11.寫一個函數 word_count，接受一個字串作為參數，並返回該字串中單詞的數量。例如：word_count("Hello world") 應返回 2。

```
def word_count(s):
    # 使用 split() 方法將字串分割成單詞，並返回單詞的數量
    return len(s.split())

# 測試範例
result = word_count("Hello world")
print(result)  # 輸出應該是 2
```

解釋：

1. `split()` 方法會將字串 `s` 按空格分割成單詞的列表。
2. 使用 `len()` 函數計算該列表中元素的數量，即為字串中的單詞數。

這樣，`word_count("Hello world")` 會返回 `2`，因為字串中有兩個單詞「Hello」和「world」。

```
def word_count(text):
    # 使用 split() 方法將字串分割成單詞並計算長度
    words = text.split()
    return len(words)

# 讓用戶輸入一個字串
input_text = input("請輸入一個字串: ")
print(f"該字串中的單詞數量是: {word_count(input_text)}")
```

說明：

- `split()` 方法會將字串根據空格（默認情況）分割成單詞，並且會自動處理多個連續空格的情況。
- 函數 `word_count` 返回的是字串分割後的單詞數量，即列表的長度。

範例輸入/輸出：

如果用戶輸入：

```
makefile
請輸入一個字串: Hello world
```

輸出會是：

```
makefile
該字串中的單詞數量是: 2
```

12.寫一個函數 is_palindrome，接受一個字串作為參數，並返回該字串是否為迴文（正反讀都相同）。例如：is_palindrome("madam") 應返回 True。

```
def is_palindrome(s):
    # 去除字串中的空格並將字串轉為小寫，然後檢查正反是否相同
    s = s.replace(" ", "").lower()
    return s == s[::-1]

# 測試範例
result = is_palindrome("madam")
print(result)  # 輸出應該是 True
```

解釋：

1. `replace(" ", "")` 會去除字串中的空格，這樣如果字串中有空格也能正確檢查迴文。
2. `lower()` 方法將字串轉為小寫，這樣無論字母大小寫，都能正確比較。
3. `s[::-1]` 是 Python 中反轉字串的簡便方法。
4. 如果原始字串和反轉字串相同，則該字串為迴文，返回 `True`，否則返回 `False`。

範例 `is_palindrome("madam")` 會返回 `True`，因為正反讀都相同。

```
def is_palindrome(s):
    # 檢查字串是否等於其反轉
    return s == s[::-1]

# 讓使用者輸入字串
input_string = input("請輸入字串來檢查是否為迴文: ")

# 呼叫 is_palindrome 函數並顯示結果
if is_palindrome(input_string):
    print(f"'{input_string}' 是迴文！")
else:
    print(f"'{input_string}' 不是迴文。")
```

說明：

1. `is_palindrome` 函數通過比較字串 `s` 與它反轉後的字串（`s[::-1]`）來判斷是否為迴文。
2. 使用 Python 的切片語法 `[::-1]` 來反轉字串。
3. `input()` 函數讓使用者輸入字串，並將結果傳遞給 `is_palindrome` 函數，然後根據返回值來輸出是否是迴文。

範例：

若你輸入字串 `"madam"`，輸出如下：

```
arduino                                                    複製程式碼

請輸入字串來檢查是否為迴文: madam
'madam' 是迴文！
```

13. 寫一個函數 split_string，接受一個字串和一個分隔符作為參數，並返回一個包含切割後子字串的清單。例如：split_string("hello world", " ") 應返回 ['hello', 'world']。

```
def split_string(s, delimiter):
    # 使用 split() 方法根據分隔符將字串切割成子字串清單
    return s.split(delimiter)

# 測試範例
result = split_string("hello world", " ")
print(result)  # 輸出應該是 ['hello', 'world']
```

解釋：

1. `split(delimiter)` 方法會根據傳入的分隔符（`delimiter`）將字串 `s` 切割成一個子字串的清單。
2. 範例中，`split_string("hello world", " ")` 會將字串 `"hello world"` 以空格（`" "`）為分隔符進行切割，並返回 `['hello', 'world']`。

您可以使用不同的分隔符來切割字串，例如逗號（`,`)或其他字符。

```
def split_string(input_string, delimiter):
    # 使用 str.split() 方法來分割字串
    return input_string.split(delimiter)

# 讓使用者輸入字串和分隔符
input_string = input("請輸入字串: ")
delimiter = input("請輸入分隔符: ")

# 執行 split_string 函數並顯示結果
result = split_string(input_string, delimiter)
print(f"切割後的結果是: {result}")
```

說明：

1. `split_string` 函數使用了 Python 字串方法 `split()`，它會將傳入的字串根據指定的分隔符進行分割，並返回一個包含子字串的清單。
2. `input()` 函數讓使用者輸入字串和分隔符。注意：分隔符會是用戶輸入的任何字符（例如空格、逗號等）。
3. 最後，打印出切割後的結果。

範例：

若使用者輸入 `hello world` 為字串，並輸入空格作為分隔符，輸出結果如下：

```less
請輸入字串: hello world
請輸入分隔符:
切割後的結果是: ['hello', 'world']
```

14. 費氏數列：寫一個函數 fibonacci，接受一個整數 n，並返回長度為 n 的費氏數列。例如：fibonacci(5) 應返回 [0, 1, 1, 2, 3]。

```
def fibonacci(n):
    # 如果 n 是 0 或 1，直接返回對應的費氏數列
    if n <= 0:
        return []
    elif n == 1:
        return [0]

    # 初始化費氏數列的前兩項
    fib_sequence = [0, 1]

    # 根據 n 的值生成後續的費氏數列項目
    for i in range(2, n):
        next_value = fib_sequence[-1] + fib_sequence[-2]
        fib_sequence.append(next_value)

    return fib_sequence

# 測試範例
result = fibonacci(5)
print(result)  # 輸出應該是 [0, 1, 1, 2, 3]
```

解釋：

1. 如果 `n <= 0`，返回空列表，表示沒有數列項目。
2. 如果 `n == 1`，返回包含單一元素 `[0]` 的數列。
3. 使用一個初始列表 `fib_sequence = [0, 1]` 來存儲費氏數列的前兩項（0 和 1）。
4. 使用 `for` 迴圈從第 3 項開始計算費氏數列的後續項目，每一項都是前兩項的和。這是通過 `fib_sequence[-1] + fib_sequence[-2]` 來實現的，並將結果添加到列表中。
5. 最後，返回生成的費氏數列。

範例 `fibonacci(5)` 會返回 `[0, 1, 1, 2, 3]`，這是前 5 項費氏數列。

```
def fibonacci(n):
    # 初始的費氏數列
    sequence = [0, 1]

    # 若 n 小於 2，直接返回前 n 項
    if n == 0:
        return []
    elif n == 1:
        return [0]

    # 產生費氏數列，直到長度為 n
    for i in range(2, n):
        sequence.append(sequence[-1] + sequence[-2])

    return sequence

# 讓使用者輸入數字
n = int(input("請輸入一個整數來計算費氏數列的長度: "))
print(f"長度為 {n} 的費氏數列是: {fibonacci(n)}")
```

說明：

1. `fibonacci` 函數用來計算費氏數列，起始數列 `[0, 1]`。
2. 當 `n` 小於 2 時，根據 `n` 的值直接返回前幾項。
3. 否則，利用一個 `for` 循環來計算後續的費氏數列，直到達到長度 `n`。
4. 使用 `input()` 讓使用者輸入 `n`，並顯示對應的費氏數列。

15. 階乘計算： 寫一個函數 factorial，接受一個整數 n，並返回 n 的階乘。例如：factorial(5) 應返回 120。

```
def factorial(n):
    # 如果 n 是 0 或 1，階乘是 1
    if n == 0 or n == 1:
        return 1
    # 否則使用遞迴來計算階乘
    return n * factorial(n - 1)

# 測試範例
result = factorial(5)
print(result)  # 輸出應該是 120
```

解釋：

1. 階乘的定義是：`n! = n * (n-1) * (n-2) * ... * 1`，並且有一個特殊情況：`0! = 1` 和 `1! = 1`。
2. 在這裡，我們使用遞迴來計算階乘。`factorial(n)` 會返回 `n * factorial(n - 1)`，直到 `n` 減少到 1 或 0 為止。
3. 如果 `n == 0` 或 `n == 1`，我們直接返回 1，這是階乘的基礎情況。

範例 `factorial(5)` 會返回 `120`，因為 `5! = 5 * 4 * 3 * 2 * 1 = 120`。

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        result = 1
        for i in range(2, n + 1):
            result *= i
        return result

# 讓使用者輸入數字
n = int(input("請輸入一個整數來計算階乘: "))
print(f"{n} 的階乘是 {factorial(n)}")
```

說明：

1. `factorial` 函數用來計算階乘，當 `n` 為 0 或 1 時，返回 1，因為 0! = 1 和 1! = 1。
2. 否則，使用 `for` 循環來計算從 2 到 `n` 的數字相乘，最終得到階乘結果。
3. `input()` 函數用來讓使用者輸入數字，並轉換為整數（`int`），然後傳入 `factorial` 函數來計算並顯示結果。

16. 最小公倍數： 寫一個函數 lcm，接受兩個整數 a 和 b，並返回它們的最小公倍數。例如：lcm(6, 8) 應返回 24。

要解決這個問題，首先我們需要知道最小公倍數 (LCM) 和最大公因數 (GCD) 之間的關係。最小公倍數可以通過以下公式來計算：

$$\text{LCM}(a,b) = \frac{|a \times b|}{\text{GCD}(a,b)}$$

其中，GCD 是兩個數字的最大公因數。可以使用 Python 中的內建 `math.gcd` 函數來計算最大公因數。

```
import math

def lcm(a, b):
    return abs(a * b) // math.gcd(a, b)

# 測試範例
print(lcm(6, 8))  # 輸出 24
```

解釋：

1. `math.gcd(a, b)` 計算 a 和 b 的最大公因數。
2. 使用公式 $\text{LCM}(a,b) = \frac{|a \times b|}{\text{GCD}(a,b)}$ 來計算最小公倍數。
3. `abs(a * b)` 確保結果是正數，然後用 `//` 進行整數除法。

這樣就能計算出兩個數字的最小公倍數了。

```
import math


def lcm(a, b):
    # 計算最小公倍數：LCM(a, b) = |a * b| / GCD(a, b)
    return abs(a * b) // math.gcd(a, b)


# 讓用戶輸入兩個整數
a = int(input("請輸入第一個整數 a："))
b = int(input("請輸入第二個整數 b："))


# 計算並顯示結果
result = lcm(a, b)
print(f"{a} 和 {b} 的最小公倍數是 {result}")
```

解釋：

1. `math.gcd(a, b)` 用來計算 a 和 b 的最大公因數。
2. `abs(a * b)` 是為了保證乘積為正數（對於正數不影響）。
3. `//` 是整數除法運算符，用來確保結果為整數。

17. 最大公因數： 寫一個函數 gcd，接受兩個整數 a 和 b，並返回它們的最大公因數。例如：gcd(48, 18) 應返回 6。

要計算兩個整數的最大公因數 (GCD)，我們可以使用 **歐幾里得算法**。歐幾里得算法的基本思想是：對於兩個數字 a 和 b，最大公因數 GCD(a, b) 具有以下性質：

$$\mathrm{GCD}(a, b) = \mathrm{GCD}(b, a\%b)$$

其中 % 是取餘數操作。當 b 等於 0 時，GCD(a, 0) 就是 a。

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b   # 用餘數替換 b，直到 b 為 0
    return a

# 測試範例
print(gcd(48, 18))   # 輸出 6
```

解釋：

1. `while b != 0:` 不斷執行直到 b 變為 0。
2. `a, b = b, a % b` 計算新的 a 和 b，這樣 b 就是 a 除以 b 的餘數。
3. 當 b 等於 0 時，返回 a，這時 a 就是最大公因數。

這樣，我們就可以用這個函數計算兩個數字的最大公因數了。

```
import math
```

```
def gcd(a, b):
    return math.gcd(a, b)

# 讓用戶輸入兩個整數
a = int(input("請輸入第一個整數 a："))
b = int(input("請輸入第二個整數 b："))

# 計算並顯示結果
result = gcd(a, b)
print(f"{a} 和 {b} 的最大公因數是 {result}")
```

解釋：

1. `math.gcd(a, b)`：此內建函數直接返回 a 和 b 的最大公因數。
2. **手動實現的** `gcd(a, b)`：使用歐幾里得算法，當 b 不為 0 時，通過遞歸（或迴圈）交換 a 和 b，並將 b 替換為 $a\%b$，直到 $b = 0$，此時 a 即為最大公因數。

18. 階乘和： 寫一個函數 sum_of_factorials，接受一個整數 n，並返回從 1 到 n 的所有整數的階乘和。例如：sum_of_factorials(3) 應返回 9（1! + 2! + 3! = 1 + 2 + 6）。

要計算從 1 到 n 的所有整數的階乘和，我們需要寫一個函數 `sum_of_factorials`，它會計算每個數字的階乘並將它們加總起來。

階乘是這樣定義的：

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

例如，3! = 3 × 2 × 1 = 6。

接下來，我們可以利用 Python 的 `math.factorial` 函數來計算階乘，並在循環中加總這些階乘。

```
import math


def sum_of_factorials(n):
    return sum(math.factorial(i) for i in range(1, n + 1))


# 測試範例
print(sum_of_factorials(3))  # 輸出 9 (1! + 2! + 3! = 1 + 2 + 6)
```

解釋：

1. `math.factorial(i)` 計算 i 的階乘。
2. 使用 `sum()` 函數對所有階乘的結果進行加總。
3. `for i in range(1, n + 1)` 則從 1 到 n 遍歷每個整數。

這樣，當你傳入一個數字 n 時，函數將返回從 1 到 n 的階乘和。例如，`sum_of_factorials(3)` 會返回 1! + 2! + 3! = 1 + 2 + 6 = 9。

```
import math


def sum_of_factorials(n):
    total = 0
```

```
    for i in range(1, n + 1):
        total += math.factorial(i)
    return total

# 讓用戶輸入一個數字
n = int(input("請輸入一個整數 n："))

# 計算並顯示結果
result = sum_of_factorials(n)
print(f"從 1 到 {n} 的階乘和是 {result}")
```

解釋：

1. `math.factorial(i)` 用來計算每個數字 i 的階乘。
2. `sum_of_factorials(n)` 用來將從 1 到 n 所有數字的階乘累加起來。
3. `input()` 用來讓用戶輸入數字 n，並將其轉換為整數。

19. 二項式係數： 寫一個函數 binomial_coefficient，接受兩個整數 n 和 k，並返回二項式係數 "n 選 k"（n choose k）。例如：binomial_coefficient(5, 2) 應返回 10。

二項式係數 (Binomial Coefficient) 是數學中計算 "n 選 k" 的方法，通常用符號 $C(n,k)$ 或 $\binom{n}{k}$ 表示。它的計算公式為：

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

這裡：

- $n!$ 是 n 的階乘。
- $k!$ 是 k 的階乘。
- $(n-k)!$ 是 $(n-k)$ 的階乘。

使用這個公式，我們可以計算二項式係數。

```
import math


def binomial_coefficient(n, k):
    return math.factorial(n) // (math.factorial(k) * math.factorial(n - k))


# 測試範例
print(binomial_coefficient(5, 2))  # 輸出 10
```

解釋：

1. `math.factorial(n)` 計算 n 的階乘。
2. 計算 $C(n,k)$ 的公式是 $\frac{n!}{k!(n-k)!}$。
3. `//` 是整數除法，保證結果是整數。

這樣，當你傳入兩個數字 n 和 k 時，函數將返回二項式係數 $\binom{n}{k}$。例如，`binomial_coefficient(5, 2)` 會返回 10，因為 $\binom{5}{2} = \frac{5!}{2!(5-2)!} = \frac{120}{2\times 6} = 10$。

```
import math


def binomial_coefficient(n, k):
```

```
    # 使用組合公式計算二項式係數 C(n, k) = n! / (k! * (n - k)!)
    return math.factorial(n) // (math.factorial(k) * math.factorial(n - k))

# 讓用戶輸入 n 和 k
n = int(input("請輸入 n 的值："))
k = int(input("請輸入 k 的值："))

# 計算並顯示結果
result = binomial_coefficient(n, k)
print(f"二項式係數 C({n}, {k}) 是 {result}")
```

解釋：

1. `math.factorial(n)` 計算 n 的階乘。
2. `binomial_coefficient(n, k)` 根據公式計算 $C(n, k)$。
3. `input()` 用來讓用戶輸入 `n` 和 `k` 的值，並將其轉換為整數。

20. 清單排序：寫一個函數 `sort_list`，接受一個包含整數的清單作為參數，並返回該清單的排序版本（從小到大）。例如：`sort_list([3, 1, 4, 1, 5])` 應返回 `[1, 1, 3, 4, 5]`。

要實現排序功能，我們可以使用 Python 的內建 `sorted()` 函數來對清單進行排序。這個函數會返回一個新的排序後的清單，並且不會改變原來的清單。或者，我們也可以使用 `list.sort()` 方法直接在原清單上進行排序。

以下是使用 `sorted()` 函數來排序清單的實現：

```
def sort_list(lst):
    return sorted(lst)

# 測試範例
print(sort_list([3, 1, 4, 1, 5]))  # 輸出 [1, 1, 3, 4, 5]S
```

解釋：

1. `sorted(lst)` 返回排序後的清單。它會按照數字的升序（從小到大）進行排序。
2. `lst` 是我們傳入的包含整數的清單。

此函數會返回清單的排序版本，不會改變原始清單。如果你希望對原始清單進行排序，可以使用 `list.sort()` 方法：

```
def sort_list(lst):
    lst.sort()
    return lst

# 測試範例
print(sort_list([3, 1, 4, 1, 5]))  # 輸出 [1, 1, 3, 4, 5]
```

這樣，`sort_list([3, 1, 4, 1, 5])` 會返回 `[1, 1, 3, 4, 5]`，並且會在原清單上進行排序。

```
def sort_list(numbers):
    # 將列表從小到大排序
```

```
    numbers.sort()
    return numbers

# 讓用戶輸入一個由逗號分隔的數字字符串
input_string = input("請輸入一個由逗號分隔的數字列表：")

# 將輸入的字符串轉換為數字列表
number_list = [int(num) for num in input_string.split(',')]

# 呼叫 sort_list 函數並顯示結果
sorted_list = sort_list(number_list)
print("排序後的列表:", sorted_list)
```

解釋：

1. `input()` 讓用戶輸入一個以逗號分隔的數字字符串。
2. `split(',')` 將輸入字符串分割成一個數字字符串列表。
3. `[int(num) for num in ...]` 用來將每個分割出來的字符串轉換為整數。
4. `sort()` 對列表進行原地排序，並返回排序後的列表。

21. 寫一個函數 remove_duplicates，接受一個包含整數的清單作為參數，並返回去除重複元素後的清單。例如：remove_duplicates([1, 2, 2, 3, 4, 4, 5]) 應返回 [1, 2, 3, 4, 5]。

要解決這個問題，我們可以寫一個名為 `remove_duplicates` 的函數，使用 Python 的 `set` 來去除重複的元素，然後將結果轉換回清單 (list)。因為 `set` 是無序的，因此我們可以在去除重複元素後，對清單進行排序以保持原來的順序。

```
def remove_duplicates(lst):
    # 使用 set 去除重複元素，並將其轉回 list
    return sorted(list(set(lst)))

# 測試範例
result = remove_duplicates([1, 2, 2, 3, 4, 4, 5])
print(result)  # 輸出: [1, 2, 3, 4, 5]
```

解釋：

1. `set(lst)`：這會將清單轉換為集合 (set)，自動去除所有重複元素。
2. `list(set(lst))`：將去重後的集合轉回清單。
3. `sorted()`：排序去重後的清單，保證結果是按升序排列。

這樣就能達到去除重複並排序的效果。

```
def remove_duplicates(lst):
    # 使用 set 去除重複元素，再轉回清單
    return list(set(lst))

# 讓使用者輸入清單
user_input = input("請輸入一個包含整數的清單（例如：[1, 2, 2, 3, 4, 4, 5]）: ")
```

```
# 使用 eval 轉換為清單
try:
    lst = eval(user_input)
    if isinstance(lst, list) and all(isinstance(i, int) for i in lst):
        result = remove_duplicates(lst)
        print(f"去除重複元素後的清單: {result}")
    else:
        print("請輸入有效的整數清單。")
except:
    print("輸入格式錯誤，請輸入有效的清單格式。")
```

程式解釋：

1. `remove_duplicates(lst)` 函數使用 Python 的 `set` 資料型態去除清單中的重複元素，因為 `set` 本身不允許重複元素。然後再將 `set` 轉回清單（`list()`），以便返回結果。
2. 使用 `input()` 函數讓使用者輸入一個清單（例如 `[1, 2, 2, 3, 4, 4, 5]`）。
3. 使用 `eval()` 函數將輸入的字串轉換為 Python 清單，並檢查輸入格式是否正確。
4. 程式會顯示去除重複元素後的清單。

22. 寫一個函數 reverse_list，接受一個包含整數的清單作為參數，並返回該清單的反轉版本。例如：reverse_list([1, 2, 3, 4, 5]) 應返回 [5, 4, 3, 2, 1]。

為了解決這個問題，我們可以寫一個名為 `reverse_list` 的函數，並利用 Python 的內建方法來反轉清單。常用的反轉清單的方法有幾種，以下是一個簡單的解法：

```
def reverse_list(lst):
    # 使用 list 的內建 reverse 方法來反轉清單
    return lst[::-1]

# 測試範例
result = reverse_list([1, 2, 3, 4, 5])
print(result)  # 輸出: [5, 4, 3, 2, 1]
```

解釋：

1. `lst[::-1]`：這是 Python 的切片語法，表示從頭到尾反向切片。簡單來說，它返回清單的反轉版本。

這樣就能達到反轉清單的目的。

```
def reverse_list(lst):
    # 使用 Python 的 slicing 技術來反轉清單
    return lst[::-1]

# 讓使用者輸入清單
user_input = input("請輸入一個包含整數的清單（例如：[1, 2, 3, 4, 5]）: ")

# 使用 eval 轉換為清單
try:
    lst = eval(user_input)
    if isinstance(lst, list) and all(isinstance(i, int) for i in lst):
```

```
            result = reverse_list(lst)
            print(f"反轉後的清單: {result}")
        else:
            print("請輸入有效的整數清單。")
except:
    print("輸入格式錯誤，請輸入有效的清單格式。")
```

程式解釋：

1. `reverse_list(lst)` 函數使用 Python 切片（slicing）技術 `lst[::-1]` 來反轉清單。這會返回一個新的清單，其中的元素順序與原清單相反。
2. 程式會請使用者輸入一個整數清單（例如 `[1, 2, 3, 4, 5]`）。
3. 使用 `eval()` 函數將輸入的字串轉換為 Python 清單。`eval()` 會將字符串解析並執行，將其轉換成對應的 Python 類型（在這裡是清單）。
4. 程式會檢查輸入是否有效，然後反轉並顯示結果。

例子：

```less
請輸入一個包含整數的清單（例如：[1, 2, 3, 4, 5]）：[1, 2, 3, 4, 5]
反轉後的清單：[5, 4, 3, 2, 1]
```

23. 寫一個函數 find_max_min，接受一個包含整數的清單作為參數，並返回一個包含最大值和最小值的元組。例如：find_max_min([3, 1, 4, 1, 5]) 應返回 (5, 1)。

為了解決這個問題，我們可以寫一個名為 `find_max_min` 的函數，並使用 Python 的內建函數 `max()` 和 `min()` 來分別獲取最大值和最小值，然後返回這兩個值作為元組。

以下是解法：

```
def find_max_min(lst):
    # 使用內建的 max() 和 min() 函數來取得最大值和最小值
    return (max(lst), min(lst))

# 測試範例
result = find_max_min([3, 1, 4, 1, 5])
print(result)  # 輸出: (5, 1)
```

解釋：

1. `max(lst)`：返回清單中的最大值。
2. `min(lst)`：返回清單中的最小值。
3. 最後，我們將最大值和最小值包裝在一個元組中，並返回。

這樣就可以獲得清單中的最大值和最小值，並將其作為元組返回。

```
def find_max_min(lst):
    # 使用內建的 max() 和 min() 函數來找到最大值和最小值
    return (max(lst), min(lst))

# 讓使用者輸入清單
user_input = input("請輸入一個包含整數的清單（例如：[3, 1, 4, 1, 5]）: ")
```

```
# 使用 eval 轉換為清單
try:
    lst = eval(user_input)
    if isinstance(lst, list) and all(isinstance(i, int) for i in lst):
        result = find_max_min(lst)
        print(f"最大值和最小值元組: {result}")
    else:
        print("請輸入有效的整數清單。")
except:
    print("輸入格式錯誤，請輸入有效的清單格式。")
```

程式解釋：

1. `find_max_min(lst)` 函數使用內建的 `max()` 和 `min()` 函數來計算並返回清單中的最大值和最小值，結果以元組的形式返回。
2. 使用 `input()` 讓使用者輸入一個清單（例如 `[3, 1, 4, 1, 5]`）。
3. 使用 `eval()` 解析用戶輸入的字串為 Python 清單。
4. 程式會檢查輸入是否有效，並計算並顯示最大值和最小值的元組。

例子：

```less
請輸入一個包含整數的清單（例如：[3, 1, 4, 1, 5]）: [3, 1, 4, 1, 5]
最大值和最小值元組: (5, 1)
```

24. 寫一個函數 cumulative_sum，接受一個包含整數的清單作為參數，並返回該清單的累積和清單。例如：cumulative_sum([1, 2, 3, 4]) 應返回 [1, 3, 6, 10]。

為了解決這個問題，我們需要寫一個名為 `cumulative_sum` 的函數，該函數將接受一個整數清單，並返回該清單的累積和清單。

累積和指的是每個元素的和，包括它之前所有的元素。例如，給定清單 `[1, 2, 3, 4]`，其累積和為 `[1, 3, 6, 10]`，這是因為：

- 第一個元素是 1。
- 第二個元素是 1 + 2 = 3。
- 第三個元素是 1 + 2 + 3 = 6。
- 第四個元素是 1 + 2 + 3 + 4 = 10。

以下是具體的解法：

```
def cumulative_sum(lst):
    result = []
    current_sum = 0
    for num in lst:
        current_sum += num  # 累加當前元素
        result.append(current_sum)  # 將累加的和加入結果清單
    return result

# 測試範例
result = cumulative_sum([1, 2, 3, 4])
print(result)  # 輸出: [1, 3, 6, 10]
```

解釋：

1. `current_sum`：用來累積清單元素的總和。
2. `for num in lst:`：迭代清單中的每一個元素。
3. 每次迭代時，將當前元素加到 `current_sum`，並將累積和加入結果清單 `result`。

這樣，當我們遍歷完清單後，就會得到累積和的清單。

```
def cumulative_sum(lst):
```

```
    result = []
    current_sum = 0
    for num in lst:
        current_sum += num
        result.append(current_sum)
    return result

# 讓使用者輸入清單
user_input = input("請輸入一個包含整數的清單（例如：[1, 2, 3, 4]）: ")

# 使用 eval 轉換為清單
try:
    lst = eval(user_input)
    if isinstance(lst, list) and all(isinstance(i, int) for i in lst):
        result = cumulative_sum(lst)
        print(f"累積和清單: {result}")
    else:
        print("請輸入有效的整數清單。")
except:
    print("輸入格式錯誤，請輸入有效的清單格式。")
```

程式解釋：

1. `cumulative_sum(lst)` 函數會遍歷清單中的每個數字，並將每個數字累加到當前的總和 `current_sum`，然後將累加的結果添加到 `result` 清單中。
2. 使用 `input()` 讓使用者輸入一個清單（例如 `[1, 2, 3, 4]`）。
3. 使用 `eval()` 將用戶輸入的字串轉換為 Python 清單。
4. 程式會檢查輸入是否有效，並計算並顯示累積和清單。

例子：

less 複製程式碼

```
請輸入一個包含整數的清單（例如：[1, 2, 3, 4]）：[1, 2, 3, 4]
累積和清單: [1, 3, 6, 10]
```

25. 寫一個函數 count_elements，接受一個包含整數的清單作為參數，並返回一個字典，該字典包含清單中每個元素出現的次數。例如：count_elements([1, 2, 2, 3, 3, 3]) 應返回 {1: 1, 2: 2, 3: 3}。

為了解決這個問題，我們可以寫一個名為 `count_elements` 的函數，該函數將接受一個整數清單，並返回一個字典，字典的鍵是清單中的每個元素，值是該元素在清單中出現的次數。

Python 提供了一些方便的方法來解決這個問題，其中最常用的是使用 `collections.Counter` 類，該類專門用來計數元素。這樣我們就可以很簡單地完成這個任務。

以下是具體的解法：

```
from collections import Counter


def count_elements(lst):
    # 使用 Counter 計算清單中每個元素的出現次數
    return dict(Counter(lst))


# 測試範例
result = count_elements([1, 2, 2, 3, 3, 3])
print(result)  # 輸出: {1: 1, 2: 2, 3: 3}
```

解釋：

1. `Counter(lst)`：`Counter` 類將清單中的每個元素作為鍵，並計算它在清單中出現的次數。
2. `dict()`：`Counter` 的結果是 `Counter` 類型，它其實是字典的子類。我們將其轉換為普通字典。

這樣，當我們呼叫 `count_elements` 函數時，會返回一個字典，表示清單中每個元素的出現次數。

```
def count_elements(lst):
    element_count = {}
    for element in lst:
        if element in element_count:
            element_count[element] += 1
        else:
            element_count[element] = 1
    return element_count

# 讓使用者輸入清單
user_input = input("請輸入一個包含整數的清單（例如：[1, 2, 2, 3, 3, 3]）: ")

# 使用 eval 轉換為清單
try:
    lst = eval(user_input)
    if isinstance(lst, list) and all(isinstance(i, int) for i in lst):
        result = count_elements(lst)
        print(f"計數結果: {result}")
    else:
        print("請輸入有效的整數清單。")
except:
    print("輸入格式錯誤，請輸入有效的清單格式。")
```

程式解釋：

1. `count_elements(lst)` 函數會遍歷清單中的每個元素，並計算每個元素出現的次數。
2. 使用 `input()` 讓使用者輸入一個清單（例如 `[1, 2, 2, 3, 3, 3]`）。
3. 使用 `eval()` 將用戶輸入的字串轉換為 Python 清單。`eval()` 是一個函數，會將字符串解析為 Python 表達式。
4. 程式會檢查輸入是否有效，並計算並顯示每個數字的出現次數。

例子：

```
css                                                    複製程式碼

請輸入一個包含整數的清單（例如：[1, 2, 2, 3, 3, 3]）：[1, 2, 2, 3, 3, 3]
計數結果: {1: 1, 2: 2, 3: 3}
```

26. 字母計數器：寫一個函數 `count_letters`，接受一個字串作為參數，並返回一個字典，該字典包含字串中每個字母出現的次數。例如：`count_letters("hello")` 應返回 `{'h': 1, 'e': 1, 'l': 2, 'o': 1}`。

要讓你能夠自己輸入字串並自訂顯示格式，可以使用 `input()` 函數來從使用者獲取輸入，然後將其傳遞到 `count_letters` 函數中。以下是改進後的程式碼，能讓你自己輸入字串和顯示訊息：

```
def count_letters(s, message="字母出現次數："):
    # 初始化一個空字典來存儲字母及其計數
    letter_count = {}

    # 遍歷字串中的每個字母
    for letter in s:
        # 如果字母已經存在於字典中，增加它的計數
        if letter in letter_count:
            letter_count[letter] += 1
        # 如果字母不在字典中，將其加入並初始化計數為 1
        else:
            letter_count[letter] = 1

    # 自訂格式化字典輸出
    result = f"{message} {letter_count}"

    # 返回格式化結果
    return result

# 從使用者獲取字串輸入
input_string = input("請輸入一個字串：")
```

input_message = input("請輸入你想要的訊息：")

呼叫函數並顯示結果

print(count_letters(input_string, message=input_message))

解釋：

1. 使用 `letter_count` 字典來儲存字母及其出現次數。
2. 遍歷字串中的每個字母：
    - 如果字母已經存在於字典中，則將其計數增加 1。
    - 如果字母不在字典中，則將其添加進去並初始化計數為 1。
3. 最後返回該字典。

說明：

1. 使用 `input()` 函數來讓使用者輸入字串 `input_string` 以及顯示訊息 `input_message`。
2. 把這些輸入傳遞給 `count_letters` 函數，並顯示結果。

範例執行：

```arduino
請輸入一個字串：hello
請輸入你想要的訊息：這是我們的字母統計
這是我們的字母統計 {'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

這樣，你就可以靈活地自訂字串和輸出訊息。

27. 寫一個函數 letter_positions，接受一個字串作為參數，並返回一個字典，該字典包含字串中每個字母出現的位置列表。例如：letter_positions("hello") 應返回 {'h': [0], 'e': [1], 'l': [2, 3], 'o': [4]}。

```
def letter_positions(s):
    # 初始化一個空字典來存儲字母及其位置
    position_dict = {}

    # 遍歷字串中的每個字母及其索引
    for index, letter in enumerate(s):
        # 如果字母已經在字典中，將索引添加到位置列表中
        if letter in position_dict:
            position_dict[letter].append(index)
        # 如果字母不在字典中，初始化它的列表並添加當前索引
        else:
            position_dict[letter] = [index]

    # 返回字典
    return position_dict

# 讓使用者輸入字串
input_string = input("請輸入一個字串：")

# 呼叫函數並顯示結果
print(letter_positions(input_string))
```

說明：

1. `enumerate(s)` 用來遍歷字串 `s` 並同時獲取每個字母的索引。
2. 使用字典 `position_dict` 存儲字母及其對應的位置。如果字母已經在字典中，就把新的索引位置加到對應的列表中；如果字母不在字典中，就創建一個新的列表並將當前索引放進去。

範例執行：

arduino 複製程式碼

```
請輸入一個字串：hello
{'h': [0], 'e': [1], 'l': [2, 3], 'o': [4]}
```

這樣，你就可以自己輸入字串並獲得每個字母在字串中的位置了！

28. 寫一個函數 first_last_count，接受一個字串作為參數，並返回一個字典，該字典包含字串中每個字母在首尾位置出現的次數。例如：first_last_count("hello world") 應返回 {'h': 1, 'd': 1, 'o': 2, 'w': 1, 'r': 1}。

```
def first_last_count(s):
    # 初始化一個空字典來存儲字母及其出現次數
    count_dict = {}

    # 只關心字串的第一個和最後一個字母
    first_char = s[0]  # 第一個字母
    last_char = s[-1]  # 最後一個字母

    # 計算字串中第一個字母出現的次數
    count_dict[first_char] = s.count(first_char)

    # 計算字串中最後一個字母出現的次數
    count_dict[last_char] = s.count(last_char)

    # 如果第一個字母和最後一個字母不同，則進行處理
    if first_char != last_char:
        # 確保不重複統計第一個和最後一個字母
        count_dict[first_char] += s.count(first_char) - 1
        count_dict[last_char] += s.count(last_char) - 1

    # 返回字典
    return count_dict
```

讓使用者輸入字串

input_string = input("請輸入一個字串：")

呼叫函數並顯示結果

print(first_last_count(input_string))

說明：

1. `first_char = s[0]` 獲取字串的首字母，`last_char = s[-1]` 獲取字串的末字母。
2. 使用 `count()` 方法來計算字母在字串中的出現次數。
3. 如果首字母和尾字母相同，則對它們的計數會加總在一起；如果不同，則將它們分開處理，避免重複計數。

範例執行：

arduino 複製程式碼

```
請輸入一個字串：hello world
{'h': 1, 'd': 1, 'o': 2, 'w': 1, 'r': 1}
```

這樣你就可以輸入任意字串，並得到字串中首尾字母出現的次數！

29. 一個函數 count_uppercase_letters，接受一個字串作為參數，並返回一個字典，該字典包含字串中每個大寫字母出現的次數。例如：count_uppercase_letters("Hello World") 應返回 {'H': 1, 'W': 1}。

```
def count_uppercase_letters(input_string):
    # 創建一個空字典來存儲大寫字母的計數
    uppercase_count = {}

    # 遍歷字串中的每一個字符
    for char in input_string:
        # 檢查字符是否為大寫字母
        if char.isupper():
            # 如果字母已經在字典中，則增加其計數
            if char in uppercase_count:
                uppercase_count[char] += 1
            else:
                # 否則，將字母加入字典並設置計數為 1
                uppercase_count[char] = 1

    # 返回字典
    return uppercase_count

# 讓使用者輸入字串
user_input = input("請輸入一個字串：")
result = count_uppercase_letters(user_input)
print(result)
```

解釋：

1. `count_uppercase_letters` 函數會接收一個字串 `input_string`。
2. 程式會遍歷字串中的每個字符，並檢查該字符是否是大寫字母 (`char.isupper()` 來檢查)。
3. 如果是大寫字母，則將其存入字典 `uppercase_count`，並累加它的出現次數。
4. 最後，返回該字典。

執行範例：

如果你輸入 "Hello World"，程式會返回：

arduino 複製程式碼

```
{'H': 1, 'W': 1}
```

30. 寫一個函數 count_vowels，接受一個字串作為參數，並返回一個字典，該字典包含字串中每個母音出現的次數。例如：count_vowels("hello world") 應返回 {'e': 1, 'o': 2}。

```
def count_vowels(input_string):
    # 母音字母
    vowels = 'aeiou'
    # 創建一個空字典來存儲母音字母的計數
    vowel_count = {}

    # 遍歷字串中的每一個字符
    for char in input_string:
        # 檢查字符是否為母音字母（忽略大小寫）
        if char.lower() in vowels:
            # 如果字母已經在字典中，則增加其計數
            if char.lower() in vowel_count:
                vowel_count[char.lower()] += 1
            else:
                # 否則，將字母加入字典並設置計數為 1
                vowel_count[char.lower()] = 1

    # 返回字典
    return vowel_count

# 讓使用者輸入字串
user_input = input("請輸入一個字串：")
```

result = count_vowels(user_input)

print(result)

解釋：

1. `count_vowels` 函數會接收一個字串 `input_string`。
2. 程式會遍歷字串中的每個字符，並檢查該字符是否是母音字母（包括大小寫）。
3. 如果是母音字母，則將其存入字典 `vowel_count`，並累加它的出現次數。
4. 最後，返回該字典。

執行範例：

如果你輸入 "hello world"，程式會返回：

arduino 複製程式碼

```
{'e': 1, 'o': 2}
```

31. 寫一個函數 count_non_letter_characters，接受一個字串作為參數，並返回一個字典，該字典包含字串中每個非字母符號出現的次數。例如：count_non_letter_characters("hello world!") 應返回 {' ': 1, '!': 1}。

```
def count_non_letter_characters(input_string):
    # 創建一個空字典來存儲非字母符號的計數
    non_letter_count = {}

    # 遍歷字串中的每一個字符
    for char in input_string:
        # 檢查字符是否不是字母
        if not char.isalpha():
            # 如果字符已經在字典中，則增加其計數
            if char in non_letter_count:
                non_letter_count[char] += 1
            else:
                # 否則，將字符加入字典並設置計數為 1
                non_letter_count[char] = 1

    # 返回字典
    return non_letter_count

# 讓使用者輸入字串
user_input = input("請輸入一個字串：")
result = count_non_letter_characters(user_input)
print(result)
```

解釋：

1. `count_non_letter_characters` 函數會接收一個字串 `input_string`。
2. 程式會遍歷字串中的每個字符，並檢查該字符是否不是字母（使用 `char.isalpha()` 來檢查，這會返回 `True` 如果是字母，反之為 `False`）。
3. 如果字符不是字母，則將其存入字典 `non_letter_count`，並累加它的出現次數。
4. 最後，返回該字典。

執行範例：

如果你輸入 "hello world!"，程式會返回：

arduino 複製程式碼

```
{' ': 1, '!': 1}
```