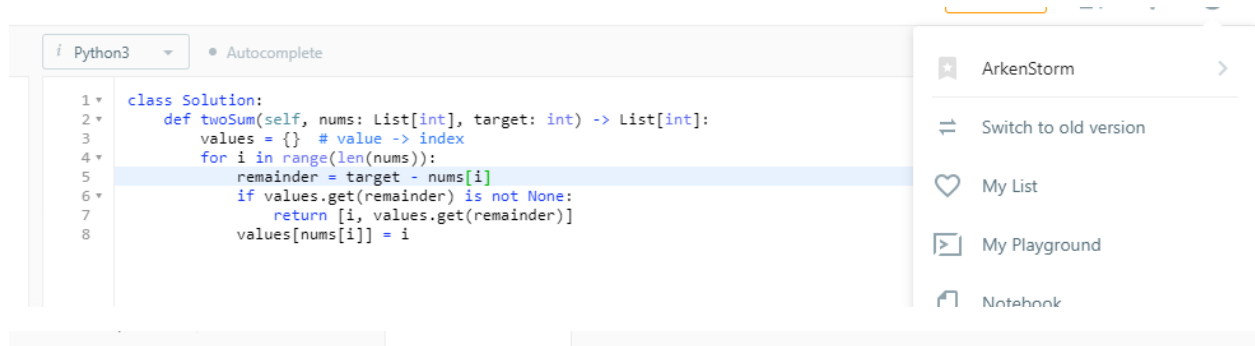


Two Sum:



```
1 class Solution:
2     def twoSum(self, nums: List[int], target: int) -> List[int]:
3         values = {} # value -> index
4         for i in range(len(nums)):
5             remainder = target - nums[i]
6             if values.get(remainder) is not None:
7                 return [i, values.get(remainder)]
8             values[nums[i]] = i
```

Success Details >

Runtime: 40 ms, faster than 97.91% of Python3 online submissions for Two Sum.

Memory Usage: 15.2 MB, less than 5.11% of Python3 online submissions for Two Sum.

Next challenges:

I discussed the problem with Michael Sederberg, Nicole Curtis, and Cara Johnson

Container with Most Water:



```
1 class Solution:
2     def maxArea(self, height: List[int]) -> int:
3         maximized = False
4         left = 0
5         right = len(height) - 1
6         best_area = (right - left) * min(height[left], height[right])
7         while left < right - 1:
8             if height[left] < height[right]:
9                 left += 1
10            else:
11                right -= 1
12            best_area = max(best_area, (right - left) * min(height[left], height[right]))
13        return best_area
```

Success Details >

Runtime: 132 ms, faster than 60.14% of Python3 online submissions for Container With Most Water.

Memory Usage: 15.3 MB, less than 5.26% of Python3 online submissions for Container With Most Water.

Next challenges:

Trapping Rain Water

I discussed the problem with Michael Sederberg, Nicole Curtis, and Cara Johnson

Combination Sum:

30-Day LeetCode Challenge! ✕

Premium + 🔔 👤

Python3 ▼ • Autocomplete

```
1 import heapq
2
3 class Solution:
4     def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
5         sol_set = set()
6         # state = (length, curr_sol, sum)
7         prio_queue = []
8         for candidate in candidates:
9             curr_sol = [candidate]
10            heapq.heappush(prio_queue, (len(curr_sol), candidate, curr_sol))
11            while len(prio_queue) > 0:
12                curr_state = heapq.heappop(prio_queue)
13                if curr_state[1] == target:
14                    sol_set.add(tuple(sorted(curr_state[2])))
15                if (target - curr_state[1]) in candidates:
16                    sol = curr_state[2].copy()
17                    sol.append(target - curr_state[1])
18                    sol_set.add(tuple(sorted(sol)))
19                for val in candidates:
20                    if curr_state[1] + val < target:
21                        sol = curr_state[2].copy()
22                        sol.append(val)
23                        heapq.heappush(prio_queue, (len(sol), curr_state[1] + val, sol))
24
25            return sol_set
```

ArkenStorm >

⇄ Switch to old version

♥ My List

📁 My Playground

📄 Notebook

🕒 Submissions

📊 Sessions

🕒 Progress

📅 Points

Description △ Solution 🕒 Submissions 💬 Discuss (999+)

Success Details >

Runtime: 408 ms, faster than 6.73% of Python3 online submissions for Combination Sum.

Memory Usage: 14.6 MB, less than 6.06% of Python3 online submissions for Combination Sum.

Next challenges:

I discussed this one with Ryan Echols.

Triangle:

Python3Autocomplete

```
1 class Solution:
2     def minimumTotal(self, triangle: List[List[int]]) -> int:
3         layer_sums = [triangle[0][0]]
4         if len(triangle) == 1:
5             return triangle[0][0]
6         for i in range(1, len(triangle)):
7             curr_sums = layer_sums
8             layer_sums = []
9             for j in range(len(triangle[i])):
10                if j == 0:
11                    layer_sums.append(curr_sums[j] + triangle[i][j])
12                elif j == len(triangle[i]) - 1:
13                    layer_sums.append(curr_sums[j - 1] + triangle[i][j])
14                else:
15                    left = curr_sums[j - 1] + triangle[i][j]
16                    right = curr_sums[j] + triangle[i][j]
17                    layer_sums.append(min(left, right))
18            return min(layer_sums)
```

ArkenStorm

Switch to old version

My List

My Playground

Notebook

Submissions

Sessions

Success Details >

Runtime: 60 ms, faster than 68.14% of Python3 online submissions for Triangle.

Memory Usage: 14.5 MB, less than 6.67% of Python3 online submissions for Triangle.

Next challenges:

I discussed this one with Nicole Curtis

Friend Circles:

Python3Autocomplete

```
1 class Solution:
2     def findCircleNum(self, M: List[List[int]]) -> int:
3         visited = set()
4         groups = 0
5         while len(visited) < len(M):
6             for i in range(len(M)):
7                 if not i in visited:
8                     visited.add(i)
9                     self.dfs(i, visited, M)
10                    groups += 1
11            return groups
12
13     def dfs(self, friend, visited, M):
14         for i in range(len(M[friend])):
15             if M[friend][i] and not i in visited:
16                 visited.add(i)
17                 self.dfs(i, visited, M)
```

ArkenStorm

Switch to old version

My List

My Playground

Notebook

Submissions

Sessions

Success Details >

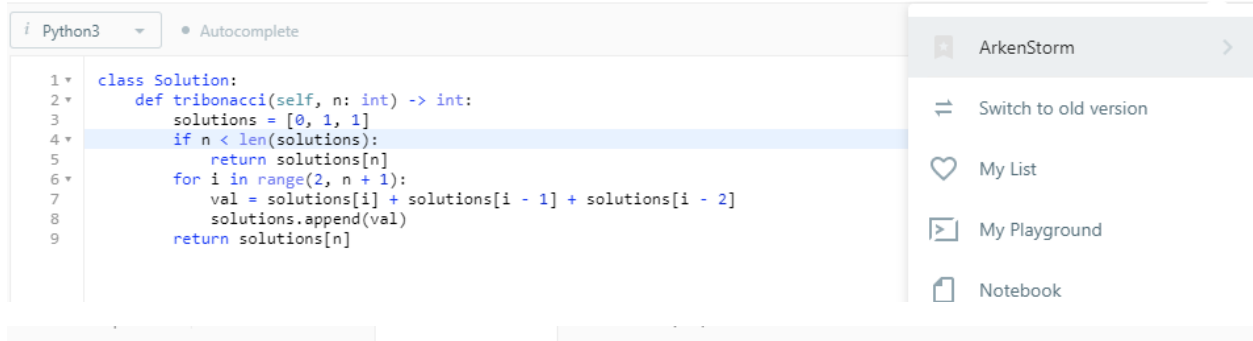
Runtime: 188 ms, faster than 95.04% of Python3 online submissions for Friend Circles.

Memory Usage: 14.2 MB, less than 5.88% of Python3 online submissions for Friend Circles.

Next challenges:

I discussed this one with Nicole Curtis and Ryan Echols

N-th Tribonacci:



```
1 class Solution:
2     def tribonacci(self, n: int) -> int:
3         solutions = [0, 1, 1]
4         if n < len(solutions):
5             return solutions[n]
6         for i in range(2, n + 1):
7             val = solutions[i] + solutions[i - 1] + solutions[i - 2]
8             solutions.append(val)
9         return solutions[n]
```

ArkenStorm

- Switch to old version
- My List
- My Playground
- Notebook

Success Details >

Runtime: 28 ms, faster than 63.00% of Python3 online submissions for N-th Tribonacci Number.

Memory Usage: 13.7 MB, less than 100.00% of Python3 online submissions for N-th Tribonacci Number.

Next challenges:

I discussed this one with Nicole Curtis and Ryan Echols

For most of them I found that they had done similar methods to my initial approaches, but I wanted to improve and challenge myself to do better algorithms. When I was discussing with Ryan, he had a bunch of linear algebra solutions that were really cool, but they hadn't even crossed my mind. So it was really interesting to see the linear algebra theory behind the solutions that he came up with.