
PS C:\Users\karth>

```

3. 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:

```

def length_of_longest_substring(s: str) -> int:
    n = len(s)
    max_length = 0
    char_index_map = {}
    start = 0
    for end in range(n):
        if s[end] in char_index_map and char_index_map[s[end]] >=
start:
            start = char_index_map[s[end]] + 1

        char_index_map[s[end]] = end
        max_length = max(max_length, end - start + 1)
    return max_length
s = "abcabcbb"

```

```
print(length_of_longest_substring(s))
```

Output:

```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe C:\Users\srika\Desktop\CSA0863\pythonProject\problem.py
3
Process finished with exit code 0
```

4. 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(nums1, nums2):
    total_len = len(nums1) + len(nums2)
    if total_len % 2 == 0:
        return (findKth(total_len // 2, nums1, nums2) + findKth(total_len // 2 - 1, nums1, nums2)) / 2
    else:
        return findKth(total_len // 2, nums1, nums2)
def findKth(k, nums1, nums2):
    if not nums1:
        return nums2[k]
    if not nums2:
        return nums1[k]
    len1, len2 = len(nums1), len(nums2)
    mid1, mid2 = len1 // 2, len2 // 2
    mid_val1, mid_val2 = nums1[mid1], nums2[mid2]
    if mid1 + mid2 < k:
        if mid_val1 > mid_val2:
            return findKth(k - mid2 - 1, nums1, nums2[mid2 + 1:])
        else:
            return findKth(k - mid1 - 1, nums1[mid1 + 1:], nums2)
    else:
        if mid_val1 > mid_val2:
            return findKth(k, nums1[:mid1], nums2)
        else:
            return findKth(k, nums1, nums2[:mid2])
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))
```

Output:

```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe C:\Users\srika\Desktop\CSA0863\pythonProject\problem.py
2
Process finished with exit code 0
```

5. 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.

Program:

```
def longestPalindrome(s: str) -> str:
    n = len(s)
    if n == 0:
        return ""
    start = 0
    max_length = 1
    dp = [[False] * n for _ in range(n)]
    for i in range(n):
        dp[i][i] = True
    for i in range(n - 1):
        if s[i] == s[i + 1]:
            dp[i][i + 1] = True
            start = i
            max_length = 2
    for length in range(3, n + 1):
        for i in range(n - length + 1):
            j = i + length - 1
            if s[i] == s[j] and dp[i + 1][j - 1]:
                dp[i][j] = True
                start = i
                max_length = length
    return s[start:start + max_length]
s = "babad"
print(longestPalindrome(s))
```

Output:



```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe C:\Users\srika\Desktop\CSA0863\pythonProject\problem.py
aba

Process finished with exit code 0
```

6. 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
    rows = [''] * numRows
    index, step = 0, 1
    for char in s:
        rows[index] += char
        if index == 0:
            step = 1
```

```

        step = 1
    elif index == numRows - 1:
        step = -1
    index += step
    return ".join(rows)
s = "varun"
numRows = 3
print(convert(s, numRows))
Output:

```

```

C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe C:\Users\srika\Desktop\CSA0863\pythonProject\problem.py
vnaur
Process finished with exit code 0

```

7. 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 $[-2^{31}, 2^{31} - 1]$, 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:

```

def reverse(x: int) -> int:
    INT_MIN, INT_MAX = -2 ** 31, 2 ** 31 - 1
    str_x = str(x)
    if str_x[0] == '-':
        sign = -1
        str_x = str_x[1:]
    else:
        sign = 1
    str_x_reversed = str_x[::-1]
    reversed_x = int(str_x_reversed) * sign
    if reversed_x < INT_MIN or reversed_x > INT_MAX:
        return 0
    else:
        return reversed_x

```

$x = 123$

`print(reverse(x))`

Output:

```

C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe C:\Users\srika\Desktop\CSA0863\pythonProject\problem.py
321
Process finished with exit code 0

```

8. 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.

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

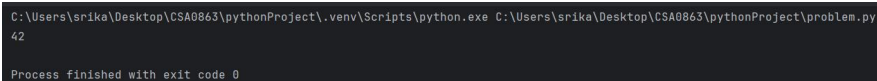
Program:

```
def myAtoi(s: str) -> int:
    INT_MIN, INT_MAX = -2 ** 31, 2 ** 31 - 1
    s = s.strip()
    if not s:
        return 0
    if s[0] in ['- ', '+ ']:
        sign = -1 if s[0] == '-' else 1
        s = s[1:]
    else:
        sign = 1
    num = 0
    for char in s:
        if not char.isdigit():
            break
        num = num * 10 + int(char)
    num *= sign
    if num < INT_MIN:
        return INT_MIN
    elif num > INT_MAX:
        return INT_MAX
    else:
        return num
```

`s = "42"`

`print(myAtoi(s))`

Output:



```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe C:\Users\srika\Desktop\CSA0863\pythonProject\problem.py
42
Process finished with exit code 0
```

9. 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:

```
def isPalindrome(x: int) -> bool:
    str_x = str(x)
    return str_x == str_x[::-1]
```

`x = 121`

`print(isPalindrome(x))`

Output:

```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe C:\Users\srika\Desktop\CSA0863\pythonProject\problem.py
True

Process finished with exit code 0
```

10. 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] == '*' and dp[0][j - 2]:
            dp[0][j] = True
    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)]

s = "aa"
p = "a"
print(isMatch(s, p))
```

Output:

```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe C:\Users\srika\Desktop\CSA0863\pythonProject\problem.py
False

Process finished with exit code 0
```