



Write a function named as `solution` which takes a list of numbers as argument and returns sum of these numbers.

Here's an implementation of the solution function in Python:

pythonCopy code

```
def solution(numbers):  
    return sum(numbers)
```

This function takes a list of numbers as an argument and returns the sum of these numbers. The `sum` built-in function in Python calculates the sum of the elements in a list or iterable.



Write a function named as `solution` which takes a list of numbers as argument and returns sum of these numbers. Example 1: In: `solution([1, 2, 3])` Out: 6

Here's an implementation of the solution function in Python:

pythonCopy code

```
def solution(numbers):  
    return sum(numbers)
```

This function takes a list of numbers as an argument and returns the sum of these numbers. The `sum` built-in function in Python calculates the sum of the elements in a list or iterable.

Example:

pythonCopy code

```
>>> solution([1, 2, 3])6
```



Write a function named as `solution` which takes a list of integers as argument and returns the count of even numbers. Example 1 In: `solution([1, 4, 2, 3, 13, 4, 1])` Out: 3

Here's an implementation of the solution function in Python:

pythonCopy code

```
def solution(numbers):  
    count = 0  
    for num in numbers:  
        if num % 2 == 0:  
            count += 1  
    return count
```

This function takes a list of integers as an argument and returns the count of even numbers in the list. The % operator in Python is used to perform modulo division, which gives the remainder of a division. If the remainder is 0, the number is even. The function uses a for loop to iterate over the elements in the list and check if each number is even. If the number is even, the count is incremented. Finally, the count is returned.

Example:

pythonCopy code

```
>>> solution([1,4,2,3,13,4,1])3
```



Write a function named as solution which a string as argument and returns the number of letters and digits (list). Example 1 In: solution('hello3worl1d') Out: [10,2] Example 2 In: solution(' 2345ab') Out: [2,4]

Here's an implementation of the solution function in Python:

pythonCopy code

```
def solution(s):  
    letters = 0  
    digits = 0  
    for char in s:  
        if char.isdigit():  
            digits += 1  
        elif char.isalpha():  
            letters += 1  
    return [letters, digits]
```

This function takes a string as an argument and returns a list of two elements, the number of letters and the number of digits in the string. The function uses a for loop to iterate over the characters in the string and check if each character is a digit or a letter using the *isdigit* and *isalpha* string methods, respectively. If the character is a digit, the *digits* count is incremented, and if it's a letter, the *letters* count is incremented. Finally, a list of the *letters* and *digits* counts is returned.

Example:

pythonCopy code

```
>>> solution('hello3wor1ld')
[10, 2]>>> solution('2345ab')
[2, 4]
```



A website requires a password to register a user. Write a function *solution* which inputs a password (string) and validate it on following rules. Password should contain at least one letter from [a-z] Password should contain at least one letter from [A-Z] Password should contain at least one character from [@, \$, #] Password should contain at least one digit from [0-9] Password should have a minimum length of 6. The function will return True in case of valid password and False in case of in-valid password. Example 1 In: *solution('helloworld')* Out: False Example 2 In: *solution(' Abc\$123')* Out: True Example 3 In: *solution(' 12345678')* Out: False

Here's an implementation of the *solution* function in Python:

pythonCopy code

```
def solution(password):
    if len(password) < 6:
        return False
    has_lower = False
    has_upper = False
    has_special = False
    has_digit = False
    for char in password:
        if char.islower():
            has_lower = True
        elif char.isupper():
```

```

        has_upper = True
    elif char in ['@', '$', '#']:
        has_special = True
    elif char.isdigit():
        has_digit = True
    return has_lower and has_upper and has_special and has_digit

```

This function takes a string *password* as an argument and returns a boolean indicating whether the password is valid or not. The function first checks if the length of the password is less than 6 characters, and if it is, it returns False. It then initializes four variables *has_lower*, *has_upper*, *has_special*, and *has_digit* to False, which will be used to track whether the password meets the criteria or not. The function then uses a for loop to iterate over the characters in the password and check if each character is lowercase, uppercase, a special character, or a digit. If a character meets any of these criteria, the corresponding variable is set to True. Finally, the function returns True if all four variables are True, indicating that the password meets all the criteria, or False otherwise.

Example:

pythonCopy code

```

>>> solution('helloworld')False>>> solution('Abc$123')True>>>
solution('12345678')False

```



You are a advance cryptographers and you have to write a code for it. In the code, every lowercase letter [a..z] is replaced with the corresponding one in [z..a], while every other character (including uppercase letters and punctuation) is left untouched. That is, 'a' becomes 'z', 'b' becomes 'y', 'c' becomes 'x', etc. For instance, the word "vmxibkgrlm", when decoded, would become "encryption". Write a function called solution which takes in a string and returns the deciphered string. Example 1 In: solution("vmxibkgrlm") Out: "encryption" Example 2 In: solution("wrw blf hvv ozhg mrtsg'h vkrhlwv?") Out: did you see last night's episode?

Here's an implementation of the solution function in Python:

pythonCopy code

```

def solution(s):
    deciphered = ''
    for char in s:

```

```

    if char.islower():
        deciphered += chr(122 - ord(char) + 97)
    else:
        deciphered += char
return deciphered

```

This function takes a string *s* as an argument and returns the deciphered string. The function initializes an empty string *deciphered* which will be used to store the deciphered string. The function then uses a for loop to iterate over the characters in the input string. If a character is lowercase, the function converts it to the corresponding character in the opposite case (a -> z, b -> y, etc.) by using the *ord* and *chr* functions in Python. If the character is not lowercase, it is left unchanged. The deciphered string is constructed by appending each deciphered character to the *deciphered* string. Finally, the function returns the *deciphered* string.

Example:

pythonCopy code

```

>>> solution("vmxibkgrlm")"encryption">>> solution("wrw blf hvv ozhg
mrtsg'h vkrhlwv?")"did you see last night's episode?"

```



Write a function named *solution* which takes a list containing valid integers (2 to 10) and valid strings ('Jack', 'Queen', 'King' and 'Ace') with exact spellings as argument. This function should return the same list after sorting. Sorting precedence: 2, 3, 4, 5, 6, 7, 8, 9, 10, 'Jack', 'Queen', 'King', 'Ace' Example 1 In: *solution*(['Queen',2,'Jack',9])) Out: [2, 9, 'Jack', 'Queen'] Example 2 In: *solution*([2,3,5,4,'Queen','King',7,'Ace','Queen',9,'Jack','King']) Out: [2, 3, 4, 5, 7, 9, 'Jack','Queen', 'Queen', 'King', 'King', 'Ace']

Here's an implementation of the *solution* function in Python:

pythonCopy code

```

def solution(arr):
    sorted_arr = sorted(arr, key=lambda x: (x if isinstance(x, int) else 11
if x == 'Jack' else 12 if x == 'Queen' else 13 if x == 'King' else 14))
    return sorted_arr

```

This function takes a list *arr* as an argument and returns the sorted list. The function uses the built-in *sorted* function to sort the list, with a custom sorting key that determines the sorting precedence. The sorting key is defined using a lambda function that takes an element *x* and returns a numerical value based on the type of *x*. If *x* is an integer, the function returns *x* itself. If *x* is the string 'Jack', the function returns 11. If *x* is the string 'Queen', the function returns 12. If *x* is the string 'King', the function returns 13. If *x* is the string 'Ace', the function returns 14. This results in the sorting precedence described in the problem statement. The function returns the sorted list *sorted_arr*.

Example:

pythonCopy code

```
>>> solution(['Queen', 2, 'Jack', 9])
[2, 9, 'Jack', 'Queen']>>> solution([2, 3, 5, 4, 'Queen', 'King', 7, 'Ace',
'Queen', 9, 'Jack', 'King'])
[2, 3, 4, 5, 7, 9, 'Jack', 'Queen', 'Queen', 'King', 'King', 'Ace']
```

MR

Write a function named as *solution* which takes a list containing strings with versions numbers separated by two dots as argument. This function should return the versions after sorting. Example 1 In: *solution*(['1.2.1' , '1.0.9' , '0.7.6' , '7.4.0' , '2.1.3']) Out: ['0.7.6' , '1.0.9' , '1.2.1' , '2.1.3' , '7.4.0'] Example 2 In: *solution*(['0.0.3' , '0.0.1' , '3.3.0' , '2.1.6']) Out: ['0.0.1' , '0.0.3' , '2.1.6' , '3.3.0']

Here is a function that implements the solution as described:

pythonCopy code

```
def solution(versions):
    def version_to_tuple(version):
        return tuple(map(int, version.split(".")))
    return sorted(versions, key=version_to_tuple)
```

The function *version_to_tuple* is used to convert each version string into a tuple of integers, which can then be sorted using the standard *sorted* function.

MR

In Excel the column is labelled with alphabets A, B, C and so on. After Z the next column is assigned AA, AB, AC, ..., ZY, ZZ, AAA, AAB and so on. Write a function named as solution which takes an integer as argument and returns its corresponding column label. Example 1 In: solution(1) Out: ' A' Example 2 In: solution(26) Out: ' Z' Example 3 In: solution(27) Out: ' AA' Example 4 In: solution(5000) Out: ' BUYB' Example 5 In: solution(0) Out: ' Invalid column number'

pythonCopy code

```
def solution(num):
    if num <= 0:
        return "Invalid column number"

    column = ""
    while num > 0:
        num -= 1
        column = chr(num % 26 + ord('A')) + column
        num = num // 26

    return column
```



Write the function named as solution for the following sample input and output. Example 1 In: solution(["P>E", "E>R", "R>U"]) Out: PERU Example 2 In: solution(["I>N", "A>I", "P>A", "S>P"]) Out: SPAIN Example 3 In: solution(["U>N", "G>A", "R>Y", "H>U", "N>G", "A>R"]) Out: HUNGARY Example 4 In: solution(["I>F", "W>I", "S>W", "F>T"]) Out: SWIFT Example 5 In: solution(["R>T", "A>L", "P>O", "O>R", "G>A", "T>U", "U>G"]) Out: PORTUGAL

Here's the code for the function *solution* that solves the problem statement:

scssCopy code

```
def solution(arr):
    graph = {}
    for edge in arr:
        src, dst = edge.split('>')
        if src not in graph:
            graph[src] = []
        graph[src].append(dst)
```

```

visited = set()
result = []
for node in graph:
    if node not in visited:
        dfs(node, graph, visited, result)
return ''.join(result[::-1])

```

```

def dfs(node, graph, visited, result):
    if node not in visited:
        visited.add(node)
    if node in graph:
        for neighbor in graph[node]:
            dfs(neighbor, graph, visited, result)
    result.append(node)

```

Explanation:

- The code creates a directed graph representation of the input array, where the elements in the array are the edges of the graph, and the first character in each edge is the source node and the second character is the destination node.
- The code performs a depth-first search (dfs) on the graph, starting from each node, and appends the visited nodes to the *result* list.
- The code returns the reverse of the *result* list as the final answer.



In a prison cells are stacked in triangular shape, and cells are give ids staring from the corner as follow. | 7 | 4 8 | 2 5 9 | 1 3 6 10
Each cell can represent as points [x, y], with x being distance from the vertical wall, and y being the height from the ground. For Example the cell at [1, 1] has ID 1, the cell at [3, 2] has ID 9, and the cell at [2, 3] has ID 8. This pattern of numbering continues indefinitely (as more cells are being added). Write function solution([x, y]) which returns the cell ID at location [x, y]. Each value of x and y will be at least 1 and no greater than 100,000. As the cell ID may be very large number, return it in form of string.
Example 1 In: solution([5,10]) Out: '96' Example 2 In: solution([3,2]) Out: '9'

Here is the implementation of the solution function:

pythonCopy code

```
def solution(coord):  
    x, y = coord  
    id = (x*(x-1))//2 + y  
    return str(id)
```



In a prison cells are stacked in triangular shape, and cells are give ids staring from the corner as follow. | 7 | 4 8 | 2 5 9 | 1 3 6 10
Each cell can represent as points $[x, y]$, with x being distance from the vertical wall, and y being the height from the ground. For Example the cell at $[1, 1]$ has ID 1, the cell at $[3, 2]$ has ID 9, and the cell at $[2, 3]$ has ID 8. This pattern of numbering continues indefinitely (as more cells are being added). Write function `solution([x, y])` which returns the cell ID at location $[x, y]$. Each value of x and y will be at least 1 and no greater than 100,000. As the cell ID may be very large number, return it in form of string. Example 1 In: `solution([5,10])` Out: '96' Example 2 In: `solution([3,2])` Out: '9'

Here is the implementation of the function "solution" that takes a list of length 2 representing the cell location $[x, y]$ as input and returns the cell ID in the form of a string:

pythonCopy code

```
def solution(location):  
    x, y = location[0], location[1]  
    cell_id = x * (x - 1) // 2 + y  
    return str(cell_id)
```

The function first unpack the values of x and y from the input location list. Then it calculates the cell ID using the formula $x * (x - 1) // 2 + y$. Finally, it returns the cell ID in the form of a string.



Write function `solution([x, y])` which returns the cell ID at location $[x, y]$. Each value of x and y will be at least 1 and no greater than 100,000. As the cell ID may be very large number, return it in form

of string. Example 1 In: `solution([5,10])` Out: '96' Example 2 In: `solution([3,2])` Out: '9'

Here's one way to implement the solution in Python:

pythonCopy code

```
def solution(coord):
    x, y = coord
    if x > y:
        return "Invalid cell location"
    cell_id = (x * (x - 1)) // 2 + y
    return str(cell_id)
```

This implementation uses a formula to calculate the cell ID based on the given coordinates x and y . The formula is $cell_id = (x * (x - 1)) // 2 + y$, where $//$ is the floor division operator. This formula works by summing up the number of cells in all the rows above the current one ($x * (x - 1) // 2$) and adding the position of the cell in the current row (y).

The function first checks if the given x is greater than y . If so, the cell location is invalid and the function returns "Invalid cell location". Otherwise, it calculates the cell ID using the formula and returns it as a string.



In a prison cells are stacked in triangular shape, and cells are give ids staring from the corner as follow. | 7 | 4 8 | 2 5 9 | 1 3 6 10
Each cell can represent as points $[x, y]$, with x being distance from the vertical wall, and y being the height from the ground. For Example the cell at $[1, 1]$ has ID 1, the cell at $[3, 2]$ has ID 9, and the cell at $[2, 3]$ has ID 8. This pattern of numbering continues indefinitely (as more cells are being added). Write function `solution([x, y])` which returns the cell ID at location $[x, y]$. Each value of x and y will be at least 1 and no greater than 100,000. As the cell ID may be very large number, return it in form of string. Example 1 In: `solution([5,10])` Out: '96' Example 2 In: `solution([3,2])` Out: '9'

Here is a python implementation of the solution function:

pythonCopy code

```
def solution(coord):
```

```
x, y = coord
id = (x * (x - 1)) // 2 + y
return str(id)
```

This function takes a coordinate *coord* represented as a list $[x, y]$, where x is the distance from the vertical wall and y is the height from the ground. The function calculates the cell ID using the formula $id = (x * (x - 1)) // 2 + y$, and then returns the result as a string.

Regenerate response

[ChatGPT Feb 13 Version](#). Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

New chat

Copy JPEGs with matching label.

Copy Matching Image Files.

Zip Lists to Dict

New chat

Anaconda Version Comparison Linux/Windows

Prison Cell ID Function

Summing List of Numbers

AI Response to "How Are You"

LinkedIn Profile Data Science Description

Build Portfolio Website Python/Django

Ans IQ Qs Possible

Central Features of a Group

Implementing empty methods in class.

Convolution of PDF/PMF

Convolution theorem for PDFs

Compiler Detects Infinite Loop

New chat

Clear conversationsUpgrade to PlusNEWLight mode[OpenAI DiscordUpdates](#)
[& FAQ](#)Log out

New chat