1. Given a list of integers, write a Python function which returns indices of the two numbers such that they add up to a specific target. Explain your logic/code.

Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9, return [0, 1].

In [1]:
```
nums = [2, 7, 11, 15]
target = 9
def two_numbers(nums,target):
# We will iterate through the nums list
    for num in nums:
# We will get the difference between the target and the specific item picked up in this iteration
        diff = target- num
# if the difference is an element of the nums list, then we will return the index of item and
# the index of difference from the array as a list format
        try:
            if nums.index(diff):
                return ([nums.index(num), nums.index(diff)])
                break
# in case the difference is not an element, we want our loop to continue to the next iteration
        except:
            continue
two_numbers(nums,target)
```

Out[1]:
```
[0, 1]
```

2. Without using swapcase(), You are given a string and your task is to swap cases. In other words, convert all lowercase letters to uppercase letters and vice versa. Explain your logic/code.

Input: McDonald’s

Output: mCdONALD’S

In [2]:
```
input_string = "McDonald’s"
def case_swap(input_string):
    # variable with empty string which will be modified through the for loop below
    output_string = ''
    # looping through the string to to check whether a letter is lower cased or not
    for letter in input_string:
        if letter.islower():
    # convert lower cased letter to upper cased and add it to the new variable
            output_string += letter.upper()
        else:
    # convert upper cased letter to lower cased and add it to the new variable
            output_string += letter.lower()
    # finally return the output string
    return output_string
case_swap(input_string)
```

Out[2]:
```
'mCdONALD’S'
```

3. Given a string and a string list, write a Python program to remove the string from the list and return the modified list. Explain your logic/code.

Input List: [‘You’,'cannot','end','a','sentence','with','because','Because','because','is','a','conjunction.']

Input String : ‘because’

Output: ['You', 'cannot', 'end', 'a', 'sentence', 'with', 'Because', 'is', 'a', ‘conjunction.']

In [3]:
```
input_list = ['You','cannot','end','a','sentence','with','because','Because',\
              'because','is','a','conjunction.']
input_string = 'because'
# We will use the count method ot get the frequency of the input string as an element of input list
# and apply the remove method on the input list that many times
def modify_list(input_list,input_string):
    for count in range(input_list.count(input_string)):
        input_list.remove(input_string)
# finally return the modified list
    return input_list
modify_list(input_list,input_string)
```

Out[3]:
```
['You',
 'cannot',
 'end',
 'a',
 'sentence',
 'with',
 'Because',
 'is',
 'a',
 'conjunction.']
```

4. Without using ‘import textwrap’

Read a string and a width, wrap the string into a paragraph of width. Explain your logic/code.

Input :

ABCDEFGHIJKLIMNOQRSTUVWXYZ 4

Output :

ABCD

EFGH

IJKL

IMNO

QRST

UVWX

YZ

In [4]:
```
input_string = 'ABCDEFGHIJKLIMNOQRSTUVWXYZ'
width = 4
def string_break(input_string, width):
    # let us create an empty string variable to store the modified string
    result = ''
    # we will loop the input string with an incrementation of steps equal to the width
    # and add a linebreak to the modified string
    for letter in range(0,len(input_string),width):
    # We want to avoid adding a linebreak at the end of the modified string
    # which is why we are comparing whether we are at last batch of iteration
        if len(input_string) - letter >4:
    # Add the combination of letters with a linebreak to the new variable
            result+=input_string[letter:letter+width]+"\n"
        else:
    # Add the combination of letters without a linebreak to the new variable
            result+=input_string[letter:letter+width]
    # print the result
    print(result)
string_break(input_string, width)
```

```
ABCD
EFGH
IJKL
IMNO
QRST
UVWX
YZ
```

5. Without using ‘from collections import Counter’

Write a Python program to combine values in a list of dictionaries. Explain your logic/code.

Input :

[{‘item’: ‘item1’, ‘amount’: 400},{‘item’: ‘item2’, ‘amount’: 300},{‘item’: ‘item1’, ‘amount’: 750}]

Output :

{‘item1’: 1150, ‘item2’: 300}

In [5]:
```
input_list = [{'item': 'item1', 'amount': 400},{'item': 'item2', 'amount': 300},\
              {'item': 'item1', 'amount': 750}]
# Creating a function that would work as counter for dictionary objects
def combined_output(item_list):
# create an empty dictionary
    combined_output = {}
# iterate through the items in the input list
    for item in item_list:
# as each item in the input list is a dictionary
# we will use the dictionary's key to check whether the key exist in the new variable
        if item['item'] not in combined_output:
# the key does not exist which is why we add the key value pair to the new variable
            combined_output[item['item']] = item['amount']
        else:
            combined_output[item['item']] += item['amount']
# finally we return the new dictionary varible
    return combined_output
combined_output(input_list)
```

Out[5]:
```
{'item1': 1150, 'item2': 300}
```

6. Write a Python program to return the number of even ints in the given array. Explain your logic/code.

count_evens([2, 1, 2, 3, 4]) → 3

count_evens([2, 2, 0]) → 3

count_evens([1, 3, 5]) → 0

In [6]:
```
# Create a function to count the number of even integers in a given array
def count_evens(nums):
# Initiate a counter variable which keeps track of occurences of even number in an array
    counter = 0
# Iterate through the array to check if a number is even or divisible by 2
    for num in nums:
        if num %2 == 0:
# If a number is divisible by 2 then the remainder will be 0 which would indicate the number is even
            counter += 1
# Finally return the counter variable
    return counter
```

In [7]:
```
count_evens([2, 1, 2, 3, 4])
```

Out[7]:
```
3
```

In [8]:
```
count_evens([2, 2, 0])
```

Out[8]:
```
3
```

In [9]:
```
count_evens([1, 3, 5])
```

Out[9]:
```
0
```

7. Without using min() and max() functions, given an array length 1 or more of ints, write a Python program to return the difference between the largest and smallest values in the array. Explain your logic/code.

big_diff([10, 3, 5, 6]) → 7

big_diff([7, 2, 10, 9]) → 8

big_diff([2, 10, 7, 2]) → 8

In [10]:
```
# create a function to derive the difference between the minimun and maximum from a given array
def big_diff(nums):
# initialize 2 variables with the first element of the array
# they will treated as the minimum and maximum to start with
    minimum = maximum = nums[0]
# iterate through the array from the second element onwards
    for num in nums[1:]:
        if num < minimum:
# the number is less the value stored in the variable "minimum", so we override the variable
# with this number
            minimum = num
        elif num > maximum:
# the number is more the value stored in the variable "maximum", so we override the variable
# with this number
            maximum = num
        else:
# we continue on to the next iteration
            continue
# create a variable to hold the difference between maximum and minimum
    difference = maximum - minimum
# finally return the difference variable
    return difference
```

In [11]:
```
big_diff([10, 3, 5, 6])
```

Out[11]:
```
7
```

In [12]:
```
big_diff([7, 2, 10, 9])
```

Out[12]:
```
8
```

In [13]:
```
big_diff([2, 10, 7, 2])
```

Out[13]:
```
8
```

8. Write a Python program to return the number of times that the string "code" appears anywhere in the given string, except we'll accept any letter for the 'd', so "cope" and "cooe" count. Explain your logic/code.

count_code('aaacodebbb') → 1

count_code('codexxcode') → 2

count_code('cozexxcope') → 2

In [14]:
```
#  create a function that will return the number of occurences of a pattern in a given string
def count_code(input_string):
# import regular expression library
    import re
# as the first 2 letters of the 4- letter pattern is 'co' and the last letter is 'e'
# we can pass [a-z] argument as the 3rd letter so regex would know any letter on the 3rd
# position is accepted
# re.findall() returns a list of all occurences for pattern match
    occurences = re.findall('co'+'[a-z]'+'e', input_string)
# finally return the length of list of occurences
    return len(occurences)
```

In [15]:
```
count_code('aaacodebbb')
```

Out[15]:
```
1
```

In [16]:
```
count_code('codexxcode')
```

Out[16]:
```
2
```

In [17]:
```
count_code('cozexxcope')
```

Out[17]:
```
2
```

In [18]:
```
# Alternate solution
def count_code2(input_string):
# Initiate a counter variable
    counter = 0
# Create a list of unique possible patterns with the first 2 characters as'co', last (4th)
# character as 'e' and third character being any character from the input string
    patterns=list(set(["co"+char+"e" for char in input_string]))
# iterate through patterns list to find each element/ pattern in the input string
    for pattern in patterns:
        if pattern in input_string:
# macth found and we add the total occurence count of the pattern to the counter
            counter+=input_string.count(pattern)
# finally return the counter variable
    return counter
```

In [19]:
```
count_code2('aaacodebbb')
```

Out[19]:
```
1
```

In [20]:
```
count_code2('codexxcode')
```

Out[20]:
```
2
```

In [21]:
```
count_code2('cozexxcope')
```

Out[21]:
```
2
```

9. We want make a package of goal kilos of chocolate. We have small bars (1 kilo each) and big bars (5 kilos each). Write a Python program to return the number of small bars to use, assuming we always use big bars before small bars. Return -1 if it can't be done. Explain your logic/code.

Method Signature : make_chocolate(small, big, goal)

make_chocolate(4, 1, 9) → 4

make_chocolate(4, 1, 10) → -1

make_chocolate(4, 1, 7) → 2

In [22]:
```
def make_chocolate(small, big, goal):
# calculate the total available bar weight
    total_weight = (small*1) + (big*5)
    if goal <= total_weight:
# calculate the remaining weight after using all the big bars
        difference = goal - (big*5)
        if difference <= (small*1):
# the remainder weight can be covered with available small bars
# so we store the remainder value in a new variable
            small_needed = difference
        else:
# the remainder weight can NOT be covered with available small bars
# so we default the value of the new variable to -1
            small_needed = -1
    else:
# the goal weight can NOT be covered with available bars
# so we default the value of the new variable to -1
        small_needed = -1
# finally we return the new variable 'small_needed'
    return small_needed
```

In [23]:
```
make_chocolate(4, 1, 9)
```

Out[23]:
```
4
```

In [24]:
```
make_chocolate(4, 1, 10)
```

Out[24]:
```
-1
```

In [25]:
```
make_chocolate(4, 1, 7)
```

Out[25]:
```
2
```

10. Given 2 strings, a and b, Write a Python program to return the number of the positions where they contain the same length 2 substring. So "xxcaazz" and "xxbaaz" yields 3, since the "xx", "aa", and "az" substrings appear in the same place in both strings. Explain your logic/code.

string_match('xxcaazz', 'xxbaaz') → 3

string_match('abc', 'abc') → 2

string_match('abc', 'axc') → 0

In [26]:
```
def string_match(string1, string2):
# first we use list comprehension to create lists of 2 letter substrings from the strings
    substrings1 = [string1[i]+string1[i+1] for i in range(len(string1)-1)]
    substrings2 = [string2[i]+string2[i+1] for i in range(len(string2)-1)]
# initailize the counter
    counter = 0
# iterate through both lists after zipping them to access items from both lists at the
# same index or position at the same time and compare for match
    for sub1,sub2 in zip(substrings1,substrings2):
        if sub1 == sub2:
# we increment the counter value if values match
            counter += 1
# finally we return the counter variable
    return counter
```

In [27]:
```
string_match('xxcaazz', 'xxbaaz')
```

Out[27]:
```
3
```

In [28]:
```
string_match('abc', 'abc')
```

Out[28]:
```
2
```

In [29]:
```
string_match('abc', 'axc')
```

Out[29]:
```
0
```