Lesson 4, Week 1: Programming is applied formal logic
第 1 週，第 4 課：程式設計是應用的形式邏輯

This is the only theory lesson in the whole course. As such, it is very, very important!
這是整個課程中唯一的理論課。因此，這是非常非常重要的！

AIM
目標

— to understand the difference between logic and formal logic
了解邏輯和形式邏輯的區別

— to understand how formal logic makes computer language possible
了解形式邏輯如何使電腦語言成為可能

After this lesson, you will be able to
完成本節課後，你將能夠

* explain what we mean by the word logic
解釋我們所說的邏輯這個詞的意思
* explain the difference between logic and formal logic
解釋邏輯和形式邏輯的區別
* explain why any computer must embody formal logic
解釋為什麼任何電腦都必須體現形式邏輯
* give a small example of a language that applies formal logic
舉一個應用形式邏輯的語言的小例子
* explain why writing computer programs is expressive, in the same way as writing poems or music
解釋為什麼編寫電腦程式是具有表現力的，就像寫詩或音樂一樣

Why this lesson?
為什麼要上這節課？

Programming often feels very strange to beginners, and if they haven't worked much with mathematical formulae, it can feel utterly alien.
對於初學者來說，通常會對程式設計感到非常奇怪，如果他們也沒有接觸太多數學公式，那會完全陌生。

This lesson cannot make the initial experience any less weird. Weird is part of the territory. But by explaining that it isn't arbitrary, nor aimed at keeping people out, but instead an essential aspect of any computer language, I hope to encourage you to persist until the Julia way of writing programs starts to feel natural. At that point, you will have learnt a new languge.
本課程無法使初始體驗變得較不怪異。怪異是這個領域的一部分。但是，藉由解釋它不是任意的，也不是旨在拒人於外，而是任何電腦語言的基本，我希望鼓勵你堅持下去，直到對 Julia 的寫作方式開始感到自然。那時，你將學會一個新的語言。

That is entirely serious: Julia, as I hope you will see below, is as rich and expressive in its own right as any natural language, such as Spanish, Mandarin and !Kung. Apart from implementing formal logic, there is no real difference with other languages. You could use it to write advertisements, love letters, novels, whatever! But mostly, since the formal logic can be implemented on a machine, you use it to write programs that make machines do things you want them to do.
這完全是認真的：Julia, 就如我希望你能在如下看到，它本身就是豐富和有表現力的，就像任何自然語言，例如西班牙語，普通話和 !Kung 語, 除了實作正式邏輯外，和其他語言其實沒有真正的區別。你可以使用它來寫廣告，情書，小說等等！但是，大多數情況下，由於可以在電腦上實現形式邏輯，因此你可以使用它編寫讓機器做你想要它們做的事情的程式。

Logic: the study of correct forms of reasoning
邏輯：對正確推理形式的研究

Reasoning is part of everyday life and language. It is simply the process of trying to make sense of things and expressing that as clearly as possible.
推理是日常生活和語言的一部分。這只是試圖理解事物並儘可能清楚地表達的過程。

Unfortunately, there are many ways people use to make sense that actually are mistakes in reasoning. For example, it makes sense that raining wets grass, and this may lead people to reason as follows: if the grass is wet, there has been rain. Although this may often be true, it isn't always true: rain is not the only way grass can get wet. Strictly speaking, it is not correct. Reasoning correctly is quite hard!
不幸的是，人們用來理解的很多方法實際上在推理上是錯誤。例如，雨讓草濕了是有意義的，這可能會導致人們推理如下：如果草濕了，那就下雨了。儘管這通常可能是正確的，但這並不總是正確的：雨不是草濕的唯一途徑。嚴格來說，這是不正確的。正確推理非常困難！

As far we know, the first people who became aware of this *and* made a study of correct forms of reasoning lived approximately 100 generations ago. This occurred separately, in at least three places: in ancient China, ancient Greece and ancient India rules were formulated for reasoning correctly.
據我們所知，最早意識到這一點**並**研究了正確的推理形式的人距今大約 100 個世代以前。這是分別發生的，至少在三個地方：在中國古代，古希臘和古代印度，規則是為了正確推理而制定的。

The word logic is used in many different ways in ordinary language. In this course, we use it in only one sense: the study of correct forms of reasoning.
邏輯一詞在普通語言中以許多不同的方式使用。在本課程中，我們僅在一種意義上使用它：正確的推理形式的研究。

Formal logic: using symbolic formulae to study and apply logic
形式邏輯：使用符號公式來研究和應用邏輯

Formal logic is simple to define: it uses symbols and formulae to study reasoning. This is fairly new discipline, it started only about 6 generations ago[1].
形式邏輯很容易定義：它使用符號和公式來研究推理。這是相當新的學科，它僅開始於大約 6 個世代以前[2]。

The aim of making logic formal was absolute precision in reasoning and absolute certainty about its correctness. It aims at this by using rules similar to the rules of algebra[3]. By doing so, it reduces logical analysis to calculations which can be independently checked. In other words, it removes the need for intuition and insight from logical analysis.
製作邏輯形式的目的是推理的絕對精確性和關於其正確性的絕對確定性。它透過使用類似於代數規則的規則來實現這一目標[4]。透過這樣做，它將邏輯分析減少到成為可以獨立檢查的計算。換句話說，它從邏輯分析中消除了對直覺和見解的需求。

Formal logic has the strange property that something obvious and something baffling can sit side by side. For an example of the obvious, see the NOT, AND and OR truth tables below. Unfortunately, we you'll have to take our word for for the ease with which baffling things come up quite unexpectedly (or read *Logicomix*, in particular pages 164 to 168, about paradoxes).
正式的邏輯具有奇怪的屬性，即明顯的事情和令人困惑的東西可能並排坐。有關明顯的示例，請參見下面的 NOT, AND 及 OR 的真值表。不幸的是，你必須相信我們所說，以緩和令人困惑的事情出乎意料地出現 (或閱讀 **Logicomix**，特別是第 164 至 168 頁，關於悖論)。

Truth tables for `NOT`, `AND`, `OR`
`NOT`, `AND`, `OR` 的真值表

In two-valued logic, which is by far the most common type of formal logic and also what Julia uses, there are two constant values, we denote them with `true` and `false` [5]. A statement is then something that is either `true` or `false`. As noted above, for a formal logic we need a symbol to stand for a statement, for example we can use P to stand for the statement "The earth spins on its axis" and Q for the statement "The earth is round"[6].
在兩值邏輯中，這是迄今為止最常見的形式邏輯類型，也為 Julia 所使用，有兩個常數值，我們用 `true` 和 `false` 表示它們[7]. 一個陳述則會是真 (`true`) 或假 (`false`)。如上所述，對於形式邏輯，我們需要一個符號來代表陳述，例如，我們可以使用 P 來代表陳述「地球在其軸上旋轉」及 Q 用於陳述

[1] See *Logicomix* by Doxiadis and Papadimitrou for an excellent and very human account of some of the most important episodes in the early history of formal logic.

[2] 參見 Doxiadis 和 Papadimitrou 的 **Logicomix**，以了解形式邏輯早期歷史上一些最重要情節的出色和非常人性化的話題。

[3] This goal has been considerably refined, because it was shown that a system cannot be consistent as well as complete and applicable to all of mathematics. For computers, it is not necessary to produce *all* correct reasoning—it is enough to ensure that all the reasoning in your computer is completely correct.

[4] 這個目標已經得到了很大的改進，因為它顯示系統無法一致，並且都適用於所有數學。對於電腦，沒有必要產生**所有**正確的推理—足以確保電腦中的所有推理都是完全正確的。

[5] In Julia code they are the values `true` and `false`.

[6] You may disagree with either of these; the first is much clearer than the second. Of course the earth is not perfectly round, but for practical purposes the imperfections are negligible, so it is perfectly reasonable to say that Q is true. This sort of thing is why ordinary language and everyday reasoning are very hard to represent in a formal language.

[7] 在 Julia 代碼中，它們是值 `true` 和 `false`。

「地球是圓形的」[8]。

So now we have two symbols P and Q. Here are three more: $\neg$, $\wedge$, $\vee$. With these, lets look at four valid formulae: $\neg P$, $\neg Q$, $P \wedge Q$, $P \vee Q$.
因此，現在我們有兩個符號 P 和 Q。這裏還有三個：$\neg$，$\wedge$，$\vee$。透過這些，讓我們來看四個有效的公式：$\neg P$，$\neg Q$，$P \wedge Q$，$P \vee Q$。

In words, these are "The earth does not spin on its axis", "The earth is not round", "The earth spins on its axis and the earth is round", and "The earth spins on its axis or the earth is round". That is, $\neg$ means `NOT`, $\wedge$ means `AND` and $\vee$ means `OR`.
用言語說，這些是「地球不在其軸上旋轉」，「地球不是圓形的」，「地球在其軸上旋轉，且地球是圓形的」，「地球在其軸上旋轉，或地球是圓形的」。也就是說，$\neg$ 表示 `NOT`，$\wedge$ 表示 `AND`，$\vee$ 表示 `OR`。

But what do these mean, in terms of formal logic? Well, they (may) change the truth values. For example, if P is true then $\neg P$ must be false. A good way to summarise exactly what they mean is by means of truth tables. In a truth table, we specify the truth value of all valid combinations.
但是，就形式邏輯而言，這些意味著什麼？好吧，他們（可能）改變真實價值觀。例如，如果 P 為 true，則 $\neg P$ 必須為 false。透過真值表來總結它們的含義是一種好方法。在真值表中，我們指定了所有有效組合的真值。

So the truth table for $\neg$ (equivalently, `NOT`) is as follows (note that *two* lines are needed to get both possibilities):
因此 $\neg$（等效地，`NOT`）的真值表如下（請注意，需要**兩**行以獲得這兩種可能性）：

P	$\neg P$
true	false
false	true

The truth table for $\wedge$ (equivalently, `AND`) requires four lines:
$\wedge$（等效地，`AND`）的真值表需要四行：

P	Q	$P \wedge Q$
true	true	true
true	false	false
false	true	false
false	false	false

In other words, the combined forumula $P \wedge Q$ is `true` only when both are `true`, and `false` otherwise.
換句話說，組合的公式 $P \wedge Q$ 僅當兩者都是 `true` 才會是 `true`，其他則會是 `false`。

The truth table for $\vee$ (equivalently, `OR`) is:
$\vee$（等效地，`OR`）的真值表是：

P	Q	$P \vee Q$
true	true	true
true	false	true
false	true	true
false	false	false

In other words, the combined forumula $P \vee Q$ is `false` only when both are `false`, and `true` otherwise.
換句話說，組合的公式 $P \vee Q$ 僅當兩者都是 `false` 才會是 `false`，其他則會是 `true`。

Because they may change truth values, `NOT`, `AND` and `OR` are called operators. To be more specific, they

[8] 你可能會不同意其中的任何一個。第一個比第二個要清楚得多。當然，地球不是完全圓形的，但是出於務實目的，這種不完美是可以忽略不計的，因此說 Q 是正確的，這是完全合理的。這種事情就是為什麼普通語言和日常推理很難用形式語言表示的原因。

are logical operators.
因為他們可能會修改真值，所以 NOT、AND 和 OR 被稱為運算子。更具體地說，他們是邏輯運運算子。

The symbols we've seen are standard for research papers in formal logic, but not in progamming. In Julia, the symbol for NOT is `!`, AND is `&&` and OR is `||` . We'll see these again later; the fact that we need two characters make one symbol is of no significance.
我們所看到的符號是形式邏輯研究論文的標準，但沒有用在程式設計上。在 Julia 中，NOT 的符號為 `!`，AND 是 `&&` 和 OR 是 `||`。我們稍後會再看到這些；我們需要兩個字元構成一個符號是無關緊要的。

Why formal logic needs a formal language
為什麼形式邏輯需要形式語言

For formal logic to work, every expression must consist of symbols. In fact, it must consist of symbols that formal logic recognises—in exactly the same way that every expression in Julia must consist of symbols that Julia recognises[9].
為了讓形式邏輯運作，每個表示式都必須由符號組成。實際上，它必須由形式邏輯認可的符號組成—與 Julia 中的每個表示式完全一樣同時也為 Julia 所認可[10]。

A formal language makes it possible to say exactly what symbols and expressions are valid in that language. It has to go beyond the three operators we've seen above. That is, in order to express something, one has to use symbols that can carry truth values. Above, we used P and Q for that, but that was for convenience only. There are much better ways, and Julia is one of those.
一種形式的語言可以準確地說出哪些符號和表示式在該語言中有效。它必須超越我們上面看到的三個運算子。也就是說，為了表達某些內容，必須使用可以承載真值的符號。上面，我們為此使用了 P 和 Q，但這僅是出於方便。有更好的一些方法，Julia 就是其中之一。

Let's repeat that: Julia is a formal language[11]. It uses characters to form values and names, and then it uses delimiters and operators to form the names and values into expressions. The rules of the formal logic that governs Julia then determine whether the expression is valid or not.
讓我們重複一遍：Julia 是一種形式語言[12]。它使用字元組成值和名稱，然後使用定界符和運算子將名稱和值組成表示式。這些形式邏輯規則主宰著 Julia ，用以決定表示式是否有效。

In other words, you can write an unbelievably large number of Julia programs, but in order for them to consist of valid expressions, you have to follow the rules of Julia's formal language and formal logic.
換句話說，你可以編寫令人難以置信數量的 Julia 程式，但是為了讓它們由有效的表示式組成，你必須遵循 Julia 的形式語言和形式邏輯的規則。

Formal languages can generative and make computers possible
形式語言可以是生產的並使電腦成為可能

[9] Introductions to formal logic have to solve a bootstrap problem: until you know the symbols, you can't do formal logic. They tend to start as we did here: by translating some very simple sentences into symbols. This has the effect of making the very simple seem complicated. Weird, but very hard to avoid.
[10] 對形式邏輯的介紹必須解決一個前導問題：在您知道符號之前，您無法進行正式邏輯。它們傾向於像我們在這裡一樣開始：將一些非常簡單的句子轉換為符號。這會有讓非常簡單的反而顯得複雜的效果。很奇怪，但是很難避免。
[11] In fact, every computer language is a formal language.
[12] 實際上，每種電腦語言都是形式語言。

By "generative" I mean that one can make new expressions from old ones. A useful language with only a few valid expressions can certainly exist, but Julia and all major computer languages allow a great many valid expressions, and also set no limit on how many such expressions are used to write a program.
用 "生產的" 一詞我的意思是，人們可以從舊的表示式中製作出新的表示式。只有幾個有效表示式的有用語言肯定也是存在的，但是 Julia 和所有主要的電腦語言都允許很多有效的表示式，也沒有對使用多少此類表示式來編寫程式進行限制。

Writing a poem is also the result of a generative process: you add one word after another, one line after another, and there is no set limit on how long your poem may be.
寫一首詩也是生產過程的結果：你新增一個又一個單詞，一行又一行，並且對你的詩也沒有設定的長度限制。

But a formal language is not just generative: because the rules are completely fixed and explicit, they can be mechanised. They can be made to be part of a machine[13]; in such a machine you have a way to represent the values `true` and `false`. Then when you apply `not` to a `true` you get the value `false`, and so on. Such a machine can produce and evaluate all the expressions of a formal language and its the formal logic.
但是形式語言不僅是生產的：由於規則是完全固定和明確的，因此可以機械化。他們可以成為機器的一部分[14]; 在這樣的電腦中，你有一種表示值 `true` 和 `false` 的方法。然後，當你將 `not` 應用於 `true` 時，你將獲得值 `false`，依此類推。這樣的機器可以生產和評估形式語言及其形式邏輯的所有表示式。

At the level of a microchip, all modern electronic digital computers contain many millions of tiny electrical circuits that are nothing but electronic versions of `NOT` , `AND`, `OR`[15]. Formal logic is what makes a computer possible; all modern computers are applications of formal logic.
在晶片的層級上，所有現代的電子數位電腦都包含數百萬個微型電路，這些電路不過是 `not`、`and`、`or` 的電子版本[16]。形式邏輯是使電腦成為可能的原因。所有現代電腦都是形式邏輯的應用。

A formal language with just a few letters and rules
只有幾個字母和規則的形式語言

Let us look at brief example of a formal language, let's call it AdHoc1. It will be very simple, just a few letters and rules. They allow the formation of some sentences in English, but also of expressions that are very far from being English. That is, the valid expressions in this language include some English but not all of it, and includes non-English also. To specify AdHoc1, we give a full list of symbols and a full list of how rules for building expressions.
讓我們看看形式語言的簡短範列，讓我們稱其為 AdHoc1。這將非常簡單，只有幾個字母和規則。它們允許組成一些英語句子，但也允許表達遠非英語。也就是說，這種語言的有效表示式包括一些英語，但並非全部，也包括非英語。為了定義 AdHoc1，我們提供了符號的完整列表，並提供了建構表示式規則的完整列表。

The symbols are a few characters: the letters `a c e i f h l n s t` and the space character.
符號是幾個字元：字母 `a c e i f h l n s t` 和空格字元。

[13] The earliest programmable computer design was by Charles Babbage over 150 years ago, for a machine to be driven by steam.
[14] 最早可程式設計的電腦設計是由 150 年前的查爾斯·巴巴奇 (Charles Babbage) 設計的，是一台由蒸汽驅動的機器。
[15] Of course, they contain much more than just these operators, but this is not a course in computer design.
[16] 當然，它們包含不僅僅是這些運算子，但這不是電腦設計的課程。

The rules are
規則是

1. Some letters are also vowels, they are: `a e i`.
   有些字母也是母音，它們是：`a e i`。
2. Some letters are also consonants, they are `c f h l n s t`.
   有些字母也是子音，它們是 `c f h l n s t`。
3. An expression must start with a consonant.
   表示式必須從子音開始。
4. An expression may not contain two vowels next to each other.
   表示式不能包含兩個連續母音。
5. Any expression can have a character added to it, except that
   任何表示式都可以再新增一個字元，除了
   (a) No expression can have more than 100 characters
   表示式不可超過 100 個字元
   (b) No expression can end with the space character
   表示式不能以空格字元結束

Note that some expressions made with AdHoc1 look like sentences in English, for example ”that is it” and ”that is it i think”. But strangely enough, ”i think that is it” is *not* a valid expression in AdHoc1, because it violates rule 3. Some amusing expressions are possible, such as ”the think that i think that i think that i that think thinks”. Also the famous paradox ”this sentence is false” is valid AdHoc1.
請注意，某些用 AdHoc1 寫成的表示式看起來像英語的句子，例如”that is it” 和”that is it i think”。但是奇怪的是，”i think that is it” 不是在 AdHoc1 中的有效表示式，因為它違反了規則 3。可能會有一些有趣的表示式，例如”the think that i think that i think that i that think thinks”。同時著名的悖論”this sentence is false” 也是有效的 AdHoc1。

However, by reading that last sentence as a paradox we are reading it as English. This is not part of AdHoc1, and requires an extension that applies AdHoc1 expressions to English. But most AdHoc1 expressions are not English, for example ”ticilllllll eee”, so this extension is very informal[17] and relies a good deal on us as people who can read English.
無論如何，透過將最後一句話作為悖論，我們將其解讀為英語。這不是 AdHoc1 的一部分，並且需要將 AdHoc1 表示式擴展以應用於英語。但是大多數 AdHoc1 表示式不是英語，例如” ticilllllll eee”，因此此擴展是非常不正式的[18]且很依賴可以閱讀英語的我們。

There are some other patterns to note (or oddities, if you prefer): you cannot form the empty expression (because of rule 3), you can’t write more than one line (there is no newline character), there are no delimiters so you cannot do punctuation. All one has so far with with AdHoc1 is a a set of rules for making
還有其他一些模式要注意 (或怪事，如果你喜歡如此說): 你不能組成空表示式 (由於規則 3), 你不能寫超過一行 (沒有換行字元)，沒有定界符，因此你不能做標點。到目前為止，所有人在 AdHoc1 有的是一組規則來製作

[17] Actually, no formal language exists that produces all the valid sentences in English and only those sentences. This is partly because people cannot agree on what all the valid sentences in English are.
[18] 實際上，沒有形式語言可以用英語產生所有有效的句子且全部僅有這些句子。部分的原因是因為人們無法就英語的所有有效句子達成共識。

Let us also note that AdHoc1 cannot be a computer language, because it has no way of making the computer do anything. For that, there needs to be a number of expressions in the language that cause something to happen inside the computer. An example in Julia is the `println` function. In fact both lines of code in your first computer program makes something happen in or via the computer. Most of the details of this is hidden from the user of Julia—and this is one of the reasons for choosing Julia as a first programming language to learn.
我們同時還要注意 AdHoc1 不能是電腦語言，因為它無法讓電腦做任何事情。為此，語言中需要有許多表示式，這些表示式能讓電腦內發生某些事情。Julia 中的一個範列是 `println` 函數。實際上，你的第一個電腦程式中的兩行代碼都可以在電腦中或透過電腦中發生。大多數的細節對 Julia 的使用者都隱藏起來了—這是學習電腦語言時選擇 Julia 作為第一個語言的原因之一。

Programming, poetry and truth
程式設計，詩歌和真理

> 'Beauty is truth, truth beauty,—that is all
> Ye know on earth, and all ye need to know.' (Keats)
> ' 美即是真，真即是美，
> —這是你們在世上所知道的和應該知道的一切。' (濟慈)

As we have repeatedly seen, a programming language like Julia allows you make many, many expressions. When you choose what names your variables are to have, which functions to employ, how to structure the logical operators in your program, and so on, you are using the language to express yourself. You are doing much the same as a poet.
正如我們一再看到的那樣，Julia 這樣的程式設計語言讓你可以做出許多許多表示式。當你選擇要如何命名變數，要使用什函數，如何建構程式中的邏輯運算子等等時，你正在使用語言來表達自己。你做的與詩人大致相同。

Moreover, you cause things to happen: certain numbers are calculated, pictures are drawn, documents are printed. A poet aims at different effects: reactions from readers/listeners, memorable descriptions, catching the mood. But like a poet, the programmer is using their skill to achieve definite purposes.
此外，你讓一些事情發生：計算某些數字，繪製圖片，並列印文件。一位詩人的目標是不同的影響：讀者 / 聽眾的反應，令人難忘的描述，引發情緒。但是像詩人一樣，程式設計師正在利用他們的技能來實現明確的目的。

Let us not forget truth: it is central to logic, as we saw above. Computer programs utterly rely on the programmer keeping a very strict eye on whether their logic maintains truth. Paraphrasing Keats, one might say: "Code is truth, truth code,—that is all you need at the keyboard".
讓我們不要忘記真理：正如我們在上面看到的那樣，這對於邏輯是核心。電腦程式完全依靠程式設計師非常嚴格地關注他們的邏輯是否保持真理。解碼濟慈，可能會說："碼即是真，真即是碼，—這就是你在鍵盤上所需要的一切"。

The kind of truth that is embodied in fine pottery is not same as the truth in beautiful poetry, and the truth that is captured in a computer program is far from both of these. But in the end, all three these forms use the expressive power of the medium to get at truth.
體現在精美陶器中的真理與美麗詩歌中的真理不同，電腦程式中捕獲的真理遠非這兩者。但是最後，這三種形式都使用媒介的表達力來實現真理。

Programmers are, in that sense, like artists and poets[19].
從這個意義上來說，程式設計師像藝術家和詩人一樣[20].

Review and summary
回顧與總結

* Logic (in this course) is the study of correct forms of reasoning
  邏輯（在本課程中）是對正確推理形式的研究
* Formal logic is the use of formulae to do logic
  形式邏輯是使用公式來執行邏輯
* Truth tables spell out exactly how logical operators work
  真值表準確地闡明了邏輯運算子的運作方式
* To be useful, a formal logic must be part of a formal language
  為了有用，形式邏輯必須是正式語言的一部分
* A formal language is a set of symbols and a set of rules for combining them
  形式語言是一組符號及一組合併它們的規則
* Julia is itself a formal language
  Julia 本身就是一種形式語言
* A programming language is capable of endless variety of expression
  程式設計語言可以有無窮的各種表示式
* Writing code is a creative as much as writing poetry or music.
  編寫代碼與寫詩歌或音樂一樣是創作。

[19] But also like storytellers, stand-up comedians, composers and basket-weavers. All of these aim at certain effects which in some sense set a standard of truth.
[20] 但也像講故事的人，站立喜劇演員，作曲家和籃子編織師一樣。所有這些目的都是針對某種意義上樹立真理標準的某些效果。