

靜態成員

TYIC桃高資訊社

函式與方法

函式(**function**)是指可以執行某些程式碼的東西

函式可以被重複呼叫(**call**、**invoke**)、

傳入(**pass**)引數、傳回(**return**)結果

所以函式可以簡化程式

函式可以定義(**define**)在幾乎任何地方

而若將函式定義在類別(**class**)中

則稱為方法(**method**)

而由於 **Java** 是完全物件導向程式語言

(**fully object-oriented language**)

所以在 **Java** 中沒有函式，只有方法

方法

方法必須先被定義才能被呼叫

```
class Main {  
    返回值型別 方法名稱(參數型別1 參數名稱1, 參數型別2 參數名稱2, ...) {  
        陳述式...  
    }  
  
    存取修飾子 static 返回值型別 方法名稱(參數型別1 參數名稱1, 參數型別2 參數名稱2, ...) {  
        陳述式...  
    }  
  
    public static void main(String[] args) {}  
}
```

java

返回值型別還可以填入 **void** 表示不回傳東西

方法定義還可以加上存取修飾子(**Access Modifier**)和 **static**

存取修飾子有三種可以填，不填表示預設

static 表示是靜態的(依附於類別)，不填表示是動態的(依附於物件)

本次只會介紹無存取修飾子的靜態方法

回傳

在方法定義中
使用 `return` 來回傳

`return` 回傳值；

java

方法回傳後便不再繼續執行
即後方的程式碼皆不會執行

注意回傳值的類型一定要和一開始定義的一樣

若方法不回傳東西則不須寫回傳值

呼叫

呼叫方法就是傳入**引數(argument)**並執行方法中的陳述式
然後方法會回傳結果

方法名稱(**引數1**, **引數2**, ...)

java

方法的**參數(parameter)**

會依序被替換成

傳入的**引數**

而**參數**就是個變數

可以對其做任何

變數能做的事

且其作用域為該方法內

```
import java.util.Scanner;

public class Main1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        System.out.println(add(a, b));
    }

    static int add(int a, int b) {
        int sum = a + b;
        return sum;
    }
}
```



6 8
14 console

java

遞迴

遞迴(Recursion)是指函數(方法)自己呼叫自己，直到終止

 Google 搜尋
「遞迴」

所以遞迴可以用來處理可以拆分成許多相似小問題的大問題

如費波那契數列(Fibonacci sequence)

 OEIS A000045
費波那契數列

第 0 項為 0，第 1 項為 1
之後的每一項皆是前兩項之和
以數學遞迴關係式表示就是

$$\begin{cases} F_0 = 0, F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \quad (n \geq 2) \end{cases}$$

以 F_4 舉例： $F_4 = F_3 + F_2$


$$= F_2 + F_1 + F_1 + F_0$$

$$= F_1 + F_0 + F_1 + F_1 + F_0 = 3$$

```
import java.util.Scanner;

public class Main2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        System.out.println(fib(n));
    }

    static int fib(int n) {
        if (n < 2) return n;
        return fib(n - 1) + fib(n - 2);
    }
}
```



10
55 console

java

方法多載

方法多載(overload)

是指定義同名方法

但參數型別或數量皆不同

程式會根據引數的型別或數量

來決定呼叫哪個方法

```
import java.util.Scanner;

public class Main3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        int c = scanner.nextInt();
        System.out.println(gcd(a, b, c));
    }

    static int gcd(int a, int b) {
        if (b == 0) return a;
        return gcd(b, a % b);
    }

    static int gcd(int a, int b, int c) {
        return gcd(gcd(a, b), c);
    }
}
```

6	8	12
2	console	

java

最大公因數

最大公因數

(greatest common divisor, 簡稱 gcd)

程式實現常使用程式碼簡潔的輾轉相除法

(歐幾里得算法, Euclidean algorithm)

輾轉相除法說明：

若 $a = bq + r$ ，則 $\gcd(a, b) = \gcd(b, r)$

```
static int gcd(int a, int b) {  
    if (b == 0) return a;  
    return gcd(b, a % b);  
}
```

java

1	256 144	144 0	0
3	112 96	144 112	1
	16 最大公因數	32 32	2
		0 餘數為 0	

商數不重要

補充：輾轉相除法證明

1. 已知 $a = bq + r$ ($a, b, r \in \mathbf{N}$)，設 $\gcd(a, b) = g$ ($g \in \mathbf{N}$)

則 $a = mg$, $b = ng$ ($m, n \in \mathbf{N}$)，且 $\gcd(m, n) = 1$

故 $r = a - bq = mg - nqg = g(m - nq)$ 必有因數 g

2. 設 $\gcd(b, r) = pg$ ($p \in \mathbf{N}$)

則 $n = pu$, $(m - nq) = pv = m - puq$ ($u, v \in \mathbf{N}$)

得 $m = pv + puq = p(v + uq)$ 必有因數 p

$\gcd(m, n) = p = 1$ ，故 $\gcd(b, r) = g$

3. 當 $r = 0$ 時 $a = bq$ ，則 $\gcd(a, b) = b = g$

1	256	144	0
	144	0	
3	112	144	1
	96	112	
	16	32	2
		32	
		0	

最大公因數

餘數為 0

簡化程式

Linux 之父林納斯·托瓦茲(Linus Torvalds)曾說過：

*If you need more than 3 levels of indentation,
you're screwed anyway, and should fix your program.*

如果你的程式需要超過三層的縮排，
那麼無論如何你瘋了，並且你需要修復你的程式

而簡化程式可以利用邏輯、語法等來實現

簡化程式

```
import java.util.Scanner;

public class Main1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int start = scanner.nextInt();
        int end = scanner.nextInt();
        if (start >= 2 && end >= 2 && end >= start) {
            for (int i = start; i <= end; i++) {
                boolean isPrime = true;
                for (int j = 2; j * j <= i; j++) {
                    if (i % j == 0) {
                        isPrime = false;
                    }
                }
                if (isPrime == true) {
                    System.out.printf("%d is prime\n", i);
                } else {
                    System.out.printf("%d is not prime\n", i);
                }
            }
        }
    }
}
```

java

```
2 7
2 is prime
3 is prime
4 is not prime
5 is prime
6 is not prime
7 is prime
```

console

```
import java.util.Scanner;

public class Main2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int start = scanner.nextInt();
        int end = scanner.nextInt();
        if (start < 2 || end < start) return;
        for (int i = start; i <= end; i++) {
            if (isPrime(i)) {
                System.out.printf("%d is prime\n", i);
                continue;
            }
            System.out.printf("%d is not prime\n", i);
        }
    }

    static boolean isPrime(int number) {
        for (int i = 2; i * i <= number; i++) {
            if (number % i == 0) return false;
        }
        return true;
    }
}
```

java

簡化程式

```
import java.util.Scanner;

public class Main1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int start = scanner.nextInt();
        int end = scanner.nextInt();
        if (start >= 2 && end >= 2 && end >= start) {
            for (int i = start; i <= end; i++) {
                boolean isPrime = true;
                for (int j = 2; j * j <= i; j++) {
                    if (i % j == 0) {
                        isPrime = false;
                    }
                }
                if (isPrime == true) {
                    System.out.printf("%d is prime\n", i);
                } else {
                    System.out.printf("%d is not prime\n", i);
                }
            }
        }
    }
}
```

java

將肯定條件改為否定條件
避免 **if** 裡太多東西

判斷質數部分脫離主方法
使主方法要做的事更明確

將輸出部分稍微更改形式
並配合新寫法改動

```
import java.util.Scanner;

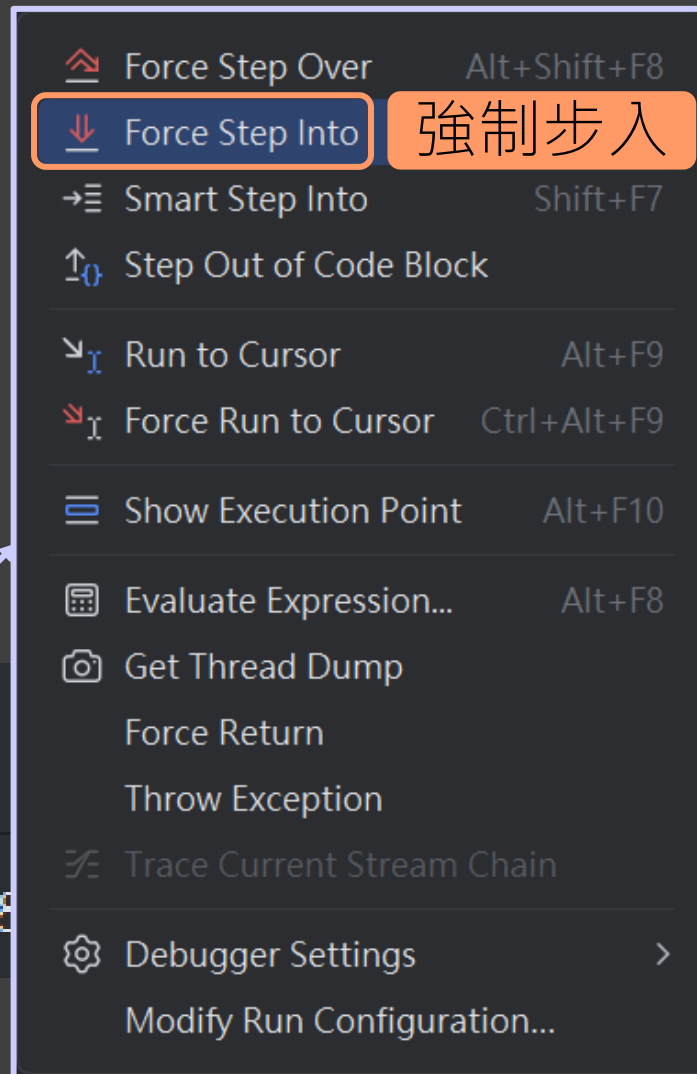
public class Main2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int start = scanner.nextInt();
        int end = scanner.nextInt();
        if (start < 2 || end < start) return;
        for (int i = start; i <= end; i++) {
            if (!isPrime(i)) {
                System.out.printf("%d is prime\n", i);
                continue;
            }
            System.out.printf("%d is not prime\n", i);
        }
    }

    static boolean isPrime(int number) {
        for (int i = 2; i * i <= number; i++) {
            if (number % i == 0) return false;
        }
        return true;
    }
}
```

java

IntelliJ IDEA - 步入和步出

步入(step in)與步過(step over)非常相似
但步過只會在目前的方法中往下執行
而步入會進入呼叫的方法內部並暫停
步出則與步入相反，會跳出當前被呼叫的方法
而若要步入的方法不是自己寫的，則可能需要
使用強制步入(force step into)才可步入



全域變數與靜態欄位

全域變數(global variable)是指
在所有作用域都可以存取(訪問, access)的變數
而同樣的, 因為 Java 是完全物件導向程式語言
所以 Java 中沒有全域變數, 但靜態欄位(field)有相同的效果
欄位是指定義在類別中的變數, 宣告方式與變數一模一樣
也可以加上 final, 而欄位還可以加上存取修飾子和 static
欄位與方法合稱為成員(member)
本次只會介紹無存取修飾子的靜態欄位

```
class Main {  
    資料型別 欄位名稱;  
    資料型別 欄位名稱 = 值;  
  
    public static void main(String[] args) {}  
}
```

java


```
class Main {  
    存取修飾子 final static 資料型別 欄位名稱;  
    存取修飾子 final static 資料型別 欄位名稱 = 值;  
  
    public static void main(String[] args) {}  
}
```

java

靜態欄位

```
2 7
2 is prime
3 is prime
4 is not prime
5 is prime
6 is not prime
7 is prime
total: 4 prime(s) / 2 composite(s)
```

java



```
import java.util.Scanner;

public class Main {
    static int prime_count = 0;
    static int composite_count = 0;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int start = scanner.nextInt();
        int end = scanner.nextInt();
        if (start < 2 || end < start) return;
        for (int i = start; i <= end; i++) {
            if (isPrime(i)) {
                System.out.printf("%d is prime%n", i);
                continue;
            }
            System.out.printf("%d is not prime%n", i);
        }
        System.out.printf("total: %d prime(s) / %d composite(s)",
            prime_count, composite_count);
    }

    static boolean isPrime(int number) {
        for (int i = 2; i * i <= number; i++) {
            if (number % i == 0) {
                composite_count++;
                return false;
            }
        }
        prime_count++;
        return true;
    }
}
```

java