Two Sum

Given array of ints, and target int, return indices of two ints so they sum to target

Practice

[1,4,8,6], target = 12, return [1,2]

12
1 + 11

[2,4,3], target = 5, return [0,2]

[-1,0], target = -1, return [0,1]

Algorithm

1. If length of array = 2, return [0,1]
2. Start from index 0, calculate the sum of each possible pair until sum equals target value

```
def twoSum(intArr[], target):                    O(n^2)
    targetArray = [0,1]
    if len(intArr == 2):
        return targetArray
    for i in range(0, len(intArr)):
        for j in range(i+1, len(intArr)):
            if intArr[i] + intArr[j] == target:
                targetArray[0] = i
                targetArray[1] = j
                return targetArray
```

Two Sum (continued)

[1,4,8,6]
target = 12

Algorithm

1. If length = 2, return [0,1]
2. Check each term
   ↳ if the difference between the target and that term exists, return the index of both terms

```
def twoSum (intArr[], target)
    targetArr = [0,1]
    if len(intArr) == 2:
        return targetArr
    for i in range (0, len(intArr)):
        diff = target - intArr[i]
        if diff in intArr:
            first = intArr[i]
            break
    for i in range (first+1, len(intArr)):
        if intArr[i] = diff:
            second = intArr[i]
            targetArr[0] = first
            targetArr[1] = second
            return targetArr
```

1

O(n)

diff = 0
first = 0

Two Sum (Solution Analysis)

- Proposed solution was close but missing a couple steps
- Syntactically, remember that Python types are dynamic, so the function header should look like;
  - ↳ def twoSum(self, intArr, target):
- There is a method in Python, .index, which returns the index of the first occurence of the value
  - ↳ I couldn't remember that when solving problem

Cleaned Up O(n) Solution:

```
def twoSum(self, intArr, target):
    targetArr = [0,1]  #array of indices
    arrLength = len(intArr) #length of array

    if (len(intArr) == 2): #if length=2, indices will
        return targetArr    #always be 0 and 1

    For i in range (0, arrLength): #iterate through array of ever value
        diff = target - intArr[i]  #set diff to complement
        if diff in intArr:         #check if complement exists,
            j = intArr.index(diff) if so set value to j
            if j != i:  #no element can be used twice, catches
    #set correct values  targetArr[0] = i # [3,2,3] case
    #at correct indices  targetArr[i] = j
                        return targetArr # return correct indices
```

Two Sum (Optimal Solution)

```
def twoSum(self, nums, target):
    mapped_nums = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in mapped_nums:
            return [mapped_nums[complement], i]
        mapped_nums[num] = i
```

Notes:

- O(n) time complexity because looking up values in a hash map is a constant time operation
- O(n) memory because hash map takes up memory
- enumerate is a tuple that stores the index in i and the current value in nums
  - ↳ Ex: [2, 5, 6, 19], target = 21
    - i = 0, num = 2
    - i = 1, num = 5
    - i = 2, num = 6
    - i = 3, num = 19
- This program uses hashmaps (same as dictionaries) such that the num is the key and the index is the value
- If complement is in the hashmap, the program searches for index w/ value