Biến và câu lệnh trong Python

Trong chương trước, chúng ta đã sử dụng các toán tử để viết các biểu thức thực hiện các phép toán số học.

Trong chương này, bạn sẽ học về biến và câu lệnh trong Python, bao gồm câu lệnh `import` và hàm `print`.

Tôi cũng sẽ giới thiệu thêm một số thuật ngữ mà chúng ta sử dụng để nói về các chương trình, bao gồm `tham số` và `mô-đun`.

Biến trong Python

Một biến là một tên tham chiếu đến một giá trị. Để tạo một biến, chúng ta có thể viết **một câu lệnh gán** như sau.

```
n = 17
```

Một câu lệnh gán có ba phần: tên của biến ở bên trái, toán tử gán ở giữa `(=)`, và một biểu thức ở bên phải.

Trong ví dụ này, biểu thức trên là một số nguyên. Trong ví dụ tiếp theo, biểu thức là một số thực.

```
pi = 3.141592653589793
```

Và trong ví dụ tiếp theo, biểu thức là một chuỗi.

```
message = 'And now for something completely different'
```

Khi bạn chạy một câu lệnh gán, sẽ không có đầu ra nào. Python tạo ra biến và gán cho nó một giá trị, nhưng câu lệnh gán không có tác động hiển thị nào. Tuy nhiên, sau khi tạo ra một biến, bạn có thể sử dụng nó như một biểu thức. Vì vậy, chúng ta có thể hiển thị giá trị của biến `message` như sau:

```
message
```

```
'And now for something completely different'
```

Bạn cũng có thể sử dụng một biến như một phần của biểu thức với các toán tử số học.

```
n + 25
```

```
42
```

```
2 * pi
```

```
6.283185307179586
```

Và bạn có thể sử dụng một biến khi gọi một hàm.

```
round(pi)
```

```
3
```

```
len(message)
```

```
42
```

Sơ đồ trạng thái

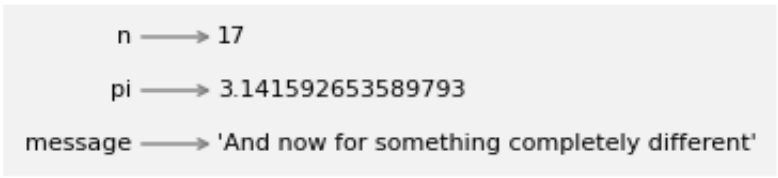
Một cách phổ biến để biểu diễn các biến trên giấy là viết tên biến kèm theo một mũi tên chỉ đến giá trị của nó.

```
# Khối mã bên dưới hỗ trợ tải các nguồn tài liệu cần thiết
# cho các chương trình trong các phần nội dung tiếp theo.

from os.path import basename, exists

def download(url):
    filename = basename(url)
    if not exists(filename):
        from urllib.request import urlretrieve

        local, _ = urlretrieve(url, filename)
        print("Downloaded " + str(local))
    return filename

download('https://github.com/AllenDowney/ThinkPython/raw/v3/thinkpython.py');
download('https://github.com/AllenDowney/ThinkPython/raw/v3/diagram.py');

import thinkpython
```

```
Downloaded thinkpython.py
Downloaded diagram.py
```

```
import math

from diagram import make_binding, Frame

binding = make_binding("message", 'And now for something completely different')
binding2 = make_binding("n", 17)
binding3 = make_binding("pi", 3.141592653589793)

frame = Frame([binding2, binding3, binding])
```

```
from diagram import diagram, adjust


width, height, x, y = [3.62, 1.01, 0.6, 0.76]
ax = diagram(width, height)
bbox = frame.draw(ax, x, y, dy=-0.25)
# adjust(x, y, bbox)
```



Loại hình vẽ này được gọi là sơ đồ trạng thái vì nó cho thấy trạng thái của từng biến (hãy coi nó như trạng thái tâm trí của biến đó). Chúng ta sẽ sử dụng sơ đồ trạng thái trong suốt cuốn sách để đại diện cho mô hình cách Python lưu trữ các biến và giá trị của chúng.

Tên biến

Tên biến có thể ngắn hoặc dài tùy ý của bạn. Chúng có thể chứa cả `chữ cái` và `số`, nhưng không được bắt đầu bằng một số. Việc sử dụng chữ cái hoa là hợp pháp, nhưng cần theo quy ước là chúng ta thường chỉ sử dụng chữ thường để đặt tên biến.

Dấu câu duy nhất có thể xuất hiện trong tên biến là ký tự gạch dưới `(_)`. Nó thường được sử dụng trong các tên có nhiều từ, chẳng hạn như `your_name` hoặc `airspeed_of_unladen_swallow`.

Nếu bạn đặt tên cho một biến bằng một tên không hợp lệ, bạn sẽ gặp lỗi cú pháp. Tên `million!` là không hợp lệ vì nó chứa dấu chấm than.

```
%%expect SyntaxError

million! = 1000000
```

```
  File "<ipython-input-12-0539b52dc75a>", line 1
    million! = 1000000
            ^
SyntaxError: invalid syntax
```

Tên `76trombones` là không hợp lệ vì nó bắt đầu bằng một số.

```
%%expect SyntaxError

76trombones = 'big parade'
```

```
  File "<ipython-input-13-ee59a172c534>", line 1
    76trombones = 'big parade'
     ^
SyntaxError: invalid decimal literal
```

Tên `class` cũng là không hợp lệ, nhưng có thể không rõ lý do tại sao.

```
%%expect SyntaxError

class = 'Self-Defence Against Fresh Fruit'
```

```
  File "<ipython-input-14-d406f8e51b6b>", line 1
    class = 'Self-Defence Against Fresh Fruit'
          ^
SyntaxError: invalid syntax
```

Hóa ra, `class` là một từ khóa, tức là một từ đặc biệt được sử dụng để xác định cấu trúc của một chương trình. Các từ khóa không thể được sử dụng làm tên biến.

Dưới đây là danh sách đầy đủ các từ khóa trong Python:

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

```
from keyword import kwlist

len(kwlist)
```

```
35
```

Bạn không cần phải ghi nhớ danh sách này. Trong hầu hết các môi trường phát triển, các từ khóa được hiển thị bằng màu sắc khác; nếu bạn cố gắng sử dụng một từ khóa làm tên biến, bạn sẽ biết ngay.

Câu lệnh import

Để sử dụng một số tính năng của Python, bạn phải `import` chúng. Ví dụ, câu lệnh sau đây nhập **mô-đun toán học**.

```
import math
```

Một mô-đun là một tập hợp các biến và hàm. Mô-đun toán học cung cấp một biến gọi là `pi`, chứa giá trị của hằng số toán học được ký hiệu là π. Chúng ta có thể hiển thị giá trị của nó như sau.

```
math.pi
```

```
3.141592653589793
```

Để sử dụng một biến trong một mô-đun, bạn phải sử dụng **toán tử chấm** `(.)` giữa tên của mô-đun và tên của biến.

Mô-đun toán học cũng chứa các hàm. Ví dụ, hàm `sqrt` tính căn bậc hai.

```
math.sqrt(25)
```

```
5.0
```

Và hàm `pow` nâng một số lên lũy thừa của một số thứ hai.

```
math.pow(5, 2)
```

```
25.0
```

Tại thời điểm này, chúng ta đã thấy hai cách để nâng một số lên lũy thừa: chúng ta có thể sử dụng hàm `math.pow` hoặc toán tử lũy thừa `**`. Cả hai cách đều hợp lệ, nhưng toán tử thường được sử dụng nhiều hơn hàm.

Biểu thức và câu lệnh trong Python

Cho đến thời điểm hiện tại, chúng ta đã thấy một vài loại biểu thức. Một biểu thức có thể là một giá trị đơn, như một số nguyên, số thực, hoặc chuỗi. Nó cũng có thể là một tập hợp các giá trị và toán tử. Và nó có thể bao gồm tên biến và các lời gọi hàm. Dưới đây là một biểu thức bao gồm nhiều yếu tố này.

```
19 + n + round(math.pi) * 2
```

```
42
```

Chúng ta cũng đã thấy một vài loại câu lệnh. Một câu lệnh là một đơn vị mã có tác dụng, nhưng không có giá trị. Ví dụ, một câu lệnh gán tạo ra một biến và gán cho nó một giá trị, nhưng chính câu lệnh đó không có giá trị.

```
n = 17
```

Tương tự, một câu lệnh `import` cũng có tác dụng — nó nhập một mô-đun để chúng ta có thể sử dụng các biến và hàm mà nó chứa — nhưng không có tác động nào có thể nhìn thấy được.

```
import math
```

Việc tính toán giá trị của một biểu thức được gọi là **đánh giá**. Việc thực thi một câu lệnh được gọi là **thực thi**.

Hàm print trong Python

Khi bạn đánh giá một biểu thức, kết quả sẽ được hiển thị.

```
n + 1
```

```
18
```

Nhưng nếu bạn đánh giá nhiều hơn một biểu thức, chỉ có giá trị của biểu thức cuối cùng được hiển thị.

```
n + 2
n + 3
```

```
20
```

Để hiển thị nhiều giá trị hơn, bạn có thể sử dụng hàm `print`.

```
print(n+2)
print(n+3)
```

```
19
20
```

Hàm này cũng hoạt động với các số thực và chuỗi.

```
print('The value of pi is approximately')
print(math.pi)
```

```
The value of pi is approximately
3.141592653589793
```

Bạn cũng có thể sử dụng một chuỗi các biểu thức được ngăn cách bằng dấu phẩy.

```
print('The value of pi is approximately', math.pi)
```

```
The value of pi is approximately 3.141592653589793
```

Lưu ý rằng hàm `print` sẽ chèn một khoảng cách giữa các giá trị.

Đối số trong Python

Khi bạn gọi một hàm, biểu thức trong dấu ngoặc đơn được gọi là **đối số.** Thông thường, tôi sẽ giải thích lý do, nhưng trong trường hợp này, ý nghĩa kỹ thuật của thuật ngữ gần như không liên quan gì đến ý nghĩa thông thường của từ, vì vậy tôi sẽ không cố gắng làm điều đó.

Một số hàm mà chúng ta đã thấy cho đến nay chỉ nhận một đối số, như hàm `int`.

```
int('101')
```

```
101
```

Một số hàm nhận hai đối số, như hàm `math.pow`.

```
math.pow(5, 2)
```

```
25.0
```

Một số hàm có thể nhận thêm các đối số tùy chọn. Ví dụ, hàm `int` có thể nhận một đối số thứ hai để xác định cơ số của số đó.

```
int('101', 2)
```

```
5
```

Chuỗi số `101` trong cơ số 2 đại diện cho số 5 trong cơ số 10.

Hàm `round` cũng nhận một đối số thứ hai tùy chọn, đó là số chữ số thập phân để làm tròn.

```
round(math.pi, 3)
```

```
3.142
```

Một số hàm có thể nhận bất kỳ số lượng đối số nào, như hàm `print`.

```
print('Any', 'number', 'of', 'arguments')
```

```
Any number of arguments
```

Nếu bạn gọi một hàm và cung cấp quá nhiều đối số, đó sẽ là một lỗi `TypeError`.

```
%%expect TypeError

float('123.0', 2)
```

```
TypeError: float expected at most 1 argument, got 2
```

Nếu bạn cung cấp quá ít đối số, đó cũng sẽ là một lỗi `TypeError`.

```
%%expect TypeError

math.pow(2)
```

```
TypeError: pow expected 2 arguments, got 1
```

Và nếu bạn cung cấp một đối số với kiểu dữ liệu mà hàm không thể xử lý, đó cũng sẽ là một lỗi `TypeError`.

```
%%expect TypeError

math.sqrt('123')
```

```
TypeError: must be real number, not str
```

Loại kiểm tra này có thể gây khó chịu khi bạn mới bắt đầu, nhưng nó giúp bạn phát hiện và sửa lỗi.

Ghi chú trong Python

Khi các chương trình trở nên lớn hơn và phức tạp hơn, chúng cũng trở nên khó đọc hơn. Ngôn ngữ lập trình thường rất cô đọng, và đôi khi rất khó để nhìn vào một đoạn mã và hiểu nó đang làm gì và tại sao.

Vì lý do này, việc thêm các ghi chú vào chương trình của bạn để giải thích bằng ngôn ngữ tự nhiên về những gì chương trình đang làm là một ý tưởng tốt. Những ghi chú này được gọi là **ghi chú**, và chúng bắt đầu bằng ký hiệu `#`.

```
# số giây của 42:42
seconds = 42 * 60 + 42
```

Trong trường hợp này, ghi chú xuất hiện trên một dòng riêng biệt. Bạn cũng có thể đặt ghi chú ở cuối một dòng:

```
miles = 10 / 1.61     # 10 km bằng bao nhiêu dặm.
```

Mọi thứ từ dấu `#` đến cuối dòng đều bị bỏ qua — nó không có ảnh hưởng gì đến việc thực thi chương trình.

Ghi chú hữu ích nhất khi chúng ghi chú những đặc điểm không hiển nhiên của mã. Có thể giả định rằng người đọc có thể tự hiểu được mã đang làm gì; điều hữu ích hơn là giải thích lý do tại sao.

Ghi chú này thừa với mã và vô ích:

```
v = 8     # gán v bằng 8
```

Ghi chú này chứa thông tin hữu ích không có trong mã:

```
v = 8     # Vận tốc tính bằng dặm mỗi giờ.
```

Tên biến tốt có thể giảm nhu cầu sử dụng ghi chú, nhưng tên quá dài có thể làm cho các biểu thức phức tạp khó đọc, vì vậy có một sự đánh đổi.

Gỡ lỗi

Ba loại lỗi có thể xảy ra trong một chương trình: *lỗi cú pháp, lỗi thời gian chạy, và lỗi ngữ nghĩa*. Việc phân biệt giữa chúng sẽ giúp bạn tìm ra lỗi nhanh hơn.

- **Lỗi cú pháp:** liên quan đến cấu trúc của một chương trình và các quy tắc về cấu trúc đó. Nếu có lỗi cú pháp ở bất kỳ đâu trong chương trình, Python sẽ không chạy chương trình. Nó sẽ hiển thị thông báo lỗi ngay lập tức.
- **Lỗi thời gian chạy:** Nếu không có lỗi cú pháp trong chương trình của bạn, chương trình có thể bắt đầu chạy. Nhưng nếu có sự cố xảy ra, Python sẽ hiển thị thông báo lỗi và dừng lại. Loại lỗi này được gọi là lỗi thời gian chạy. Nó cũng được gọi là **ngoại lệ** vì nó cho thấy có điều gì đó bất thường đã xảy ra.
- **Lỗi ngữ nghĩa:** Loại lỗi thứ ba là lỗi `ngữ nghĩa`, có nghĩa là liên quan đến ý nghĩa. Nếu có lỗi ngữ nghĩa trong chương trình của bạn, chương trình sẽ chạy mà không tạo ra thông báo lỗi, nhưng nó không thực hiện đúng điều bạn mong muốn. Việc xác định lỗi ngữ nghĩa có thể khó khăn vì bạn phải làm việc ngược lại bằng cách nhìn vào kết quả đầu ra của chương trình và cố gắng tìm ra nó đang làm gì.

Như chúng ta đã thấy, một tên biến không hợp lệ sẽ tạo ra một lỗi cú pháp.

```
%%expect SyntaxError

million! = 1000000
```

```
  File "<ipython-input-39-0539b52dc75a>", line 1
    million! = 1000000
            ^
SyntaxError: invalid syntax
```

Nếu bạn sử dụng một toán tử với một kiểu mà nó không hỗ trợ, đó sẽ là một lỗi thời gian chạy.

```
%%expect TypeError

'126' / 3
```

```
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

Cuối cùng, đây là một ví dụ về lỗi ngữ nghĩa. Giả sử chúng ta muốn tính giá trị trung bình của 1 và 3, nhưng chúng ta quên quy tắc thứ tự thực hiện và viết như sau:

```
1 + 3 / 2
```

```
2.5
```

Khi biểu thức này được đánh giá, nó không tạo ra thông báo lỗi, vì vậy không có lỗi cú pháp hay lỗi thời gian chạy. Nhưng kết quả không phải là giá trị trung bình của `1` và `3`, nên chương trình không chính xác. Đây là một lỗi ngữ nghĩa vì chương trình vẫn chạy nhưng không thực hiện đúng những gì dự định.

Thuật ngữ

- **variable - biến:** Tên đại diện cho một giá trị.
- **assignment statement - câu lệnh gán:** Câu lệnh gán một giá trị cho một biến.
- **state diagram - sơ đồ trạng thái:** Biểu diễn đồ họa của một tập hợp các biến và các giá trị mà chúng đại diện.
- **keyword - từ khóa:** Một từ đặc biệt được sử dụng để xác định cấu trúc của một chương trình.
- **import statement - câu lệnh nhập:** Câu lệnh đọc một tệp module để chúng ta có thể sử dụng các biến và hàm có trong đó.
- **module - mô-đun:** Một tệp chứa mã Python, bao gồm các định nghĩa hàm và đôi khi các câu lệnh khác.
- **dot operator - toán tử chấm:** Toán tử `.` được sử dụng để truy cập một hàm trong một mô-đun khác bằng cách chỉ định tên mô-đun theo sau bởi một dấu chấm và tên hàm.
- **evaluate - đánh giá:** Thực hiện các phép toán trong một biểu thức để tính toán một giá trị.
- **statement - câu lệnh:** Một hoặc nhiều dòng mã đại diện cho một lệnh hoặc hành động.
- **execute - thực thi:** Chạy một câu lệnh và thực hiện những gì nó chỉ định.
- **argument - đối số:** Một giá trị được cung cấp cho một hàm khi hàm được gọi.
- **comment - chú thích:** Văn bản được bao gồm trong một chương trình cung cấp thông tin về chương trình nhưng không ảnh hưởng đến việc thực thi.
- **runtime error - lỗi thời gian chạy:** Một lỗi gây ra chương trình hiển thị thông báo lỗi và thoát.
- **exception - ngoại lệ:** Một lỗi được phát hiện khi chương trình đang chạy.
- **semantic error - lỗi ngữ nghĩa:** Một lỗi khiến chương trình thực hiện sai hành động, nhưng không hiển thị thông báo lỗi.

Bài tập

```
# Ô này yêu cầu Jupyter cung cấp thông tin gỡ lỗi chi tiết
# Khi xảy ra lỗi thời gian chạy. Chạy nó trước khi làm các bài tập.

%xmode Verbose
```

Hãy hỏi trợ lý ảo

Một lần nữa, tôi khuyến khích bạn sử dụng một trợ lý ảo để tìm hiểu thêm về bất kỳ chủ đề nào trong chương này.

Nếu bạn tò mò về bất kỳ từ khóa nào mà tôi đã liệt kê, bạn có thể hỏi: *Tại sao `class` lại là một từ khóa?* hoặc *Tại sao tên biến không thể là từ khóa?*

Bạn có thể nhận thấy rằng `int`, `float` và `str` không phải là từ khóa trong Python. Chúng là các biến đại diện cho kiểu dữ liệu và có thể được sử dụng như các hàm. Vì vậy, việc có một biến hoặc hàm với một trong những tên đó là hợp pháp, nhưng được khuyến cáo là không nên. Hãy hỏi một trợ lý: *Tại sao lại xấu khi sử dụng `int`, `float` và `str` làm tên biến?*

Cũng hãy hỏi: *Các hàm tích hợp sẵn trong Python là gì?* Nếu bạn tò mò về bất kỳ hàm nào trong số đó, hãy yêu cầu thêm thông tin.

Trong chương này, chúng ta đã nhập mô-đun toán học và sử dụng một số biến và hàm mà nó cung cấp. Hãy hỏi một trợ lý: *Có những biến và hàm nào trong mô-đun toán học?* và *ngoài hàm `math`, những mô-đun nào được coi là quan trọng trong Python?*

Bài tập 1

Lặp lại lời khuyên từ chương trước, bất cứ khi nào bạn học một tính năng mới, bạn nên cố tình mắc lỗi để xem điều gì sai.

- Chúng ta đã thấy rằng `n = 17` là hợp lệ. Còn `17 = n` thì sao?
- Thế còn `x = y = 1` thì như thế nào?
- Trong một số ngôn ngữ, mọi câu lệnh đều kết thúc bằng dấu chấm phẩy (;). Điều gì xảy ra nếu bạn đặt một dấu chấm phẩy vào cuối một câu lệnh Python?
- Điều gì xảy ra nếu bạn đặt một dấu chấm vào cuối một câu lệnh?
- Điều gì xảy ra nếu bạn đánh vần sai tên của một mô-đun và cố gắng nhập “maath”?

```
import maath
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-13-c40f797fdd49> in <cell line: 1>()
----> 1 import maath

ModuleNotFoundError: No module named 'maath'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

Bài tập 2

Luyện tập sử dụng trình thông dịch Python như một máy tính:

Phần 1: Thể tích của một hình cầu với bán kính r được tính theo công thức là $\frac{4}{3}\pi r^3$. Vậy thể tích của một hình cầu có bán kính 5 là bao nhiêu? Bắt đầu với một biến có tên là `radius`, sau đó gán kết quả cho một biến có tên là `volume`. Hiển thị kết quả. Thêm các ghi chú để chỉ rõ rằng `radius` là đơn vị cm và `volume` là đơn vị cm^3.

Phần 2: Một quy tắc trong lượng giác nói rằng với bất kỳ giá trị nào của x, thì $(\cos x)^2 + (\sin x)^2 = 1$. Hãy kiểm tra xem điều này có đúng với một giá trị cụ thể của x là 42.

Tạo một biến có tên là x với giá trị này. Sau đó, sử dụng `math.cos` và `math.sin` để tính sin và cos của x, rồi tính tổng bình phương của chúng.

Kết quả nên gần bằng 1. Nó có thể không chính xác là 1 vì số học thập phân không hoàn toàn chính xác — nó chỉ xấp xỉ đúng.

Phần 3: Ngoài `pi`, biến khác được định nghĩa trong mô-đun toán học là `e`, đại diện cho cơ số của logarit tự nhiên, được viết trong ký hiệu toán học là e. Nếu bạn chưa quen thuộc với giá trị này, hãy hỏi trợ lý ảo "Giá trị của `math.e` là bao nhiêu?"

Bây giờ, hãy tính e^2 theo ba cách:

1. Sử dụng `math.e` và toán tử lũy thừa `(**)`.
2. Sử dụng `math.pow` để nâng `math.e` lên lũy thừa 2.
3. Sử dụng `math.exp`, nhận đối số là một giá trị x, và tính e^x.

Bạn có thể nhận thấy rằng kết quả cuối cùng hơi khác so với hai kết quả kia. Hãy thử xem kết quả nào là chính xác.