# Experiment 1.4

**Student Name: Yash Gupta** **UID: 20BCS5009**
**Branch: BE-CSE** **Section/Group:20BCS_DM-716 B**
**Semester: 6** **Date of Performance:06/03/23**
**Subject Name: CC LAB** **Subject Code: 20CSP_351**

## 1. Aim:

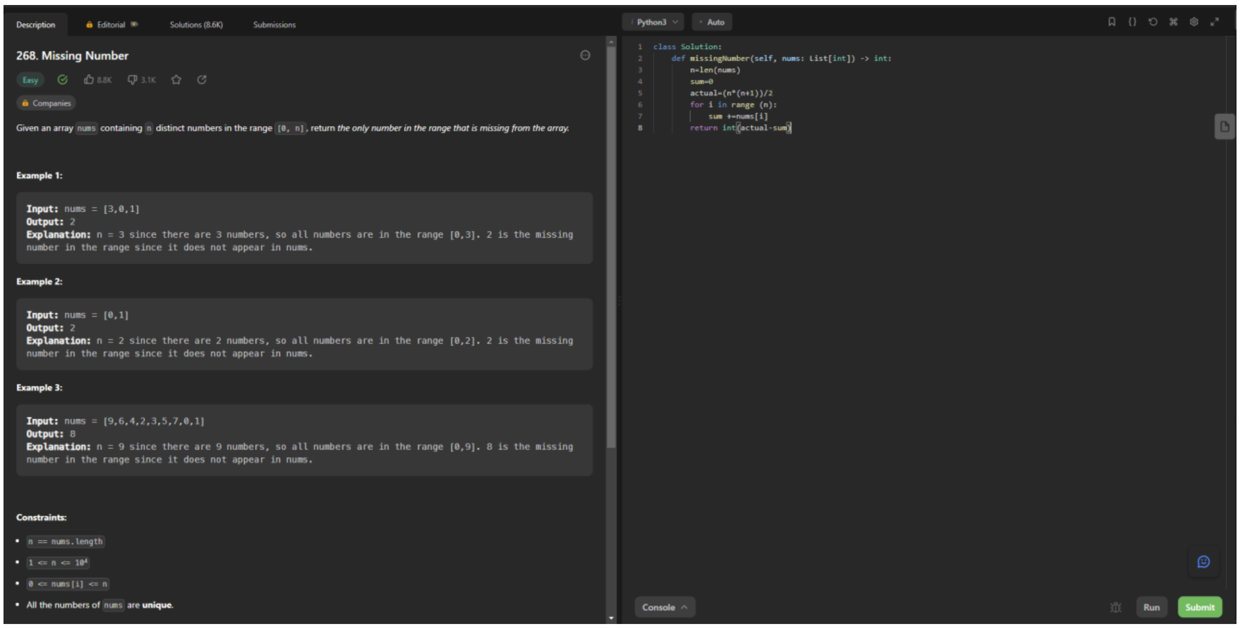
To implement the concept of hashing.

## 2. Objective:

- The objective is to build problem solving capability and to learn the basic concepts of data structures.
- Understand the problem and find out better approach to solve particular problem

## 3. LeetCode code and output:

- **Missing Number**



268. Missing Number

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return *the only number in the range that is missing from the array.*

**Example 1:**

```
Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing
number in the range since it does not appear in nums.
```

**Example 2:**

```
Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing
number in the range since it does not appear in nums.
```

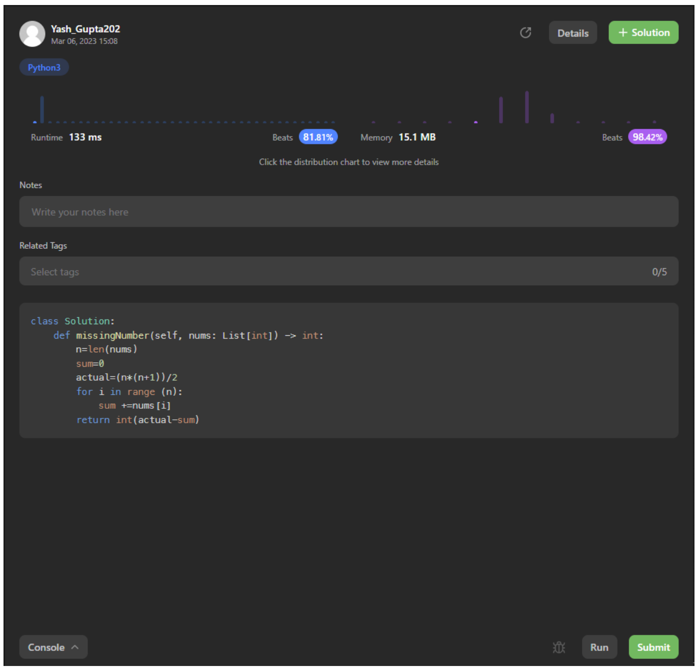
**Example 3:**

```
Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing
number in the range since it does not appear in nums.
```

**Constraints:**

- `n == nums.length`
- `1 <= n <= 10^4`
- `0 <= nums[i] <= n`
- All the numbers of `nums` are **unique.**

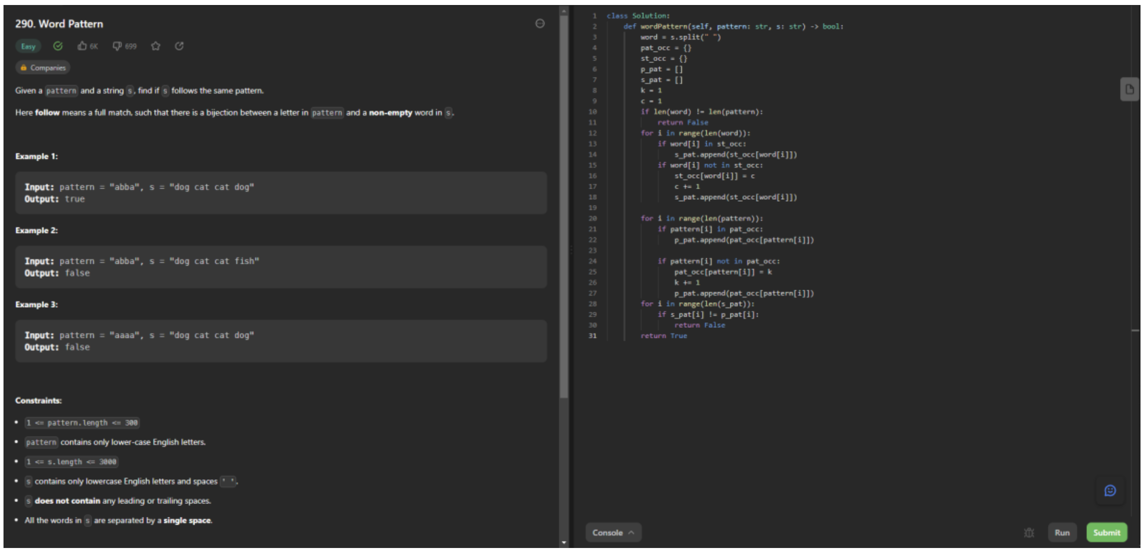
```python
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        n=len(nums)
        sum=0
        actual=(n*(n+1))/2
        for i in range (n):
            sum +=nums[i]
        return int(actual-sum)
```



```python
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        n=len(nums)
        sum=0
        actual=(n*(n+1))/2
        for i in range (n):
            sum +=nums[i]
        return int(actual-sum)
```

```python
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        n=len(nums)
        sum=0
        actual=(n*(n+1))/2
        for i in range (n):
            sum +=nums[i]
        return int(actual-sum)
```

- **Word Pattern**



290. Word Pattern

Given a `pattern` and a string `s`, find if `s` follows the same pattern.

Here **follow** means a full match, such that there is a bijection between a letter in `pattern` and a **non-empty** word in `s`.

**Example 1:**

```
Input: pattern = "abba", s = "dog cat cat dog"
Output: true
```

**Example 2:**

```
Input: pattern = "abba", s = "dog cat cat fish"
Output: false
```

**Example 3:**

```
Input: pattern = "aaaa", s = "dog cat cat dog"
Output: false
```

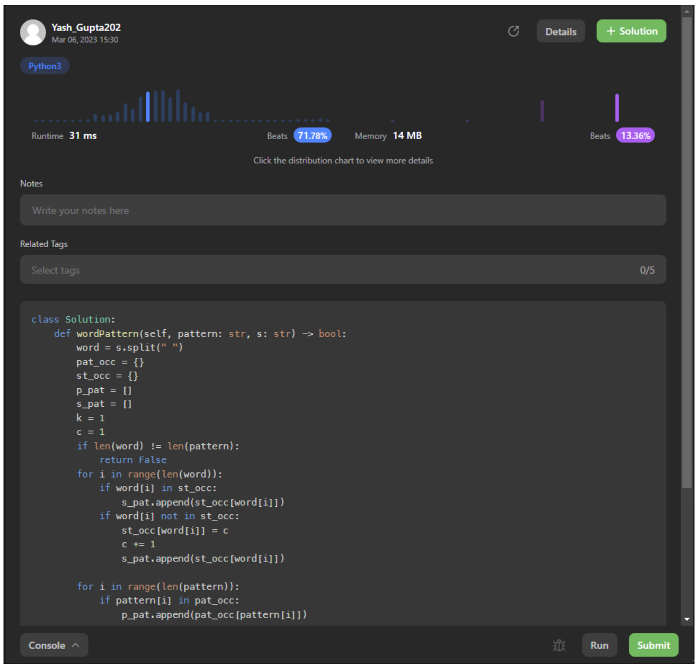
**Constraints:**

- `1 <= pattern.length <= 300`
- `pattern` contains only lower-case English letters.
- `1 <= s.length <= 3000`
- `s` contains only lowercase English letters and spaces `' '`.
- `s` **does not contain** any leading or trailing spaces.
- All the words in `s` are separated by a **single space**.

```python
class Solution:
    def wordPattern(self, pattern: str, s: str) -> bool:
        word = s.split(" ")
        pat_occ = {}
        st_occ = {}
        p_pat = []
        s_pat = []
        k = 1
        c = 1
        if len(word) != len(pattern):
            return False
        for i in range(len(word)):
            if word[i] in st_occ:
                s_pat.append(st_occ[word[i]])
            if word[i] not in st_occ:
                st_occ[word[i]] = c
                c += 1
                s_pat.append(st_occ[word[i]])

        for i in range(len(pattern)):
            if pattern[i] in pat_occ:
                p_pat.append(pat_occ[pattern[i]])

            if pattern[i] not in pat_occ:
                pat_occ[pattern[i]] = k
                k += 1
                p_pat.append(pat_occ[pattern[i]])
        for i in range(len(s_pat)):
            if s_pat[i] != p_pat[i]:
                return False
        return True
```


Yash_Gupta202
Mar 06, 2023 15:30

Python3

Runtime 31 ms Beats 71.78% Memory 14 MB Beats 13.36%

Click the distribution chart to view more details

Notes

Write your notes here

Related Tags

Select tags 0/5

```
class Solution:
    def wordPattern(self, pattern: str, s: str) -> bool:
        word = s.split(" ")
        pat_occ = {}
        st_occ = {}
        p_pat = []
        s_pat = []
        k = 1
        c = 1
        if len(word) != len(pattern):
            return False
        for i in range(len(word)):
            if word[i] in st_occ:
                s_pat.append(st_occ[word[i]])
            if word[i] not in st_occ:
                st_occ[word[i]] = c
                c += 1
                s_pat.append(st_occ[word[i]])

        for i in range(len(pattern)):
            if pattern[i] in pat_occ:
                p_pat.append(pat_occ[pattern[i]])
```

```
class Solution:
    def wordPattern(self, pattern: str, s: str) -> bool:
        word = s.split(" ")
        pat_occ = {}
        st_occ = {}
        p_pat = []
        s_pat = []
        k = 1
        c = 1
        if len(word) != len(pattern):
            return False
        for i in range(len(word)):
            if word[i] in st_occ:
                s_pat.append(st_occ[word[i]])
            if word[i] not in st_occ:
                st_occ[word[i]] = c
                c += 1
                s_pat.append(st_occ[word[i]])

        for i in range(len(pattern)):
            if pattern[i] in pat_occ:
                p_pat.append(pat_occ[pattern[i]])

            if pattern[i] not in pat_occ:
                pat_occ[pattern[i]] = k
                k += 1
                p_pat.append(pat_occ[pattern[i]])
        for i in range(len(s_pat)):
            if s_pat[i] != p_pat[i]:
                return False
        return True
```