# 1 calculate

```
def calculate(operation, num1, num2):
    if operation == '加':
        return num1 + num2
    elif operation == '減':
        return num1 - num2
    elif operation == '乘':
        return num1 * num2
    elif operation == '除':
        if num2 != 0:
            return num1 / num2
        else:
            return "除數不能為零"
    else:
        return "無效的操作"

# 從用戶輸入操作和數字
operation = input("請輸入操作（加、減、乘、除）：")
num1 = float(input("請輸入第一個數字："))
num2 = float(input("請輸入第二個數字："))

# 計算並顯示結果
result = calculate(operation, num1, num2)
print(f"計算結果：{result}")
```

## 說明：

1. `input()` 函數會從用戶那裡獲取輸入：
   - `operation` 變數用來接收用戶輸入的操作符（如 "加"、"減"、"乘" 或 "除"）。
   - `num1` 和 `num2` 變數用來接收用戶輸入的數字。
2. 然後，將這些值傳遞給 `calculate` 函數進行計算。
3. 最後，將計算結果打印出來。

# 2 advanced_calculate

```
def advanced_calculate(operation, num1, num2, show_steps=False):
    steps = ""    # 用來記錄計算步驟

    if operation == '加':
        result = num1 + num2
        steps = f"{num1} + {num2} = {result}"
    elif operation == '減':
        result = num1 - num2
        steps = f"{num1} - {num2} = {result}"
    elif operation == '乘':
        result = num1 * num2
        steps = f"{num1} * {num2} = {result}"
    elif operation == '除':
        if num2 != 0:
            result = num1 / num2
            steps = f"{num1} / {num2} = {result}"
        else:
            return "除數不能為零", ""
    elif operation == '次方':
        result = num1 ** num2
        steps = f"{num1} ^ {num2} = {result}"
    elif operation == '取餘數':
        result = num1 % num2
        steps = f"{num1} % {num2} = {result}"
    else:
        return "無效的操作", ""

    # 根據布林值決定是否顯示步驟
    if show_steps:
        return result, steps
    else:
        return result, ""



# 從用戶輸入操作和數字
operation = input("請輸入操作（加、減、乘、除、次方、取餘數）: ")
num1 = float(input("請輸入第一個數字："))
num2 = float(input("請輸入第二個數字："))
show_steps = input("是否顯示計算步驟？(是/否)：").strip().lower() == "是"
```

```
# 計算並顯示結果
result, steps = advanced_calculate(operation, num1, num2, show_steps)
print(f"計算結果：{result}")
if steps:
    print(f"計算過程：{steps}")
```

## 函數解釋：

1. **參數**:
   - `operation`: 字符串，指定要進行的運算（例如 "加"、"減"、"乘"、"除"、"次方" 或 "取餘數"）。
   - `num1` 和 `num2`: 兩個數字，進行指定的運算。
   - `show_steps`: 布林值，決定是否顯示計算過程。
2. **計算邏輯**:
   - 根據 `operation` 的不同，進行加、減、乘、除、次方或取餘數的計算。
   - 若 `operation` 為 "除" 且除數為零，會返回錯誤信息。
   - 計算的過程會被記錄到 `steps` 變數中。
3. **返回結果**:
   - 若 `show_steps` 為 `True`，則返回計算結果和計算步驟；否則只返回計算結果。

# 3 unit_converter

```
def unit_converter(conversion_type, value, decimal_places):
    if conversion_type == '米-公里':
        result = value / 1000
        return round(result, decimal_places)
    elif conversion_type == '克-千克':
        result = value / 1000
        return round(result, decimal_places)
    elif conversion_type == '華氏-攝氏':
        result = (value - 32) * 5/9
        return round(result, decimal_places)
    elif conversion_type == '攝氏-華氏':
        result = (value * 9/5) + 32
        return round(result, decimal_places)
    else:
        return "無效的轉換類型"

# 測試範例
conversion_type = input("請輸入轉換類型（米-公里、克-千克、華氏-攝氏、攝氏-華氏）：")
value = float(input("請輸入值："))
decimal_places = int(input("請輸入小數點後的位數："))

# 計算並顯示結果
result = unit_converter(conversion_type, value, decimal_places)
print(f"轉換結果：{result}")
```

### 函數解釋：

1. `conversion_type`：指定要進行的轉換類型，例如 "米-公里"、"克-千克"、"華氏-攝氏"、"攝氏-華氏" 等。
2. `value`：要轉換的數值。
3. `decimal_places`：小數點後的位數，用來格式化轉換後的結果。

### 轉換邏輯：

- **米-公里**：將米數除以 1000 轉換為公里。
- **克-千克**：將克數除以 1000 轉換為千克。
- **華氏-攝氏**：根據華氏轉攝氏公式 `(華氏 - 32) * 5/9` 進行轉換。
- **攝氏-華氏**：根據攝氏轉華氏公式 `(攝氏 * 9/5) + 32` 進行轉換。

### 返回值：

- 將結果四捨五入到指定的小數點位數後返回。

# 4 math_quiz

```
import random

def math_quiz():
    # 隨機選擇運算符
    operations = ['加', '減', '乘', '除']
    operation = random.choice(operations)

    # 隨機生成兩個數字
    num1 = random.randint(1, 100)
    num2 = random.randint(1, 100)

    # 根據選擇的運算符生成問題和正確答案
    if operation == '加':
        correct_answer = num1 + num2
        question = f"{num1} + {num2} = ?"
    elif operation == '減':
        correct_answer = num1 - num2
        question = f"{num1} - {num2} = ?"
    elif operation == '乘':
        correct_answer = num1 * num2
        question = f"{num1} * {num2} = ?"
    elif operation == '除':
        # 確保除數不為零
        num2 = random.randint(1, 100)
        correct_answer = round(num1 / num2, 2)  # 保留兩位小數
        question = f"{num1} / {num2} = ?"

    # 顯示問題並接受用戶答案
    user_answer = float(input(f"{question} 請輸入你的答案："))

    # 檢查答案是否正確
    if user_answer == correct_answer:
        return "正確"
    else:
        return "錯誤"

# 測試範例
result = math_quiz()
print(result)
```

**函數解釋：**

1. **隨機生成問題**：
   - 隨機選擇一個運算符（加、減、乘、除）。
   - 隨機生成兩個數字 `num1` 和 `num2`。
   - 根據運算符生成相應的數學問題。
2. **計算正確答案**：
   - 根據運算符計算正確的答案：
     - "加"：`num1 + num2`
     - "減"：`num1 - num2`
     - "乘"：`num1 * num2`
     - "除"：`num1 / num2`，並保留兩位小數。
3. **接受用戶答案**：
   - 使用 `input()` 函數提示用戶輸入答案。
   - 將答案轉換為浮點數進行比較。
4. **檢查答案**：
   - 如果用戶輸入的答案與計算的正確答案相同，返回 `"正確"`；否則返回 `"錯誤"`。

# 5 basic_statistics

```
import statistics

def basic_statistics(numbers):
    # 計算平均數
    mean = statistics.mean(numbers)

    # 計算中位數
    median = statistics.median(numbers)

    # 計算標準差
    stdev = statistics.stdev(numbers)

    return mean, median, stdev

# 從用戶輸入數字
input_data = input("請輸入一組數字（以空格分隔）：")

# 將輸入的字串轉換為數字清單
numbers = list(map(float, input_data.split()))

# 計算統計數據
mean, median, stdev = basic_statistics(numbers)

# 顯示結果
print(f"平均數：{mean}")
print(f"中位數：{median}")
print(f"標準差：{stdev}")
```

**說明：**

1. **輸入數字清單**：
   - 使用 `input()` 函數讓用戶輸入一組數字。用戶輸入的數字之間應該以空格分隔。
   - 透過 `split()` 函數將輸入的字串分割成數字的字串清單，然後用 `map()` 函數將這些字串轉換為浮點數並生成一個數字清單。
2. **計算統計數據**：
   - 使用 `statistics.mean()` 計算平均數。
   - 使用 `statistics.median()` 計算中位數。
   - 使用 `statistics.stdev()` 計算標準差。
3. **顯示結果**：
   - 輸出平均數、中位數和標準差。

# 6 geometry_calculator

```
import math

def geometry_calculator(shape_type, dimension1, dimension2=None):
    if shape_type == '圓形':
        # 面積 = π * r^2
        area = math.pi * (dimension1 ** 2)
        return round(area, 2)

    elif shape_type == '矩形':
        # 面積 = 長 * 寬
        area = dimension1 * dimension2
        return round(area, 2)

    elif shape_type == '三角形':
        # 面積 = 0.5 * 底 * 高
        area = 0.5 * dimension1 * dimension2
        return round(area, 2)

    else:
        return "無效的形狀類型"

# 測試範例
shape_type = input("請輸入形狀類型（圓形、矩形、三角形）：")
dimension1 = float(input("請輸入第一個數值（半徑、長度或底邊長度）："))
dimension2 = None

if shape_type == '矩形' or shape_type == '三角形':
    dimension2 = float(input("請輸入第二個數值（寬度或高度）："))

# 計算並顯示面積
area = geometry_calculator(shape_type, dimension1, dimension2)
print(f"面積：{area}")
```

## 函數解釋：

1. `shape_type`：形狀類型，接受 "圓形"、"矩形" 或 "三角形"。
2. `dimension1` 和 `dimension2`：用來表示形狀的長度、寬度或高度，根據不同形狀有所不同：
   - 對於圓形，`dimension1` 是圓的半徑，`dimension2` 不需要提供。
   - 對於矩形，`dimension1` 是長度，`dimension2` 是寬度。
   - 對於三角形，`dimension1` 是底邊長，`dimension2` 是高度。

# 7 reverse_string

```
def reverse_string(input_string):
    # 返回字串的反轉版本
    return input_string[::-1]

# 測試範例
input_string = input("請輸入一個字串：")
reversed_string = reverse_string(input_string)
print(f"反轉後的字串：{reversed_string}")
```

## 函數解釋：

1. `input_string[::-1]`：這是 Python 的切片語法，`[::-1]` 用來將字串反轉。
    - `input_string` 是要反轉的字串。
    - `[::-1]` 表示從頭到尾步長為 `-1`，也就是倒著遍歷字串。

# 8 string_length

```
def string_length(input_string):
    # 返回字串的長度
    return len(input_string)

# 測試範例
input_string = input("請輸入一個字串：")
length = string_length(input_string)
print(f"字串的長度是：{length}")
```

## 函數解釋：

1. `len(input_string)`：Python 的 `len()` 函數返回傳入字串的長度。
    - `input_string` 是用戶提供的字串。

# 9 toggle_case

```
def toggle_case(input_string):
    # 使用 str.swapcase() 方法來互換字母的大小寫
    return input_string.swapcase()


# 測試範例
input_string = input("請輸入一個字串：")
toggled_string = toggle_case(input_string)
print(f"大小寫互換後的字串：{toggled_string}")
```

## 函數解釋：

1. `input_string.swapcase()`：這是 Python 的字符串方法 `swapcase()`，它會將字串中所有大寫字母轉換為小寫字母，並將所有小寫字母轉換為大寫字母。

# 10 word_count

```
def word_count(input_string):
    # 使用 str.split() 方法將字串分割為單詞，然後計算單詞的數量
    words = input_string.split()
    return len(words)

# 測試範例
input_string = input("請輸入一個字串：")
count = word_count(input_string)
print(f"單詞的數量是：{count}")
```

## 函數解釋：

1. `input_string.split()`：這個方法會將字串按照空白字符（包括空格、換行符等）分割成一個單詞列表。當沒有指定分隔符時，`split()` 方法會自動以空格或任何空白字符為分隔符。
2. `len(words)`：這將計算分割後的單詞列表的長度，即單詞的數量。

# 11 is_palindrome

```
def is_palindrome(input_string):
    # 將字串轉換為小寫，並移除空格進行比較
    normalized_string = input_string.lower().replace(" ", "")

    # 檢查字串是否等於反轉後的字串
    return normalized_string == normalized_string[::-1]

# 測試範例
input_string = input("請輸入一個字串：")
if is_palindrome(input_string):
    print("是迴文")
else:
    print("不是迴文")
```

## 函數解釋：

1. `input_string.lower()`：將字串轉換為小寫，這樣就不會受到大小寫的影響。
2. `input_string.replace(" ", "")`：移除字串中的空格，避免空格影響比較（如果需要考慮其他非字母字符，也可以進行額外的處理）。
3. `normalized_string[::-1]`：使用切片將字串反轉，並與原字串進行比較。
4. `normalized_string == normalized_string[::-1]`：如果字串等於它的反轉版本，則說明這是一個迴文。

# 繪製簡單的線條圖

```
import turtle

def draw_radiating_lines(num_lines):
    # 設定畫布
    screen = turtle.Screen()
    screen.bgcolor("white")  # 設定背景顏色

    # 創建一個 turtle 物件
    pen = turtle.Turtle()
    pen.speed(0)  # 設定畫筆速度為最快

    # 計算每條線的角度增量
    angle_increment = 360 / num_lines

    # 繪製輻射線條
    for i in range(num_lines):
        pen.setheading(i * angle_increment)  # 設定畫筆方向
        pen.penup()
        pen.goto(0, 0)  # 移動到畫布中心
        pen.pendown()
        pen.forward(100)  # 畫線條

    # 完成後關閉畫布
    turtle.done()

# 請用戶輸入輻射線條的數量
num_lines = int(input("請輸入輻射線條的數量："))

# 調用函數繪製輻射線條
draw_radiating_lines(num_lines)
```

**程式解釋：**

1. `turtle.Screen()`：創建一個畫布，並設置背景顏色為白色。
2. `turtle.Turtle()`：創建一個 `Turtle` 物件，用來畫線。
3. `pen.setheading(angle)`：設置畫筆的角度方向，每次繪製的線條會根據這個角度進行旋轉。
4. `pen.penup()` 和 `pen.pendown()`：控制畫筆的提起和放下，`penup` 使畫筆不留痕跡，`pendown` 讓畫筆開始畫。
5. `pen.goto(0, 0)`：將畫筆移動到畫布的中心。
6. `pen.forward(100)`：畫一條長度為 100 的直線。
7. `turtle.done()`：繪製完成後保留畫布，直到用戶關閉。

# 繪製同心方形

```
import turtle

def draw_concentric_squares(num_squares, side_increment):
    # 設定畫布
    screen = turtle.Screen()
    screen.bgcolor("white")  # 設定背景顏色

    # 創建一個 turtle 物件
    pen = turtle.Turtle()
    pen.speed(5)  # 設定畫筆速度，數字越大越快

    # 設定初始邊長
    side_length = 20

    for _ in range(num_squares):
        # 畫一個方形
        for _ in range(4):
            pen.forward(side_length)  # 畫一條邊
            pen.left(90)  # 轉 90 度
        # 增加邊長
        side_length += side_increment

        # 移到畫下一個方形的位置
        pen.penup()
        pen.goto(-side_length/2, side_length/2)  # 重新定位到方形的左上角
        pen.pendown()

    # 完成後保留畫布
    turtle.done()

# 請用戶輸入同心方形的數量和每次增加的邊長
num_squares = int(input("請輸入同心方形的數量："))
side_increment = int(input("請輸入每次增加的邊長："))

# 調用函數繪製同心方形
draw_concentric_squares(num_squares, side_increment)
```

## 主要調整：

1. `pen.speed(5)`：這樣設置可以控制繪圖的速度，數字越高繪製越快。這樣可以讓圖形更快速顯示，特別是在畫布上繪製很多方形時。
2. `turtle.done()`：這個函數確保繪圖完成後畫布仍然顯示，直到手動關閉它。

# 繪製同心圓

```
import turtle

# 接收用戶輸入
num_circles = int(input("請輸入同心圓的數量："))
radius_increment = int(input("請輸入每次圓的半徑增量："))

# 設置畫布
screen = turtle.Screen()
screen.bgcolor("white")  # 設定背景顏色為白色

# 創建海龜
pen = turtle.Turtle()
pen.speed(10)  # 設定繪圖速度，越大越快

# 使用 for 迴圈繪製同心圓
for i in range(num_circles):
    pen.penup()  # 提起畫筆，移動到起始位置
    pen.goto(0, - (i * radius_increment))  # 移動到圓的起始位置，Y 軸向下移動
    pen.pendown()  # 放下畫筆，準備繪製圓
    pen.circle(i * radius_increment)  # 繪製圓，半徑為 i * radius_increment

# 完成繪製，保持畫布開啟
turtle.done()
```

## 變更說明：

1. `num_circles` 和 `radius_increment` 由用戶通過 `input()` 輸入。這樣可以靈活地讓用戶設定同心圓的數量和半徑增量。
    - `input()` 會接收用戶的輸入，返回值是字串，所以我們使用 `int()` 來將字串轉換為整數。
2. `int(input(...))` 用來將用戶輸入的值轉換為整數型別。

# 繪製交錯矩形

```
import turtle

# 接收用戶輸入
num_rectangles = int(input("請輸入交錯矩形的數量："))
width = int(input("請輸入矩形的寬度："))
height = int(input("請輸入矩形的高度："))

# 設置畫布
screen = turtle.Screen()
screen.bgcolor("white")  # 設定背景顏色為白色

# 創建海龜
pen = turtle.Turtle()
pen.speed(10)  # 設定繪圖速度，越大越快

# 使用 for 迴圈繪製交錯矩形
for i in range(num_rectangles):
    pen.penup()  # 提起畫筆，準備移動到繪製位置

    # 交錯排列：每次移動矩形的起始位置
    # 當 i 為偶數時，矩形往右移，當 i 為奇數時，矩形往左移
    if i % 2 == 0:
        pen.goto(-width * (i // 2), height * (i // 2))  # 每次依次向右和向下移動
    else:
        pen.goto(width * ((i + 1) // 2), height * (i // 2))  # 每次交錯向左和向下移動

    pen.pendown()  # 放下畫筆，開始繪製矩形

    # 繪製矩形
    for _ in range(2):  # 矩形需要兩次長邊，兩次短邊
        pen.forward(width)  # 繪製長邊
        pen.left(90)  # 轉 90 度
        pen.forward(height)  # 繪製短邊
        pen.left(90)  # 轉 90 度

    pen.penup()  # 提起畫筆，準備繪製下一個矩形

# 完成繪製，保持畫布開啟
turtle.done()
```

## 程式說明：

1. **用戶輸入**：
   - `num_rectangles`：交錯矩形的數量。
   - `width`：每個矩形的寬度。
   - `height`：每個矩形的高度。
2. **矩形的交錯排列**：
   - 在 `for` 迴圈中，當迴圈變數 `i` 為偶數時，矩形向右移動；當 `i` 為奇數時，矩形向左移動。這樣就能創造出交錯排列的效果。
3. **繪製矩形**：
   - 透過 `pen.forward()` 和 `pen.left(90)` 來繪製每個矩形。
   - 矩形繪製完畢後，海龜回到起始位置，準備繪製下一個矩形。

# 繪製對角線

```
from browser import document as doc
from browser import html

# 創建畫布元素
canvas = html.CANVAS(width=500, height=500)
doc <= canvas

brython_div = doc["brython_div1"]
brython_div <= canvas

# 繪製對角線
context = canvas.getContext('2d')

# 使用 for 迴圈繪製從左上角到右下角的對角線
for i in range(500):
    context.fillStyle = 'black'    # 設置畫筆顏色
    context.fillRect(i, i, 1, 1)    # 繪製一個小矩形，從(i, i)到右下角
```

## 說明：

1. **畫布的創建**：我們創建了一個 500x500 像素的畫布。
2. **畫布的插入**：直接將畫布元素插入到文檔中，這樣畫布會顯示在頁面上。
3. **繪製對角線**：通過一個 `for` 迴圈，我們繪製了一系列小矩形，這些矩形的頂點從 `(0,0)` 開始，最終形成了一條對角線。

# 繪製格子圖案

```
from browser import document as doc
from browser import html

# 創建畫布元素
canvas = html.CANVAS(width=400, height=400)
doc <= canvas  # 直接將畫布插入到文檔中

# 獲取畫布的上下文
context = canvas.getContext('2d')

# 設置格子的大小
grid_size = 100  # 每個格子的大小，400px / 4 = 100px

# 使用 for 迴圈繪製 4x4 格子
for row in range(4):  # 繪製 4 行
    for col in range(4):  # 繪製 4 列
        x = col * grid_size  # 計算每個格子的 X 位置
        y = row * grid_size  # 計算每個格子的 Y 位置
        context.fillStyle = 'lightgray'  # 設置顏色
        context.fillRect(x, y, grid_size, grid_size)  # 繪製格子
        if (row + col) % 2 == 0:  # 條件：每個格子交替顏色
            context.fillStyle = 'black'
            context.fillRect(x, y, grid_size, grid_size)  # 繪製交替顏色的格子
```

## 代碼解釋：

1. **畫布設置**：
   - 我們創建了一個 400x400 像素的畫布，這樣就能夠繪製 4x4 格子。
   - 每個格子的邊長為 100 像素，這樣 400px ÷ 4 = 100px，畫布的每個格子就會是 100x100 像素。
2. **繪製格子的邏輯**：
   - 外層 `for` 迴圈控制行（4行），內層 `for` 迴圈控制列（4列）。
   - 每個格子的起始位置是 `(x, y)`，我們使用 `fillRect(x, y, 100, 100)` 來繪製正方形。
   - 用 `if (row + col) % 2 == 0` 來讓格子交替顯示不同顏色（這是一種模擬棋盤格樣式的方式）。如果條件為真，就填充一個顏色，否則繪製另一個顏色。

# 繪製不同顏色的線條

```
from browser import document as doc
from browser import html
import random

# 創建畫布元素
canvas = html.CANVAS(width=400, height=400)
doc <= canvas  # 直接將畫布插入到文檔中

# 獲取畫布上下文
context = canvas.getContext('2d')

# 顏色列表，可以從中隨機選擇顏色
colors = ['red', 'blue', 'green', 'orange', 'purple', 'yellow', 'pink']

# 使用 for 迴圈繪製若干條線條，每次改變顏色
for i in range(10):  # 繪製 10 條線條
    # 隨機選擇一種顏色
    context.strokeStyle = random.choice(colors)  # 改變線條顏色
    context.lineWidth = 2  # 設置線條寬度

    # 計算線條的起始點和終點
    start_x = random.randint(0, canvas.width)  # 隨機起點 X
    start_y = random.randint(0, canvas.height)  # 隨機起點 Y
    end_x = random.randint(0, canvas.width)  # 隨機終點 X
    end_y = random.randint(0, canvas.height)  # 隨機終點 Y

    # 開始繪製線條
    context.beginPath()  # 開始路徑
    context.moveTo(start_x, start_y)  # 移動到起始點
    context.lineTo(end_x, end_y)  # 繪製到終點
    context.stroke()  # 繪製線條
```

## 代碼說明：

1. **顏色選擇**：
   - 我們使用一個顏色列表（`colors`），並在每次迴圈中隨機選擇一種顏色來繪製線條。
   - `random.choice(colors)` 用來隨機選擇顏色。
2. **繪製線條**：
   - 使用 `for` 迴圈來繪製多條線條。在這個例子中，繪製了 10 條線條（`for i in range(10)`）。
   - 每次繪製線條時，隨機生成起點和終點的座標，並選擇一個顏色。
   - `context.beginPath()` 和 `context.moveTo()` 用來開始路徑並設定起點，然後使用 `context.lineTo()` 來設定終點，最後使用 `context.stroke()` 來繪製線條。
3. **畫布設置**：
   - 畫布的大小為 400x400 像素，您可以根據需要修改這個大小。