

$$(X, O) = e^{-\frac{x^2}{2\sigma^2}}$$

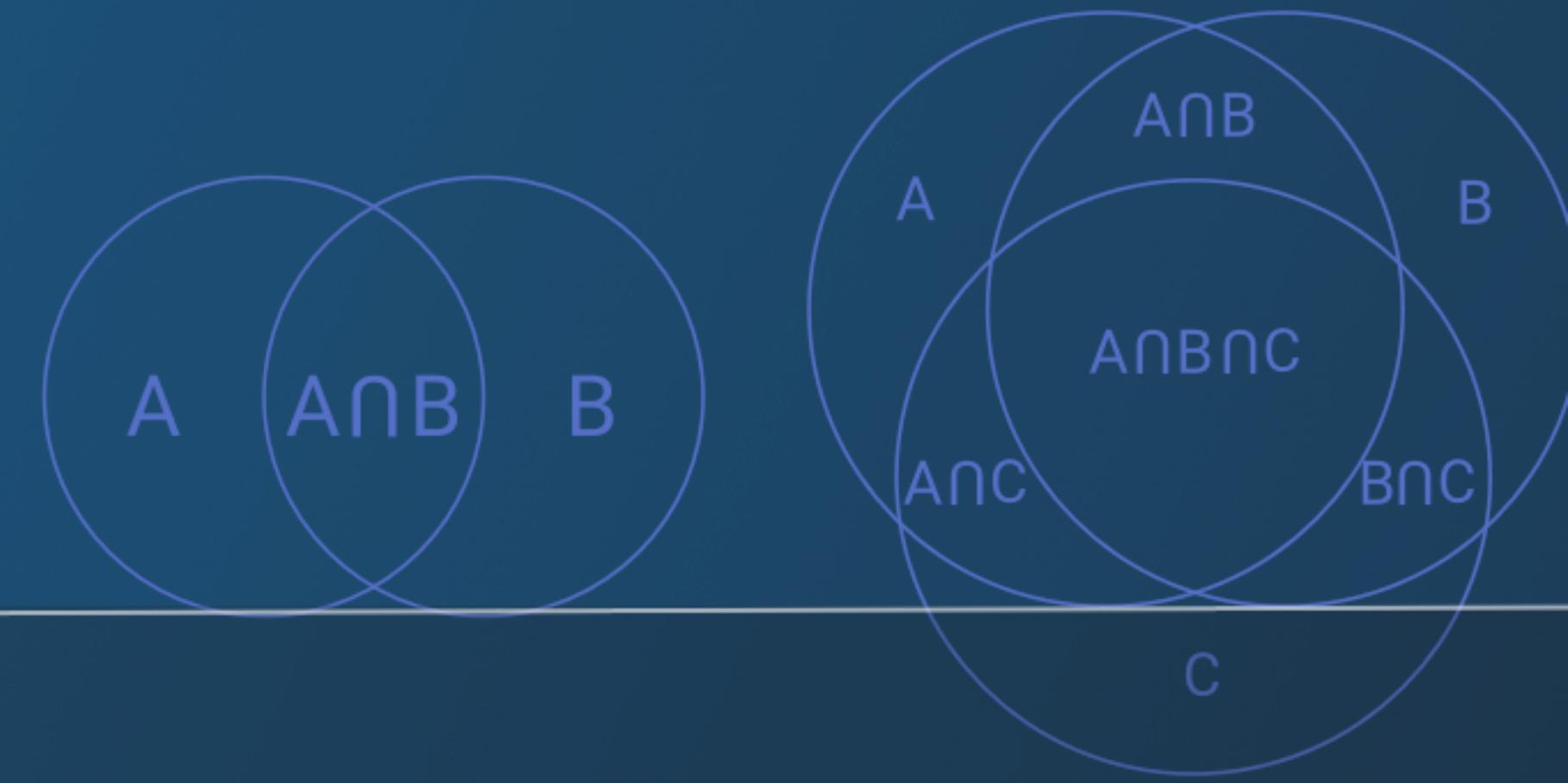
$$x(X, O) = -\frac{x}{\sigma^2} G(X, O) = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$$

$$xx(X, O) = \frac{x^2 - \sigma^2}{\sigma^4} G(X, O) = \frac{x^2 - \sigma^2}{\sigma^4} e^{-\frac{x^2}{2\sigma^2}}$$

$$xxx(X, O) = -\frac{x^3 - x\sigma^2}{\sigma^6} G(X, O) = -\frac{x^3 - x\sigma^2}{\sigma^6} e^{-\frac{x^2}{2\sigma^2}}$$

Julia 程式語言學習馬拉松

Day 06



$$\ln(x + \sqrt{1+x^2}) + x - \frac{1}{x + \sqrt{1+x^2}} \left(1 + \frac{x}{\sqrt{1+x^2}} \right)$$

$$\ln(x + \sqrt{1+x^2}) + x - \frac{1}{x + \sqrt{1+x^2}} \left(\frac{\sqrt{1+x^2} + x}{\sqrt{1+x^2}} \right)$$



cupay 陪跑專家 : James Huang

函式 (Functions)





重要知識點



- 介紹函式的語法，以及如何呼叫、傳入引數，並取得回傳值。
- 介紹匿名 (anonymous) 函式的使用。
- 介紹函式的點運算 (Dot Operation)。
- 有些函式提供了變更引數值的功能，在範例中會為大家介紹。
- 多重分派 (Multiple Dispatch) 是 Julia 的重要特性，我們將看到函式多重分派。
- 變數作用域 (Scope of Variable) 的差異。



Julia 函式 (Functions)



- 透過函式的定義，將相同模式的動作或邏輯，抽象化提取出來成為可以重覆被呼叫使用的模塊。這對於程式的開發、閱讀、測試、維護的工作很有幫助。
- 在官方文件裡裡面，對於函式的定義是：“函式是一個將數組 (tuple) 引數 (argument) 對照到回傳值的物件”。也就是說，呼叫時把引數傳遞給函式，函式內的運算結果再透過回傳值傳回。
- 在使用函式時，必須特別留意變數作用域 (scope) 對於變數值的作用，在後續內容裡面會詳細說明。



Julia 函式 (Functions) 的基本語法與呼叫



```
In [1]: 1 function foo(x, y)  
2     x + y  
3 end  
4  
5 foo(2, 3)
```

Out[1]: 5

```
In [2]: 1 foo(x, y) = x + y  
2 foo(2, 3)
```

Out[2]: 5

```
In [3]: 1 g = foo  
2 g(2, 3)
```

Out[3]: 5

```
In [4]: 1 Σ(x,y) = x + y  
2 Σ(2, 3)
```

Out[4]: 5

- Cell 1: 函式的宣告和呼叫，Julia 的函式宣告是以 `function` 保留字做為開頭，`end` 做為結尾。
- Cell 2: 也可以使用簡潔的表示方式，但是僅能將單一表達式 (expression) 指定給函式。
- Cell 3: 可將函式當做物件指定給另一物件。
- Cell 4: 函式的名稱也可以使用 Unicode 字元。



Julia 函式 (Functions) 的回傳值



```
In [6]: 1 function a(x, y)  
2     return x + y  
3 end  
4  
5 a(2, 3)
```

Out[6]: 5

```
In [7]: 1 b(x,y) = x + y  
2  
3 b(2, 3)
```

Out[7]: 5

```
In [8]: 1 function c(x,y)  
2     return x * y  
3     x + y  
4 end  
5  
6 c(2, 3)
```

Out[8]: 6

- Cell 6: 函式使用 `return` 保留字回傳值。
- Cell 7: 函式的回傳值不一定需要用 `return`，回傳值會是最後一個表達式的結果。
- Cell 8: 如果有 `return` 的話，就以 `return` 的表達式做為回傳值。



Julia 函式 (Functions) 的回傳值



```
In [9]: 1 # multiple return values
2 function d(x,y)
3     i = x * y
4     j = x + y
5
6     return i, j
7 end
8
9 r1, r2 = d(2, 3)
10 println("r1: ", r1)
11 println("r2: ", r2)
```

```
r1: 6
r2: 5
```

```
In [10]: 1 function e(x::Int64, y::Int64)::Float32
2     x + y
3 end
4
5 typeof(e(2, 3))
```

```
Out[10]: Float32
```

- Cell 9: 函式可以有多個回傳值，回傳值以逗號分隔。
- Cell 10: 函式的引數和回傳值均可指定資料型別。



Julia 函式(Functions) 的引數預設值



- 函式的引數可設定預設值，有預設值的引數可視為選用 (optional) 的引數，在呼叫函式時不強制傳入。

```
In [11]: 1 function date(year::Int64, month::Int64=1, day::Int64=1)
2     return year, month, day
3 end
4
5 println("傳入年月日: ", date(2019, 7, 1))
6 println("傳入年, 使用預設的月日: ", date(2019))
```

傳入年月日: (2019, 7, 1)
傳入年, 使用預設的月日: (2019, 1, 1)



Julia 匿名 (Anonymous) 函式



```
In [12]: 1 # 匿名函式的寫法-1  
2 x -> x * 2 * π
```

```
Out[12]: #3 (generic function with 1 method)
```

```
In [13]: 1 # 或是：匿名函式的寫法-2  
2 function (x)  
3     x * 2 * π  
4 end
```

```
Out[13]: #5 (generic function with 1 method)
```

```
In [14]: 1 # 透過匿名函式，計算半徑為 2, 4, 6 的圓周  
2 diameters = [2, 4, 6]  
3 map(x -> x * 2 * π, diameters)
```

```
Out[14]: 3-element Array{Float64,1}:  
12.566370614359172  
25.132741228718345  
37.69911184307752
```

- 函式可以是匿名的，也就是沒有給函式名稱。範例中 Cell 12 或 Cell 13 的寫法都可以。
- 我們可以很方便的把匿名函式當做引數傳入到另一個函式。Cell 14 的範例就是把匿名函式傳給 map 函式，計算出不同半徑的圓周長度。



Julia 函式的點運算 (Dot Operation)



- 在運算子的介紹中，我們曾介紹了運算子的點運算，同樣的在函式也可以使用點運算。
- 以下範例將陣列做為引數傳入，透過函式的點運算，傳回陣列各元素的平方值。

In [15]:

```
1 A = [1.0, 2.0, 3.0]
2
3 function B(x)
4     x ^= 2
5 end
6
7 B.(A)
```

Out[15]: 3-element Array{Float64,1}:

```
1.0
4.0
9.0
```



Julia 函式變更引數值



- 有時候我們希望函式在執行後，也一併變更引數的值，我們會看到在函式名稱後帶“!”。範例中的 sort 函式提供了變更與不變更引數的兩種版本。

```
In [20]: 1 # 不變更 (non-modifying) 版本  
2 v = [3, 1, 2]; sort(v)
```

```
Out[20]: 3-element Array{Int64,1}:  
1  
2  
3
```

```
In [21]: 1 # v 的原始排序並未改變  
2 v
```

```
Out[21]: 3-element Array{Int64,1}:  
3  
1  
2
```

```
In [22]: 1 # 變更 (modifying) 版本，在執行完 sort!() 後 v 的順序也已改變  
2 v = [3, 1, 2]; sort!(v); v
```

```
Out[22]: 3-element Array{Int64,1}:  
1  
2  
3
```



Julia 函式多重分派 (Multiple Dispatch)



Functions Multiple Dispatch Example

```
In [14]: 1 function h(x::Int64, y::Int64)::Int64  
2     println("Int64 版本")  
3     x + y  
4 end  
5  
6 function h(x::Float64, y::Float64)::Float64  
7     println("Float64 版本")  
8     x + y  
9 end
```

Out[14]: h (generic function with 2 methods)

```
In [15]: 1 h(2.0, 3.0)
```

Float64 版本

Out[15]: 5.0

```
In [16]: 1 h(2, 3)
```

Int64 版本

Out[16]: 5

- 有時候相同功能的函式，可能會需要處理不同型別的值，這時候我們可以透過多重分派 (或譯多態分發) 的方式，定義同名但是傳入或回傳不同型別。
- Julia 動態程式語言，會在執行階段 (runtime) 進行判斷。



Julia 變數作用域 (Scope of Variable) 探討



- 變數在程式中依在程式區塊中宣告的位置，而會有不同的作用域，在這邊透過函式的範例，來演示 global scope 和 local scope，以及在函式中的位置的影響。

```
In [23]: 1 # 告訴 global 變數 x 與 y  
2 x, y = 1, 2  
3  
4 function baz()  
5     # 在函式內宣告一個新的 x，這裡的 x 是屬於 local 變數  
6     # 有沒有 local 保留字都可以  
7     local x = 2  
8  
9     function bar()  
10        x = 10          # 賦予 local x 新的值  
11  
12        return x + y  
13        # y 是 global 變數，此 return 值應為 10 + 2  
14        # 其中 10 是 local x 的新值 10  
15    end  
16  
17    return bar() + x # 回傳 bar() 函式傳回值與 local x 相加的值，應為 12 + 10  
18  
19 end  
20  
21 println("baz(): ", baz())  
22  
23 println("global x = $x, y = $y") # global x 與 y 值仍不變
```

baz(): 22
global x = 1, y = 2

知識點 回顧

- 今天的內容介紹了函式的基本語法、傳入值、回傳值的運用，以及匿名函式的使用。
- 點運算可以讓我們很方便地將函式向量化 (Vectorizing Functions)。
- Day 5 時有介紹過運算子就是函式，大家可以回顧先前的範例。
- 多重分派 (Multiple Dispatch) 在後續的內容中會有更深入的介紹。
- Julia 提供很多內建的函式，讓開發者不需要從頭開發所有的功能，在接下來的內容和範例中，我們也會陸續介紹常用的函式。



推薦閱讀

- 官方文件 Functions
- Julia面向對象（多重派發）





解題時間

請跳出 PDF 至官網 Sample Code
& 作業開始解題