# **測試環境設定**

## **測試範例**

開啟您的終端機，建立工作資料夾 (如果沒有)，並進入該資料夾。 在工作資料夾中，執行下列命令，建立新專案名稱:Sample1

管理員身份: Set-ExecutionPolicy RemoteSigned

dotnet new console -n Sample1 -f net8.0

code -r Sample1



撰寫一個簡單的.net 程式，計算數字的階乘，並觀察 Copilot 如何協助產生程式碼。

**提示：**

// 寫一個方法計算數字的階乘

Program.cs

using System;

// 寫一個方法計算數字的階乘

int Factorial(int n)

{

if (n == 0)

return 1;

else

return n \* Factorial(n - 1);

}

// 主程式

Console.WriteLine("請輸入一個非負整數：");

int number;

while (!int.TryParse(Console.ReadLine(), out number) || number < 0)

{

Console.WriteLine("輸入無效，請輸入一個非負整數：");

}

int result = Factorial(number);

Console.WriteLine($"{number} 的階乘是 {result}");

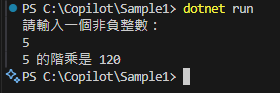
// 程式結束

// 解說: 這段程式碼定義了一個遞迴方法 Factorial 用來計算非負整數的階乘。主程式部分提示使用者輸入一個非負整數，並使用 TryParse 方法確保輸入有效。最後，計算並輸出該數字的階乘結果。

**Copilot 產生：** Copilot 會自動產生 factorial 方法與 main 方法，減少撰寫樣板程式碼的時間。透過這種小型程式測試，可確保 Copilot 設定正確，並準備好處理更複雜的任務。

使用 命令啟動應用程式。 應用程式將會執行，並在命令下方列印

dotnet run



# **基本.Net程式設計與 AI 協作**

## **寫簡單 .Net 程式**

Copilot 能快速產生常見任務的程式碼，例如計算、字串處理等。以下為判斷質數的範例：

**提示：**

// 寫一個CS方法判斷數字是否為質數

Program.cs

// 寫一個CS方法判斷數字是否為質數

bool IsPrime(int n)

{

if (n <= 1) return false;

for (int i = 2; i \* i <= n; i++)

{

if (n % i == 0) return false;

}

return true;

}

// 測試方法

Console.WriteLine(IsPrime(11)); // True

Console.WriteLine(IsPrime(4)); // False

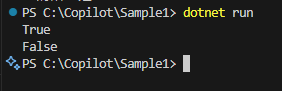
//解說

// 這個方法IsPrime接受一個整數n作為參數，首先檢查n是否小於或等於1，如果是，則返回false，因為質數定義為大於1的自然數。接著，使用一個for迴圈從2開始檢查到√n（即i \* i <= n），如果n能被i整除，則返回false，表示n不是質數。如果迴圈結束後沒有找到任何因數，則返回true，表示n是質數。

**Copilot 產生：** Copilot 會產生 isPrime(int n) 方法，並優化效率，例如提前排除 1 與 2、3 的倍數，並使用 6k±1 規則減少不必要的迴圈。測試方法則示範如何呼叫並印出結果。

使用 命令啟動應用程式。 應用程式將會執行，並在命令下方列印

dotnet run



# **使用 .Net集合與串流**

## **範例一：過濾使用者年齡**

**提示：**

// 寫一個方法，從 User 物件清單中過濾出符合年齡條件的使用者

// 寫一個方法，從 User 物件清單中過濾出符合年齡條件的使用者

using System;

using System.Collections.Generic;

using System.Linq;

public class User

{

public string Name { get; set; }

public int Age { get; set; }

}

public class UserFilter

{

public static List<User> FilterByAge(List<User> users, int minAge, int maxAge)

{

return users.Where(u => u.Age >= minAge && u.Age <= maxAge).ToList();

}

}

public class UserService

{

public static void Main(string[] args)

{

List<User> users = new List<User>

{

new User { Name = "Alice", Age = 25 },

new User { Name = "Bob", Age = 30 },

new User { Name = "Charlie", Age = 35 },

new User { Name = "David", Age = 40 }

};

int minAge = 30;

int maxAge = 40;

List<User> filteredUsers = UserFilter.FilterByAge(users, minAge, maxAge);

Console.WriteLine("Filtered Users:");

foreach (var user in filteredUsers)

{

Console.WriteLine($"{user.Name}, Age: {user.Age}");

}

}

}

// 程式輸出：

// Filtered Users:

// Bob, Age: 30

// Charlie, Age: 35

// David, Age: 40

//解說

// 這段程式碼定義了一個User類別，包含Name和Age屬性。UserFilter類別提供了一個靜態方法FilterByAge，用於根據年齡範圍過濾使用者。在UserService的Main方法中，創建了一個User物件的清單，然後使用UserFilter.FilterByAge方法過濾出年齡在30到40之間的使用者，並將結果輸出到控制台。

//

// 程式碼中使用了LINQ的Where方法來進行過濾操作，這使得代碼簡潔且易於理解。'

// 最後，透過迴圈將符合條件的使用者名稱和年齡打印出來。

// 這樣的設計使得過濾邏輯與使用者資料結構分離，便於維護和擴展。

// 此外，使用靜態方法使得過濾功能可以在不需要實例化UserFilter類別的情況下直接調用。

// 這種方法提高了代碼的可重用性和靈活性。

**Copilot 產生：**

* 使用 FilterUsersByAge ，快速過濾出年齡大於等於指定值的使用者。
* 產生 User 類別，並在 main 方法示範如何使用。

dotnet run

## 

## **範例二：產生隨機產品名稱**

**提示：**

// 寫一個方法，隨機產品名稱清單,產品要有ID,名稱,價格屬性

// 寫一個方法，隨機產品名稱清單,產品要有ID,名稱,價格屬性

using System;

using System.Collections.Generic;

using System.Linq;

class Product

{

public int ID { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

}

class Program

{

static void Main(string[] args)

{

var products = GenerateRandomProducts(10);

foreach (var product in products)

{

Console.WriteLine($"ID: {product.ID}, Name: {product.Name}, Price: {product.Price:C}");

}

}

static List<Product> GenerateRandomProducts(int count)

{

var random = new Random();

var products = new List<Product>();

var productNames = new List<string>

{

"Laptop", "Smartphone", "Tablet", "Headphones", "Smartwatch",

"Camera", "Printer", "Monitor", "Keyboard", "Mouse"

};

for (int i = 0; i < count; i++)

{

var product = new Product

{

ID = i + 1,

Name = productNames[random.Next(productNames.Count)],

Price = Math.Round((decimal)(random.NextDouble() \* 1000), 2) // Price between 0 and 1000

};

products.Add(product);

}

return products;

}

}

//解說:

// 這段程式碼定義了一個Product類別，包含ID、Name和Price屬性。

// Main方法中呼叫GenerateRandomProducts方法來生成指定數量的隨機產品清單，並將每個產品的詳細資訊輸出到控制台。

// GenerateRandomProducts方法使用隨機數生成器來選擇產品名稱並設定價格，最後返回一個包含隨機產品的清單。

// 執行此程式碼將會在控制台顯示10個隨機生成的產品資訊。

// // 注意: 每次執行程式碼時，生成的產品名稱和價格可能會不同，因為它們是隨機生成的。

// 你可以調整GenerateRandomProducts方法中的count參數來生成不同數量的產品。

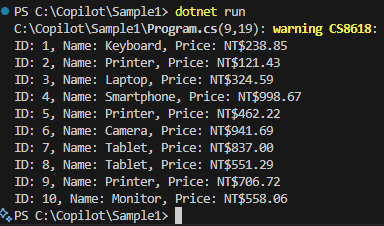
// 你也可以擴展productNames清單以包含更多產品名稱。

// 這樣可以增加生成產品的多樣性。

**Copilot 產生：**

* 使用形容詞與名詞陣列，隨機組合產生產品名稱。
* 產生 GenerateRandomProductNames 方法，並在 main 方法示範如何使用。

dotnet run



Copilot Chat

ASK: Auto

prng cwe修正建議



