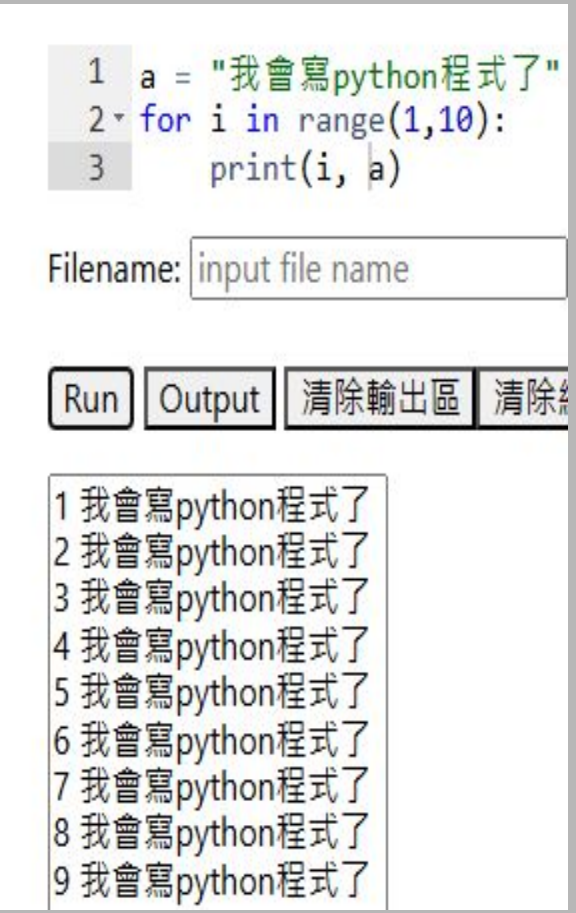
計算機程式報告

41323148　鄭宇哲
41323150　蕭日政

W-

~7

For 迴圈



1 **定義變數與設定迴圈範圍**

a 儲存字串 `"我會寫python程式了"`，`range(1, 10)` 產生從 1 到 9 的整數序列

2 **進行迴圈**

每次迴圈，變數 `i` 取值從 1 到 9

3 **輸出結果**

`print(i, a)` 同時印出變數 `i` 和字串 `a`，中間以空格分隔

W-10

等腰三角形的星號輸出

```
n = 5  # 總共的行數
space = '_'
for i in range(n):  # 對於每一行
    # 印出空格
    print(space * (n - i - 1), end='')  # 在每行前面印出空格
    # 印出星號
    print('*' * (2 * i + 1))  # 每行印出 2*i + 1 個星號
```

成品:

```
____*
___***
__*****
_*******
*********
<completed in 2.40
ms>
```

1.n = 5 定義三角形的總行數(此處為 5 行)

2.使用 for 迴圈 每一行依次計算並印出空格與星號的組合

3.print(space * (n - i - 1), end='') 在每行前印出適量的空格，使三角形對齊
n - i - 1 計算出該行需要的空格數量
print('*' * (2 * i + 1)) 在每行印出星號，數量依次增加，符合等腰三角形的模式

Python 保留字與合法變數命名

```
'''
Python 的 keywords 為保留字, 不可作為變數名稱
'''
import keyword

# 取得 Python 的 keywords
keywords = keyword.kwlist

print(keywords)
and1 = [5,[4,3],2,1]
print(and1[2])
```

成品:

```
['False', 'None', 'True',
'and', 'as', 'assert',
'async', 'await', 'break',
'class', 'continue', 'def',
'del', 'elif', 'else',
'except', 'finally', 'for',
'from', 'global', 'if',
'import', 'in', 'is',
'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise',
'return', 'try', 'while',
'with', 'yield']
2
<completed in  3.30
ms>
```

1.Python 的 Keywords 是保留字，無法作為變數名稱

使用 keyword.kwlist 獲取 Python 中所有保留字的列表，並打印出來
保留字像 and, or, if 等無法作為變數名稱

2.正確的變數命名

定義了一個名為 and1 的變數，其中包含一個多層巢狀列表。雖然 and 是保留字，但 and1 是合法的變數名稱

3.訪問列表元素

使用 and1[2] 獲取列表的第三個元素（索引從 0 開始計算），並打印其 值（此處為 2）

利用python以及數學公式計算

```python
def calculate_acceleration(v0_kmh, v, d):
    """
    Calculate the constant acceleration required to stop an
    final velocity, and the stopping distance.

    Parameters:
    v0_kmh (float): Initial velocity in km/h.
    v (float): Final velocity in m/s (typically 0 when stopp
    d (float): Distance over which the object stops (in mete

    Returns:
    float: The constant acceleration in m/s².
    """
    # Convert initial velocity from km/h to m/s
    v0 = (v0_kmh * 1000) / 3600  # Convert km/h to m/s

    # Calculate acceleration using the equation: a = (v^2 -
    a = (v**2 - v0**2) / (2 * d)

    return a

# Example usage:
v0_kmh = 310  # Initial velocity in km/h
v = 0  # Final velocity in m/s (since the object stops)
d = 1000  # Distance in meters

# Call the function to calculate acceleration
acceleration = calculate_acceleration(v0_kmh, v, d)

# Output the result
print(f"The constant acceleration required to stop the jet i
```

```python
def calculate_acceleration(v0_kmh, v, d):
    """
    計算停止一個物體所需的恆定加速度，已知初速度、末速度和停

    參數:
    v0_kmh (float): 初速度，單位為公里/小時 (km/h)。
    v (float): 末速度，單位為公尺/秒 (m/s)（停止時通常為 0）
    d (float): 停止距離，單位為公尺 (m)。

    返回:
    float: 恆定加速度，單位為公尺/秒² (m/s²)。
    """
    # 將初速度從公里/小時轉換為公尺/秒
    v0 = (v0_kmh * 1000) / 3600  # 將 km/h 轉換為 m/s

    # 使用公式計算加速度: a = (v^2 - v0^2) / (2 * d)
    a = (v**2 - v0**2) / (2 * d)

    return a
# 範例使用:
v0_kmh = 310  # 初速度，單位為公里/小時 (km/h)
v = 0  # 末速度，單位為公尺/秒 (m/s)（物體停止時）
d = 1000  # 停止距離，單位為公尺 (m)

# 調用函數計算加速度
acceleration = calculate_acceleration(v0_kmh, v, d)

# 輸出結果
print(f"為使噴射機停止所需的恆定加速度為 {acceleration:.2f}
```

1.**函式 `calculate_acceleration`**

參數

v0_kmh: 初始速度（以 km/h 為單位）
v: 最終速度（以 m/s 為單位，通常為 0）
d: 停止距離（以公尺為單位）

公式

使用運動學公式計算加速度

2.單位轉換

v0v_0v0 從 km/h 轉為 m/s，

3.結果計算

計算出所需的加速度，並回傳該值

4.範例使用

初始速度:310 km/h
最終速度:0 m/s（靜止）
停止距離:1000 公尺

成品：

The constant acceleration required to stop the jet is -3.71 m/s².

W-11

繪製近似圓形的程式碼

```
import math

# 定義圓形列印函數
def print_circle():
    # 圓形的參數
    size = 10  # 字元區域大小 (10x10)
    radius = size // 2  # 圓的半徑
    center = radius - 0.5  # 圓心位置

    # 調整比例因子，讓圓形更接近正圓
    aspect_ratio = 2  # 調整行與列的比例

    # 繪製圓形
    for y in range(size):
        for x in range(size * aspect_ratio):
            # 計算該點是否在圓內
            distance = math.sqrt(((x / aspect_ratio) - center) ** 2 + (y - center)
** 2)
            if distance <= radius:
                print("*", end="")  # 圓內用 "*" 表示
            else:
                print(" ", end="")  # 圓外用空格表示
        print()  # 換行

# 呼叫函數列印圓形
print_circle()
```

初始化參數與圓形配置

設定繪圖區域大小 (`size`)、圓的半徑 (`radius`)、圓心 (`center`)，並用比例因子 (`aspect_ratio`) 調整列印的寬高比例，讓圓形更接近正圓。

巢狀迴圈遍歷點座標

使用兩層迴圈遍歷每個點 `(x, y)` 的座標，計算該點到圓心的距離，判斷是否在圓的範圍內。

繪製圓形

如果點位於圓內，列印 `*`；否則列印空格，最後換行完成整個圓形圖案的輸出。

成品：

在 HTML Canvas 上繪製格子與圓形

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

# 利用 html 建立 canvas 超文件物件
canvas = html.CANVAS(width=400, height=400)
brython_div = doc["brython_div1"]
brython_div <= canvas

# 每一格的 pixel 數
gs = 40

# gs*tc = canvas width and height
ctx = canvas.getContext("2d")

def dRect(lux, luy, w, h, s=1, c="lightgrey"):
    ctx.lineWidth = s
    ctx.strokeStyle = c  # 修正大小寫
    ctx.beginPath()
    ctx.rect(lux, luy, w, h)
    ctx.stroke()

def grid(width, height, grid_pix):
    for i in range(width):
        for j in range(height):
            dRect(i * grid_pix, j * grid_pix, grid_pix, grid_pix, 1, "lightgrey")

def fill(x, y, color):
    ctx.fillStyle = color
    ctx.fillRect(x * gs, y * gs, gs, gs)

def draw_circle(grid_size, radius):
    center = grid_size // 2
    for y in range(grid_size):
        for x in range(grid_size):
            # 使用格子中心進行距離計算
            distance = (((x + 0.5) - center) ** 2 + ((y + 0.5) - center) ** 2) ** 0.5
            if distance <= radius:
                fill(x, y, "black")

# 繪製 10x10 的格子
grid(10, 10, gs)

# 使用黑色方格圈出一個圓，半徑為 4 格
draw_circle(10, 4)
```

建立畫布與格子設定

使用 `html.CANVAS` 建立 400x400 畫布，設定每格大小為 40 像素，並繪製 10x10 的灰色網格。

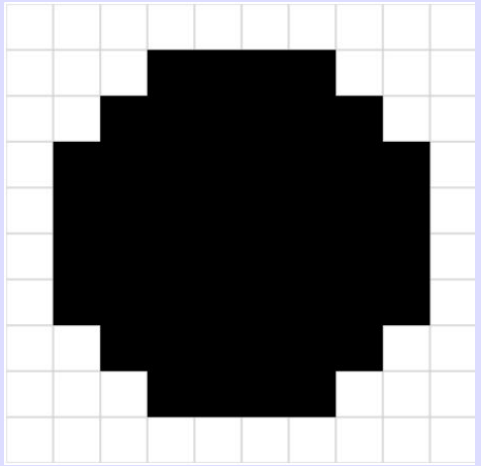
繪製矩形與填充方格

定義 `dRect` 函數繪製單個格子邊框，並用 `fill` 函數填滿指定方格的顏色。

繪製圓形

使用 `draw_circle` 函數，根據方格中心與圓心的距離判斷哪些格子位於圓內，填滿黑色方格形成圓形。

成品：



W-12

循環中印出目前數字


```
n = 5  #總共的行數
spase = "_"
'''
for i in range(n):  #對於每一行
    #印出空格
    print(space * (n - i - 1), end='')
'''
"""
for i in range(n):  #對於每一行
    #印出空格
    print(space * (n - i - 1), end='')
"""
for i in range(n):  #對於每一行
    #印出空格
    #print(space * (n - i - 1), end='')
    print("目前的數字 " + str(i))
```

設定行數與變數

定義行數為 `n = 5`，並設定變數 `space = "_"` 作為模擬空格的符號。

編寫並註解部分邏輯

使用巢狀迴圈的雛形，計算並印出每行所需的空格數(註解的部分展示如何實現金字塔對齊，但未執行)。

輸出測試行號

用 `print("目前的數字 " + str(i))` 輸出每行的行號，作為測試迴圈邏輯的基礎，方便後續調整程式。

成品：

```
目前的數字 0
目前的數字 1
目前的數字 2
目前的數字 3
目前的數字 4
<completed in  3.90
ms>
```

W-13

多彩幾何圖形繪製 (1)

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

canvas = html.CANVAS(width=500, height=500)
brython_div = doc["brython_div1"]
brython_div <= canvas

ctx = canvas.getContext("2d")
ctx.lineWidth = 4

# 混色的
ctx.globalCompositeOperation = "screem"

# 黑邊圓*2 (每個畫四個邊邊)
ctx.strokeStyle = 'black'

# 第一個圓
ctx.beginPath()
ctx.arc(160, 160, 141, 0.25 * math.pi, 0.75 * math.pi)
ctx.fillStyle = "Coral"
ctx.fill()
ctx.stroke()

ctx.beginPath()
ctx.arc(160, 160, 141, 0.75 * math.pi, 1.25 * math.pi)
ctx.fillStyle = "MediumPurple"
ctx.fill()
ctx.stroke()

ctx.beginPath()
ctx.arc(160, 160, 141, 1.25 * math.pi, 1.75 * math.pi)
ctx.fillStyle = "LimeGreen"
ctx.fill()
ctx.stroke()

ctx.beginPath()
ctx.arc(160, 160, 141, 1.75 * math.pi, 0.25 * math.pi)
ctx.fillStyle = "Beige"
ctx.fill()
ctx.stroke()
# 第二個圓
ctx.beginPath()
ctx.arc(260, 260, 141, 1.25 * math.pi, 1.75 * math.pi)
ctx.fillStyle = "SkyBlue"
ctx.fill()
ctx.stroke()

ctx.beginPath()
ctx.arc(260, 260, 141, 1.75 * math.pi, 0.25 * math.pi)
ctx.fillStyle = "Crimson"
ctx.fill()
ctx.stroke()

ctx.beginPath()
ctx.arc(260, 260, 141, 0.25 * math.pi, 0.75 * math.pi)
ctx.fillStyle = "LightCyan"
ctx.fill()
ctx.stroke()

ctx.beginPath()
ctx.arc(260, 260, 141, 0.75 * math.pi, 1.25 * math.pi)
ctx.fillStyle = "Gold"
ctx.fill()
ctx.stroke()

# 紅方形*2 (4個三角形)
ctx.strokeStyle = 'red'
ctx.beginPath()
ctx.moveTo(60, 60)
ctx.lineTo(60, 260)
ctx.lineTo(260, 260)
ctx.fillStyle = "ForestGreen"
ctx.fill()
ctx.stroke()

ctx.beginPath()
ctx.moveTo(60, 60)
ctx.lineTo(260, 60)
ctx.lineTo(260, 260)
ctx.fillStyle = "DarkSlateBlue"
ctx.fill()
ctx.stroke()
```

```
ctx.beginPath()
ctx.moveTo(160, 160)
ctx.lineTo(360, 160)
ctx.lineTo(360, 360)
ctx.fillStyle = "LightYellow"
ctx.fill()
ctx.stroke()

ctx.beginPath()
ctx.moveTo(160, 160)
ctx.lineTo(160, 360)
ctx.lineTo(360, 360)
ctx.fillStyle = "PaleGreen"
ctx.fill()
ctx.stroke()

# 重疊的小方型
ctx.beginPath()
ctx.moveTo(160, 160)
ctx.lineTo(160, 260)
ctx.lineTo(260, 260)
ctx.fillStyle = "SteelBlue"
ctx.fill()
ctx.stroke()

ctx.beginPath()
ctx.moveTo(160, 160)
ctx.lineTo(260, 160)
ctx.lineTo(260, 260)
ctx.fillStyle = "DarkOrchid"
ctx.fill()
ctx.stroke()

# 藍斜線
ctx.strokeStyle = 'blue'
ctx.beginPath()
ctx.moveTo(60, 60)
ctx.lineTo(360,360)
ctx.stroke()
```

初始化畫布與設置繪圖環境

創建 500x500 的畫布並插入網頁，取得繪圖上下文，設置線條寬度和混色模式。

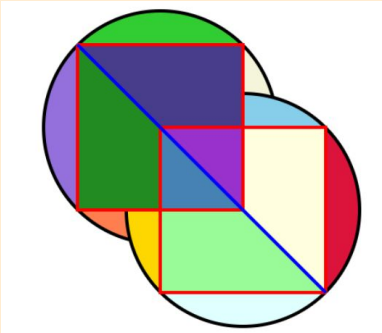
繪製圓形、三角形與矩形

畫兩個圓形，每個圓分為四個顏色區域；畫多個三角形和矩形，填充不同顏色，並進行重疊。

繪製連接線條

使用藍色斜線從左上角到右下角，增強圖形之間的連結和視覺效果。

成品：



繪製多彩幾何圖形 (2)

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

# 設置畫布
canvas = html.CANVAS(width=600, height=600)
brython_div = doc["brython_div1"]
brython_div <= canvas

ctx = canvas.getContext("2d")
ctx.lineWidth = 1

# 畫兩個矩形（紅色邊框）
ctx.strokeStyle = 'red'
ctx.beginPath()
ctx.rect(100, 100, 200, 200)
ctx.rect(200, 200, 200, 200)
ctx.stroke()

# 畫兩個圓形（黑色）
ctx.strokeStyle = 'black'
ctx.beginPath()
ctx.arc(200, 200, 141.4, 0, 2 * 3.14)
ctx.stroke()
ctx.beginPath()
ctx.arc(300, 300, 141.4, 0, 2 * 3.14)
ctx.stroke()

# 計算點的布林值屬性
def calculate_point_properties(x, y):
    in_circle1 = (x - 200) ** 2 + (y - 200) ** 2 < 141.4 ** 2
    in_circle2 = (x - 300) ** 2 + (y - 300) ** 2 < 141.4 ** 2
    in_square1 = 100 <= x <= 300 and 100 <= y <= 300
    in_square2 = 200 <= x <= 400 and 200 <= y <= 400
    return {
        "in_circle1": in_circle1,
        "in_circle2": in_circle2,
        "in_square1": in_square1,
        "in_square2": in_square2,
    }

# 根據布林值屬性判定區域
def get_region_via_conditions(properties):
    regions = []

    if properties["in_circle1"] and properties["in_square1"]:
        regions.append("circle1_in_square1")  # 第一個圓內且在第一個正方形內
    elif properties["in_circle1"]:
        regions.append("circle1_outside_square1")  # 第一個圓內但不在正方形內
    if properties["in_circle2"] and properties["in_square2"]:
        regions.append("circle2_in_square2")  # 第二個圓內且在第二個正方形內
    elif properties["in_circle2"]:
        regions.append("circle2_outside_square2")  # 第二個圓內但不在正方形內

    if not properties["in_circle1"] and not properties["in_circle2"]:
        regions.append("outside_circles")  # 不在兩個圓內

    return regions

# 根據區域決定顏色
def get_region_color_by_regions(regions):
    if "circle1_in_square1" in regions:
        return "green"
    if "circle1_outside_square1" in regions:
        return "red"
    if "circle2_in_square2" in regions:
        return "blue"
    if "circle2_outside_square2" in regions:
        return "yellow"
    if "outside_circles" in regions:
        return "lightgreen"  # 圓外部區域
    return None

# 掃描畫布並著色
def scan_and_draw():
    for y in range(0, canvas.height, 10):  # 每10像素掃描一次
        for x in range(0, canvas.width, 10):
            properties = calculate_point_properties(x, y)
            regions = get_region_via_conditions(properties)
            color = get_region_color_by_regions(regions)
            if color:
                ctx.fillStyle = color
                ctx.beginPath()
                ctx.arc(x + 5, y + 5, 5, 0, 2 * 3.14)
                ctx.fill()

# 開始掃描和標示
scan_and_draw()
```

1.建立畫布與上下文

使用 `html.CANVAS` 元素創建 500x500 的畫布，並透過 `getContext("2d")` 獲取繪圖上下文 (`ctx`)，用於執行後續的繪圖操作

定義全局混色模式 `ctx.globalCompositeOperation = "screen"`，以實現顏色的混合效果

2.繪製圖形

使用 `ctx.arc()` 方法繪製兩個圓形，將其分成多個扇形區域，並使用不同的填充顏色（如 Coral、SkyBlue 等），最後用 `ctx.stroke()` 加上黑色或其他指定顏色的邊框

使用 `ctx.moveTo()` 和 `ctx.lineTo()` 定義多個三角形區域，填充不同顏色（如 ForestGreen、DarkSlateBlue），並繪製紅色邊框，組合成方形

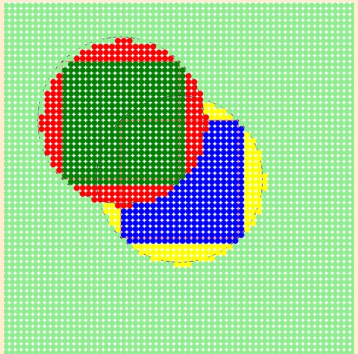
在兩個大方形內繪製疊加的小方形，並以 `ctx.moveTo()` 和 `ctx.lineTo()` 畫出藍色斜線，使畫面層次更豐富

3.填充顏色與描邊

使用 `ctx.fillStyle` 設定填充顏色，搭配 `ctx.fill()` 填滿每個圖形

使用 `ctx.strokeStyle` 設定描邊顏色，搭配 `ctx.stroke()` 為每個圖形添加清晰的邊界，使其更加突出

成品：



W-14

使用 Brython 繪製三圓交集圖

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

# 定義一個函式來初始化畫布
def initialize_canvas(canvas_width=400, canvas_height=400, id="bry
    canvas = html.CANVAS(width=canvas_width, height=canvas_height)
    brython_div = doc[id]  # 獲取指定 id 的 div 元素
    brython_div <= canvas  # 將 canvas 插入到該 div 中
    ctx = canvas.getContext("2d")
    return canvas, ctx

# 定義一個隨機顏色生成函式
def random_color_generator():
    r = random.randint(0, 255)
    g = random.randint(0, 255)
    b = random.randint(0, 255)
    return f"rgb({r}, {g}, {b})"

# 判斷點 (px, py) 是否在圓內
def is_point_in_circle(px, py, cx, cy, r):
    return (px - cx) ** 2 + (py - cy) ** 2 <= r ** 2

# 繪製三個圓並處理交集
def draw_circles(x1, y1, r1, x2, y2, r2, x3, y3, r3):
    canvas, ctx = initialize_canvas(400, 400)
    ctx.clearRect(0, 0, canvas.width, canvas.height)

    color_dict = {}

    for py in range(0, canvas.height):
        for px in range(0, canvas.width):
            # 判斷該點是否在各圓內
            in_circle1 = is_point_in_circle(px, py, x1, y1, r1)
            in_circle2 = is_point_in_circle(px, py, x2, y2, r2)
            in_circle3 = is_point_in_circle(px, py, x3, y3, r3)
```

中間在下一頁

```
# 設定三個圓的圓心和半徑
x1, y1, r1 = 150, 200, 100  # 圓1
x2, y2, r2 = 250, 200, 100  # 圓2
x3, y3, r3 = 200, 300, 100  # 圓3

# 呼叫函式繪製三個圓
draw_circles(x1, y1, r1, x2, y2, r2, x3, y3, r3)
```

初始化畫布與繪圖環境

使用 initialize_canvas 函式創建 400x400 的畫布，將其嵌入指定的網頁區域，獲取繪製工具來完成圖形操作。

設定輔助功能

定義 random_color_generator 函式生成隨機顏色，用於圖形的顏色設定。

定義 is_point_in_circle 函式，檢查某個範圍是否落在圓形內，供交集部分使用。

繪製與交集處理準備

在 draw_circles 函式內，通過設定中心座標與半徑確定三個圓形的位置與大小，清理畫布後開始繪製，並準備實現交集部分的處理。

定義三個圓的屬性

分別設定圓1、圓2、圓3的中心座標與半徑，用來描述三個圓的大小與位置關係。

呼叫繪圖函式

利用 draw_circles 函式，將上述三個圓的屬性傳遞進去，進行畫布繪製與處理。

呈現三圓交疊效果

在畫布上繪製三個圓，根據後續的繪圖邏輯，視覺化它們的重疊部分與不同區域的區分。

中間段

```
# 確定交集情況
if in_circle1 and in_circle2 and in_circle3:
    if "intersection_all" not in color_dict:
        color_dict["intersection_all"] = random_color_generator()
    current_color = color_dict["intersection_all"]
elif in_circle1 and in_circle2:
    if "intersection_1_2" not in color_dict:
        color_dict["intersection_1_2"] = random_color_generator()
    current_color = color_dict["intersection_1_2"]
elif in_circle1 and in_circle3:
    if "intersection_1_3" not in color_dict:
        color_dict["intersection_1_3"] = random_color_generator()
    current_color = color_dict["intersection_1_3"]
elif in_circle2 and in_circle3:
    if "intersection_2_3" not in color_dict:
        color_dict["intersection_2_3"] = random_color_generator()
    current_color = color_dict["intersection_2_3"]
elif in_circle1:
    if "circle1" not in color_dict:
        color_dict["circle1"] = random_color_generator()
    current_color = color_dict["circle1"]
elif in_circle2:
    if "circle2" not in color_dict:
        color_dict["circle2"] = random_color_generator()
    current_color = color_dict["circle2"]
elif in_circle3:
    if "circle3" not in color_dict:
        color_dict["circle3"] = random_color_generator()
    current_color = color_dict["circle3"]
else:
    current_color = "white"  # 背景色

ctx.fillStyle = current_color
ctx.fillRect(px, py, 1, 1)  # 填充一個像素
```

判斷像素區域與圓的關係

判斷像素區域是否屬於多個圓的重疊部分，或單獨屬於某個圓的範圍。

使用條件結構分別處理三個圓的交集、兩個圓的交集，以及單獨屬於某個圓的情況。

分配顏色給不同區域

使用字典 `color_dict` 儲存各範圍的顏色。

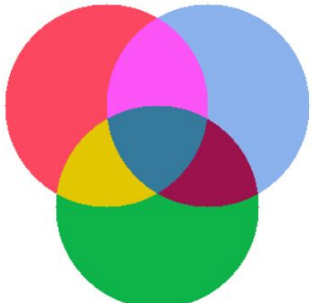
當某範圍的顏色尚未定義時，調用 `random_color_generator` 為該範圍生成一個隨機顏色並記錄下來，確保每個範圍的顏色一致。

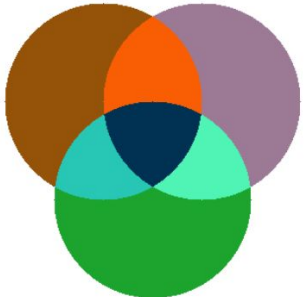
將顏色填充到畫布

設定 `ctx.fillStyle` 為當前區域的顏色

使用 `ctx.fillRect` 方法以像素為單位填充該區域到畫布，逐步完成整體圖形的繪製

成品：





W-15

從1累加到100

```python
# 定義函數來計算和顯示結果
def calculate_sum():
    total = 0
    for i in range(1, 101):  # 從 1 遍歷到 100
        total += i
    print(f"總和是: {total}")  # 在控制台顯示結果

# 呼叫函數執行
calculate_sum()
```

1 **定義函數**

函數 `calculate_sum` 使用變數 `total` 將總和初始化為 0，並透過迴圈累加 1 到 100。

2 **計算總和**

`for i in range(1, 101)` 逐一取值從 1 到 100，將每個數字加到 `total`。

3 **輸出結果**

使用 `print` 將計算結果輸出為 `"總和是: 5050"`。

成品：

```
總和是: 5050
<completed in  3.60
ms>
```

呼叫函式進行累加 (1)

```python
def addto(start, end):
    """
    計算從 start 到 end 的整數總和。

    參數:
    - start: 起始值 (整數)
    - end: 結束值 (整數)

    回傳:
    - 總和 (整數)
    """
    total = sum(range(start, end + 1))  # 使用內建的 sum 函數
    return total

# 呼叫函式並印出結果
result = addto(1, 100)
print(f"累加的結果是: {result}")
```

1 **設計函數 `addto`**

函數接收兩個參數，`start` 和 `end`，使用 Python 的內建函數 `sum` 配合 `range` 計算總和，簡潔高效。

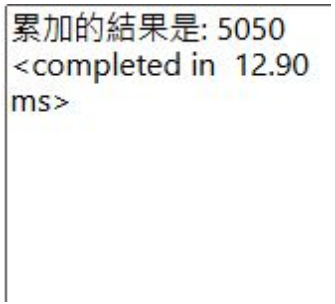
2 **計算過程**

`range(start, end + 1)` 生成從 `start` 到 `end`(包含)的序列，將其所有值加總，並返回。

3 **執行與結果顯示**

函數被呼叫並傳入範圍 `1` 到 `100`，結果被存入變數 `result`，最後用格式化字串顯示 `"累加的結果是： 5050"`。

成品：



呼叫函式進行累加並篩選 (1)

```python
def add_only_even(start, end):
    """
    計算從 start 到 end 之間所有偶數的總和。

    參數:
    - start: 起始值 (整數)
    - end: 結束值 (整數)

    回傳:
    - 偶數總和 (整數)
    """
    # 使用範圍產生偶數並計算總和
    total = sum(i for i in range(start, end + 1) if i % 2 == 0)
    return total

# 呼叫函式並印出結果
result = add_only_even(1, 100)
print(f"累加的偶數結果是: {result}")
```

1.函數定義

定義函數 `add_only_even`，接受兩個參數 :`start` 和 `end`，代表累加的起始和結束值。

2.偶數篩選與累加

使用生成式 `sum(i for i in range(start, end + 1) if i % 2 == 0)`，篩選出範圍內所有偶數(`i % 2 == 0`)，並計算總和。

3.函數呼叫與輸出結果

呼叫函數計算從 1 到 100 的偶數總和，並使用 `print` 輸出結果。結果顯示為累加的偶數結果是: 2550

成品：

```
累加的偶數結果是:
2550
<completed in  12.10
ms>
```

呼叫函式進行累加並篩選 (2)

```python
def add_avoid_8(start, end):
    """計算從 start 到 end 的累加結果，避開包含 '8' 的數字以及基數（奇數）"""
    valid_numbers = [x for x in range(start, end + 1) if '8' not in str(x) and x % 2 == 0]
    print("符合條件的數字:", valid_numbers)  # 打印符合條件的數字，看看哪些數字被選中
    return sum(valid_numbers)

# 測試函式
result = add_avoid_8(1, 100)
print(f"從 1 到 100 的累加總和（避開包含 '8' 的數字以及基數）是：{result}")
```

1.條件篩選

使用生成式 `[x for x in range(start, end + 1) if '8' not in str(x) and x % 2 == 0]`:

`if '8' not in str(x)`:將數字轉換為字串，檢 查是否不包含字元 `'8'`

`and x % 2 == 0`:確認該數字為偶數

2.計算符合條件的數字總和

將符合條件的數字存入 `valid_numbers`，再用 `sum(valid_numbers)` 計算總和

3.顯示過程與結果

使用 `print("符合條件的數字：", valid_numbers)` 打印符合條件的數字清單，用於檢 查邏輯，

並且最後打印總和結果為符合條件的數字

成品：

```
符合條件的數字: [2, 4, 6, 10, 12, 14, 16, 20, 22, 24, 26, 30, 32, 34, 36, 40, 42, 44, 46, 50, 52, 54, 56, 60, 62, 64, 66, 70, 72, 74, 76, 90, 92, 94, 96, 100]
從 1 到 100 的累加總和（避開包含 '8' 的數字以及基數）是：1688
<completed in 13.80 ms>
```

W16_exam1

圖形繪製與標註示範

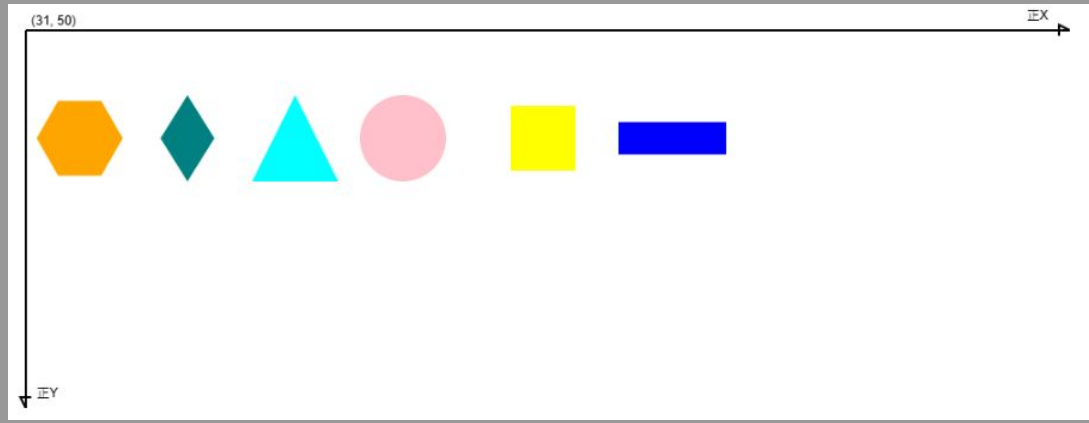
```python
from browser import document, html
import math

# 初始化畫布
def initialize_canvas(canvas_width=1000, canvas_height=400):
    canvas = html.CANVAS(width=canvas_width, height=canvas_height)
    document["brython_div1"] <= canvas  # 加入到 HTML 容器中
    ctx = canvas.getContext("2d")  # 獲取 2D 繪圖上下文
    return canvas, ctx

# 設定學號後四碼原點 (31, 50)
origin_x, origin_y = 3.1 * 10, 5 * 10  # 原始座標

# 初始化畫布
canvas, ctx = initialize_canvas(1000, 400)

# 繪製 X 軸和 Y 軸，只畫向右和向下的部分，並標註
def draw_axes():
    ctx.beginPath()

    # 繪製 X 軸向右延伸
    ctx.moveTo(origin_x, origin_y)  # 原點
    ctx.lineTo(canvas.width, origin_y)  # 向右延伸
    ctx.lineTo(canvas.width - 10, origin_y - 5)  # X 軸箭頭
    ctx.lineTo(canvas.width - 10, origin_y + 5)  # X 軸箭頭
    ctx.strokeStyle = "black"
    ctx.lineWidth = 2
    ctx.stroke()

    # 繪製 Y 軸向下延伸
    ctx.beginPath()
    ctx.moveTo(origin_x, origin_y)  # 原點
    ctx.lineTo(origin_x, canvas.height)  # 向下延伸
    ctx.lineTo(origin_x - 5, canvas.height - 10)  # Y 軸箭頭
    ctx.lineTo(origin_x + 5, canvas.height - 10)  # Y 軸箭頭
    ctx.stroke()

    # 標註原點及方向
    ctx.fillStyle = "black"
    ctx.font = "12px Arial"
    ctx.fillText(f"({31}, {50})", origin_x + 5, origin_y - 5)
    ctx.fillText("正X", canvas.width - 40, origin_y - 10)  # X 軸正方向
    ctx.fillText("正Y", origin_x + 10, canvas.height - 10)  # Y 軸正方向
```

1.初始化畫布與坐標系：畫布大小設置為 1000x400，並以指定的原點 (31, 50) 作為坐標系的基準。

2.繪製各種形狀：包含六邊形、菱形、三角形、圓形、矩形及長條形，這些形狀會根據設定的坐標進行繪製，並且會在畫布上顯示。

3.繪製 X、Y 軸：畫出坐標系的 X、Y 軸並標註其方向，將原點標註為 (31, 50)。

成品



使用chatgpt 進行編程

```python
# 繪製六邊形
def draw_hexagon(x, y, size, color):
    ctx.beginPath()
    for i in range(6):
        angle = (i * 60) * (math.pi / 180)
        px = x + size * math.cos(angle)
        py = y + size * math.sin(angle)
        if i == 0:
            ctx.moveTo(px, py)
        else:
            ctx.lineTo(px, py)
    ctx.closePath()
    ctx.fillStyle = color
    ctx.fill()

# 繪製菱形
def draw_diamond(x, y, width, height, color):
    ctx.beginPath()
    ctx.moveTo(x, y - height / 2)
    ctx.lineTo(x + width / 2, y)
    ctx.lineTo(x, y + height / 2)
    ctx.lineTo(x - width / 2, y)
    ctx.closePath()
    ctx.fillStyle = color
    ctx.fill()

# 繪製三角形
def draw_triangle(x, y, size, color):
    ctx.beginPath()
    ctx.moveTo(x, y - size)  # 頂點
    ctx.lineTo(x - size, y + size)  # 左下角
    ctx.lineTo(x + size, y + size)  # 右下角
    ctx.closePath()
    ctx.fillStyle = color
    ctx.fill()
```

以下都是使用 **chatgpt** 來
幫我們運算出
每個點跟點的位置
然後再進行繪製

```python
# 繪製圓形
def draw_circle(x, y, radius, color):
    ctx.beginPath()
    ctx.arc(x, y, radius, 0, 2 * math.pi)
    ctx.fillStyle = color
    ctx.fill()

# 繪製矩形
def draw_rectangle(x, y, width, height, color):
    ctx.beginPath()
    ctx.rect(x, y, width, height)
    ctx.fillStyle = color
    ctx.fill()

# 繪製長條形
def draw_bar(x, y, width, height, color):
    draw_rectangle(x, y, width, height, color)

# 繪製圖形
def draw_shapes():
    # 繪製圖形時調整 y 坐標，使其在原點下方
    draw_hexagon(origin_x + 50, origin_y + 100, 40, "orange")  # 六邊形
    draw_diamond(origin_x + 150, origin_y + 100, 50, 80, "teal")  # 菱形
    draw_triangle(origin_x + 250, origin_y + 100, 40, "cyan")  # 三角形
    draw_circle(origin_x + 350, origin_y + 100, 40, "pink")  # 圓形
    draw_rectangle(origin_x + 450, origin_y + 70, 60, 60, "yellow")  # 矩形
    draw_bar(origin_x + 550, origin_y + 85, 100, 30, "blue")  # 長條形

# 執行繪圖
draw_axes()
draw_shapes()
```

程式註解：

html：提供建立 HTML 標籤的方法，例如 <canvas>。

ocument（縮寫為 doc）：操作 HTML 文件結構（DOM），用於尋找或修改頁面中的元素

建立canvas超文件 且設定 寬高皆為500

取得畫布的 2D 繪圖上下文（context），它是所有繪圖操作的基礎

beginPath()：開始新繪圖路徑。

strokeStyle = 'black'：設定線條顏色為黑色。

lineWidth = 2：設定線條寬度為 2 像素。

moveTo(31, 48)：設定線條起點為 (31, 48)。

lineTo(400, 48)：設定線條終點為 (400, 48)。

stroke()：繪製出這條線條。

font：設定文字樣式為 12px 的 Arial 字體。

fillStyle：設定文字顏色為黑色。

fillText：將文字 "(31, 48)" 繪製在 (35, 61)，稍微靠右下於原點。

w16_exam2

迷宮圖繪製程式

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

# 建立畫布
canvas = html.CANVAS(width=600, height=600)  # 畫布大小增加兩倍
brython_div = doc["brython_div1"]  # 將畫布放置於網頁中的指定 div
brython_div <= canvas

# 取得 canvas 的 2D 繪圖上下文
ctx = canvas.getContext("2d")

# 每一格的像素大小
pixel_scale = 100  # 每格的像素比例增加兩倍 (原為20，放大到40)

# 定義原點 (31, 50) 作為起始點偏移
offset_x = 31 * 2  # 放大兩倍
offset_y = 50 * 2  # 放大兩倍

# 定義迷宮線條
lines = [
    ((0, 0), (1, 0)),
    ((2, 0), (5, 0)),
    ((5, 0), (5, 5)),
    ((0, 5), (4, 5)),
    ((4, 5), (4, 3)),
    ((2, 3), (4, 3)),
    ((1, 1), (1, 4)),
    ((3, 3), (3, 2)),
    # 第二條線
    ((0, 0), (0, 5)),
    ((0, 5), (4, 5)),
    ((4, 4), (2, 4)),
    ((2, 2), (2, 1)),
    ((1, 1), (4, 1)),
    ((4, 0), (4, 2)),
]

# 繪製迷宮線條
ctx.strokeStyle = "blue"  # 設定線條顏色
ctx.lineWidth = 2  # 設定線條寬度

for line in lines:
    start = line[0]
    end = line[1]
    ctx.beginPath()
    ctx.moveTo(
        offset_x + start[0] * pixel_scale,
        offset_y + start[1] * pixel_scale
    )
    ctx.lineTo(
        offset_x + end[0] * pixel_scale,
        offset_y + end[1] * pixel_scale
    )
    ctx.stroke()

# 將座標標示移動到紅點附近的適當位置
ctx.font = "18px Arial"  # 放大字體大小 (原為14px)
ctx.fillStyle = "red"
ctx.fillText("(31, 50)", offset_x + 10, offset_y - 15)  # 在紅點的右上方標出座標
```

1.建立畫布與設定像素比例

建立一個 600×600600 \times 600600×600 的畫布，插入到 HTML 中的指定區域
設定每格像素比例為 100，並計算原點 (31,50)(31, 50)(31,50) 的偏移位置，用作迷宮起始點

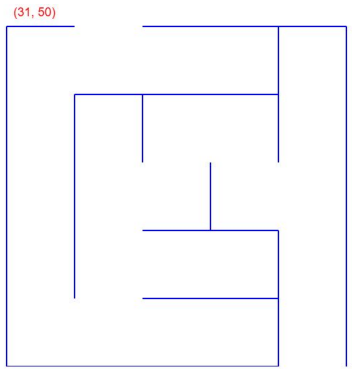
2.定義與繪製迷宮線條

定義迷宮的結構，以線段的起點和終點座標表示
根據像素比例與偏移 值計算線條的實際位置，繪製為藍色線條在畫布上呈現

3.標示原點座標

使用紅色在畫布上標示原點 (31,50)(31, 50)(31,50) 的位置
在紅點右上方顯示座標標籤，字體加大以便於 閱讀

成品：



w16_exam3

迷宮繪製與過關路線顯示

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

# 建立畫布
canvas = html.CANVAS(width=600, height=600)  # 畫布大小增加兩倍
brython_div = doc["brython_div1"]  # 將畫布放置於網頁中的指定 div
brython_div <= canvas

# 取得 canvas 的 2D 繪圖上下文
ctx = canvas.getContext("2d")

# 每一格的像素大小
pixel_scale = 100  # 每格的像素比例增加兩倍 (原為20，放大到40)

# 定義原點 (31, 50) 作為起始點偏移
offset_x = 31 * 2  # 放大兩倍
offset_y = 50 * 2  # 放大兩倍

# 定義迷宮線條 (外框)
lines = [
    ((0, 0), (1, 0)),
    ((2, 0), (5, 0)),
    ((5, 0), (5, 5)),
    ((0, 5), (4, 5)),
    ((4, 5), (4, 3)),
    ((2, 3), (4, 3)),
    ((1, 1), (1, 4)),
    ((3, 3), (3, 2)),
    # 第二條線
    ((0, 0), (0, 5)),
    ((0, 5), (4, 5)),
    ((4, 4), (2, 4)),
    ((2, 2), (2, 1)),
    ((1, 1), (4, 1)),
    ((4, 0), (4, 2)),
]

# 繪製迷宮線條 (外框)
ctx.strokeStyle = "black"  # 設定外框顏色為黑色
ctx.lineWidth = 2  # 設定線條寬度

for line in lines:
    start = line[0]
    end = line[1]
    ctx.beginPath()
    ctx.moveTo(
        offset_x + start[0] * pixel_scale,
        offset_y + start[1] * pixel_scale
    )
    ctx.lineTo(
        offset_x + end[0] * pixel_scale,
        offset_y + end[1] * pixel_scale
    )
    ctx.stroke()

# 定義過關路線
solution_path = [
    ((1.5, 0), (1.5, 0.5)),
    ((0.5, 0.5), (1.5, 0.5)),
    ((0.5, 0.5), (0.5, 4.5)),
    ((0.5, 4.5), (1.5, 4.5)),
    ((1.5, 4.5), (1.5, 2.5)),
    ((1.5, 2.5), (2.5, 2.5)),
    ((2.5, 2.5), (2.5, 1.5)),
    ((2.5, 1.5), (3.5, 1.5)),
    ((3.5, 1.5), (3.5, 2.5)),
    ((3.5, 2.5), (4.5, 2.5)),
    ((4.5, 2.5), (4.5, 5)),
]

# 繪製過關路線
ctx.strokeStyle = "red"  # 設定過關路線顏色為紅色
ctx.lineWidth = 3  # 過關路線稍粗一些，便於區分

for line in solution_path:
    start = line[0]
    end = line[1]
    ctx.beginPath()
    ctx.moveTo(
        offset_x + start[0] * pixel_scale,
        offset_y + start[1] * pixel_scale
    )
    ctx.lineTo(
        offset_x + end[0] * pixel_scale,
        offset_y + end[1] * pixel_scale
    )
    ctx.stroke()

# 將座標標示移動到紅點附近的適當位置
ctx.font = "18px Arial"  # 放大字體大小 (原為14px)
ctx.fillStyle = "blue"  # 設定座標文字顏色為藍色
ctx.fillText("(31, 50)", offset_x + 10, offset_y - 15)  # 在紅點的右上方標出座標
```

1.繪製迷宮框架

根據定義的座標繪製迷宮的外框，並將原點(31,50) 平移到畫布中(從exam_2取來)

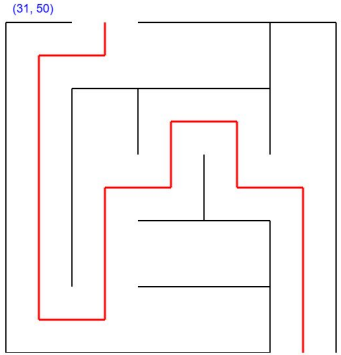
2.繪製過關路線

將過關路線繪製為紅色線條，使用較粗的線條，使路徑清晰可見

3.標示起始座標

在畫布上以藍色文字標示原點(31,50)的座標，字體大小調整為18px

成品：



THANK YOU