第 1 週，第 2 課：解構第 1 課

目標

— 解構第 1 課中的代碼
— 提供有關 Julia 中表示式的細節
— 傳達有效代碼的重要性
— 開始仔細學習 Julia 語言的過程

學完本課，您將能夠

* 舉幾個 Julia 中有效表示式的例子
* 根據名稱和值解釋變數的結構
* 準確解釋 Julia 中的字串值是什麼
* 廣義地解釋什麼是函數，以及 Julia 如何識別函數
* 解釋運算子與其他類型函數的區別
* 舉例說明一些 Julia 分隔符號及其用法

複習：第 1 課的代碼

在第 1 課中，您在 REPL 中運行了以下代碼

```
mystringexample1 = "Hello, world"          … 行 1
println(mystringexample1)                  … 行 2
```

您還建立了一個檔案，我們將其稱為 `myfile.jl`，儘管您可能使用了不同的名稱[1]。

```
include("myfile.jl")                       … 行 3
```

解構第 1 行，它建立了一個變數

如何閱讀第 1 行：它有一個左手邊，一個右手邊，用等號連接它們。

[1] 在本課中，每當我們提到 `myfile.jl`，請注意我們指的是您建立的文件，並在您的腦海中（以及在代碼範例中）將其替換為您自己的檔案名。

右側是一個字串值（更多內容見下文）：`"Hello, world"`。

左側是變數的名稱：`mystringexample1`。

`=` 符號將變數名稱綁定到給定的值。

這實際上改變了電腦中的一些記憶體。就技術上來說，等號是一種特殊的函數，即運算子，它的全稱是 “指派運算子”[2]。

我們說第 1 行建立了變數，因為在第 1 行之前 `mystringexample1` 不是命名空間的一部分。我們可以指派一個新值，例如使用此行

```
mystringexample1 = "a new value"
```

在這種情況下，我們不會建立變數，只是將現有名稱綁定到新值。這個新值不必是字串，順便說一下，它可以是一個數字

```
mystringexample1 = 1.1111
```

或者實際上是 Julia 可以使用的任何其他類型的值。如第 1 課所述，名稱不能從命名空間中刪除，只能新增。在本課程中，我們僅使用最上層命名空間，該命名空間的存在與你的 Julia 會話一樣長。

解構字串值

字串是一個字元序列。如你所見，在 Julia 中，我們透過將字串值括在一對雙引號中來表示它[3]。

我們之前已經簡要討論過字元。請注意，在 Julia 中，字元總是用單引號括起來。

[展示：`a = 'a', b = "a"`]

在本課程中，我們將僅使用國際鍵盤的一次按鍵[4]可用的字元來形成字串。但是更多的字元在 Julia 中是有效的[5] 稍後我們將簡要向你展示如何輸入它們。你可能希望開始在變數和函數名稱中使用它們，但學習這樣做是你自己的專案，它不是本課程的一部分。

解構變數名稱

只有部分在 Julia 字串值中可用的字元被允許作為變數名稱。

[2]所有電腦語言都至少有一個指派運算子。它們在影響電腦記憶體的細節方面有所不同，但在本課程中，我們先遠離這些細節。你需要知道的是，在 Julia 中，指派運算子將其右側的值綁定到其左側的變數名稱。

[3]Julia 有很多種類的值；在本課程中，我們只討論其中的幾個。請參閱第…課中對類型的討論。

[4]可能透過 Ctrl 或 Tab 鍵修改

[5]包括希臘語、阿拉伯語、梵語等字母表。事實上，在 Julia 中，不僅字母字元而且非字母字元（例如用於書寫普通話的字元）都是有效字元。

變數名稱必須以字母開頭[6] 並且必須以字母、數字或底線或驚嘆號接續。在本課程中，我們僅使用羅馬字母表中的字母，但 Julia 接受的字母多於這些字母。

最佳做法是僅使用小寫字母和數字，並使用描述性名稱，例如 `mystringexample1`[7]。

第 1 行是一個有效的 Julia 表示式

這一點非常重要：當 Julia 處理有效代碼時，電腦會發生變化—其中一些變化達到了代碼的目的 (列印輸出、計算、圖片 …)。第 1 行是有效的，因為：

- 第 1 行包含 Julia 能識別的符號。
- 這些符號組合成三個有效群組。
- 這三個有效群組組合成一個有效表示式。

問題：Julia 能識別哪些符號？
答：非常多！但在本課程中，我們將只使用標準國際鍵盤上可見的符號，所有這些符號 Julia 都能識別。

問題：第 1 行中哪些是有效的符號群組？
答：三個，分別是 `mystringexample1`、`=` 和 `"Hello, world"`。也就是說，Julia 識別出一個有效的變數名稱、一個有效的運算子和一個有效的字串[8]。

問題：為什麼這個有效符號群組的組合是一個有效的表示式？
答：因為 `=` 運算子可以這樣運作。實際上，它只能這樣運作：左邊的名字，中間的 `=` 和右邊的值。

只有當表示式的所有部分都被 Julia 正確識別並按照 Julia 的規則組合時，我們才有一個有效的 Julia 代碼表示式。生成有效代碼的規則非常嚴格，我們將在第 3 課中討論為什麼會這樣。

最後，讓我們注意第 1 行的某些部分本身就是有效代碼，即名稱和值。一個有效的表示式可以是一些更大的有效表示式的一部分。

評估無效代碼會產生錯誤訊息

這裡有一些行話：我們說 Julia 評估它得到的每個表示式。這僅僅意味著 Julia 會嘗試執行代碼指示它執行的操作。

例如以下幾行無效代碼

[6]以及其他一些字元，但尤其不是數字，請參閱 Julia 文件了解詳細資訊。

[7]Julia 社群通常遵守此規則。這使得驚嘆號具有非常特殊的含義，我們稍後會看到。儘管規則允許，但強烈不鼓勵在用戶代碼中使用底線；這個想法是將底線限制為 Julia 內部的特殊含義。

[8]之間的空格不是必需的，正如 Julia 從其他線索中識別的那樣。然而，空格的存在/不存在有時在 Julia 中確實很重要，我們稍後會看到。

```
=
= "Hello, world"
mystringexample1 =
mystringexample1 "Hello, world"
"Hello, world" = mystringexample1
```

從 Julia 產生錯誤訊息，僅此而已[9] 。在第 3 課中，我們將開始閱讀錯誤訊息和除錯無效代碼。

一個微妙的解釋

然而，這可能會讓你大吃一驚，

```
Hello, = mystringexample1
```

是有效代碼。這不是因為 `Hello,` 是一個有效的變數名稱，而是因為逗號的特殊作用。在這裡，因為它遵循指派運算子左側的有效名稱，所以逗號表示 Julia 應該執行多重指派。讓我們用一個左邊有兩個名字的例子：

```
Hello, world = mystringexample1                    … 行 4
```

請注意，第 4 行引入了一種新的指派形式：右邊不是一個值，而是一個變數名稱。沒問題，Julia 只是使用綁定到變數名稱的值。

展示：我們評估第 4 行，詢問它建立的變數名稱和值

多重指派的運作方式如下：在 `=` 的左側，變數用逗號分隔[10]。獲得列表後，Julia 會在右側查找值。單個字串值可以提供多個單獨的值可能看起來很奇怪，但請記住，字串是一個字元序列。由於左側是一個序列，Julia 將右側視為一個序列。如你所見，右側的多餘項被忽略。

以這種方式使用逗號進行多重指派是 Julia 允許你建立非常緊湊且通常也很容易閱讀的代碼的方式之一。如果做得好，它確實可以幫助你編寫其他人喜歡閱讀的代碼，這可以極大地簡化協作。這包括與自己合作，幾個月或幾年後，當你嘗試使舊代碼適應新用途時。

> 不用擔心在 Julia 中編寫有效表示式的難度。正如你在第 1 課中看到的，這很容易。與任何電腦語言一樣，Julia 允許極長的有效表示式。是的，要確保這樣的表示式有效可能非常困難。但這一切都無關緊要。你可以用簡短的表示式做大量的事情！
> 而且你不必一次學習所有的 Julia。在本課程中，我們將逐步向你介紹越來越多的有效表示式形成方法，任何時候都不會太多。

第 2 行呼叫一個函數

當 Julia 代碼告訴一個函數做某事時，我們說它呼叫了這個函數。這裡我們使用一些輸入呼叫函數 `println`。這輸入是 `mystringexample1`，即變數的名稱。

[9] 這不能完全保證，但絕對是 Julia 的創造者想要的！
[10] 如你所見，你可以只有一個變數後跟逗號，然後是 = 符號。

要使此代碼成為有效代碼，`println` 必須能夠格式化該變數的值[11]。在 Julia 中呼叫函數會使事情發生。

你應該思考一下：函數呼叫`println(mystringexample1)`做了幾件事：首先它接受變數名稱，然後獲取變數的值，然後格式化它—在這種情況下，幾乎沒有什麼可做的—然後它會在螢幕上列印格式化的字串，然後在下一個`julia>`提示符之前顯示一個空行。

Julia 是如何知道一段代碼引用了一個函數的？很簡單：代碼是一個有效的變數名稱，後面緊跟括號— 也就是說，函數名稱後面沒有任何空格，下一個字元是`(`。之後是函數的輸入，輸入之後是關閉的`)`。

然而，`println` 實際上並不需要任何輸入：[展示：`println()`, `include()`]。你會看到 `println()` 的行為就好像你給了它一個空字串：它列印一行空的內容，然後跳到新行，然後跳到 `julia>` 提示符. 另一方面，`include()` 會引發錯誤。

規則有一個例外，即 Julia 透過名稱後面的 `(` 識別函數。正如我們所指出的，運算子是一種特殊的函數。它們（大部分）是數學符號，例如 `-`、`+`、`>` 以及它們形成的表示式具有非常接近通常數學含義的規則[12]。在本課程中，你將學習更多的 Julia 內建函數，並且你還將編寫自己的。很大一部分 Julia 代碼是透過函數編寫的。

最終解構：分隔符

我們在 `println()` 中使用的括號在 Julia 中起著重要作用：它們是分隔符。它們用於告訴 Julia 函數的輸入在哪裡開始和結束。同樣，字串周圍的雙引號是分隔符，逗號用於進行多重指派時也是如此。

請注意你輸入的內容以產生有效代碼：使用有效字元，你輸入了值、名稱、運算子和分隔符。這涵蓋了我們在本課程中使用的所有有效代碼[13]。

回顧與總結

- Julia 代碼由有效的表示式組成。
- 在第 1 週，我們使用包含值、名稱、運算子和分隔符的有效表示式。
- Julia 中的名稱必須以字母開頭，並以字母或數字或底線或驚嘆號接續。
- 變數是綁定到值的名稱。
- 呼叫函數意味著在其名稱後面加上括號，將傳遞給函數的值和/或變數名稱括起來。
- 運算子是一種不需要括號的特殊函數—通常它們是像 `=`、`+`、`<` 等符號。
- 字串值周圍的 `" "` 對，和函數輸入周圍的括號 `( )` 等分隔符有助於構建 Julia 代碼。

[11] 像 `"Hello, world"` 這樣的普通字串是所有值中最容易格式化的。
[12] 但請注意：數學含義只是一個指示。必須嚴格遵守規則，所以你必須知道它們！
[13] 如前所述，我們僅使用 Julia 中有效字元的一小部分。同樣，我們只使用了 Julia 的一些運算子和分隔符。