

✓ Lập trình như một cách tư duy

Mục tiêu đầu tiên của cuốn sách này là hướng dẫn bạn cách lập trình bằng Python. Nhưng việc học lập trình có nghĩa là học một cách tư duy mới, vì vậy mục tiêu thứ hai của cuốn sách này là giúp bạn tư duy như một nhà Khoa học máy tính. Cách tư duy này kết hợp một số đặc điểm tốt nhất của *toán học*, *kỹ thuật* và *khoa học tự nhiên*. Giống như các nhà toán học, các nhà khoa học máy tính sử dụng ngôn ngữ chính thức để biểu thị ý tưởng của họ — cụ thể là các phép toán. Giống như các kỹ sư, họ thiết kế các thứ, lắp ráp các thành phần thành hệ thống, và đánh giá các sự đánh đổi giữa các lựa chọn. Giống như các nhà khoa học, họ quan sát hành vi của các hệ thống phức tạp, hình thành giả thuyết, và kiểm tra các dự đoán.

Chúng ta sẽ bắt đầu với các yếu tố cơ bản nhất của lập trình và tiến dần lên. Trong chương này, chúng ta sẽ xem cách Python làm việc với các số, chữ cái, và từ ngữ. Và bạn sẽ học cách thực hiện các phép toán số học bằng Python.

Bạn cũng sẽ bắt đầu làm quen với các thuật ngữ của lập trình, bao gồm các thuật ngữ như toán tử, biểu thức, giá trị, và kiểu dữ liệu. Những thuật ngữ này rất quan trọng, bạn sẽ cần chúng để hiểu phần còn lại của cuốn sách, để giao tiếp với các lập trình viên khác, và để sử dụng cũng như hiểu các trợ lý ảo.

✓ Toán tử số học

Một **toán tử số học** là một ký hiệu đại diện cho một phép toán số học. Ví dụ, dấu cộng (+) thực hiện phép cộng.

$30 + 12$

⇒ 42

Dấu trừ (−) là toán tử thực hiện phép trừ.

$43 - 1$

⇒ 42

Dấu hoa thị (*) thực hiện phép nhân.

6 * 7

 42

Và dấu gạch chéo (/), thực hiện phép chia.

84 / 2

 42.0

Lưu ý rằng kết quả của phép chia là 42.0 chứ không phải 42. Đó là vì có hai loại phép chia trong Python:

- số nguyên, đại diện cho các số nguyên,
- số thực, đại diện cho các số có dấu thập phân.

Nếu bạn cộng, trừ, hoặc nhân hai số nguyên, kết quả sẽ là một số nguyên. Nhưng nếu bạn chia hai số nguyên, kết quả sẽ là một số thực. Python cung cấp một toán tử khác, // để thực hiện phép chia nguyên. Kết quả của phép chia nguyên luôn là một số nguyên.

84 // 2

 42

Phép chia nguyên cũng được gọi là phép chia làm tròn xuống vì nó luôn làm tròn xuống.

85 // 2

 42

Cuối cùng, toán tử ** thực hiện phép lũy thừa; nghĩa là, nó nâng một số lên một lũy thừa:

7 ** 2

 49

Trong một số ngôn ngữ khác, dấu mũ (^) , được sử dụng cho phép lũy thừa, nhưng trong Python, nó là một toán tử *bitwise* được gọi là XOR . Nếu bạn không quen thuộc với các toán tử *bitwise*, kết quả có thể sẽ bất ngờ:

$7 \wedge 2$

 5

Tôi sẽ không đề cập đến các toán tử *bitwise* trong cuốn sách này, nhưng bạn có thể đọc thêm về chúng tại <http://wiki.python.org/moin/BitwiseOperators>.

✓ Biểu thức trong Python

Một tập hợp các toán tử và số được gọi là **biểu thức**. Một biểu thức có thể chứa bất kỳ số lượng toán tử và số nào. Ví dụ, đây là một biểu thức chứa hai toán tử.

$6 + 6 ** 2$

 42

Lưu ý rằng phép lũy thừa xảy ra trước phép cộng. Python tuân theo thứ tự ưu tiên mà bạn có thể đã học trong lớp toán: phép lũy thừa xảy ra trước phép nhân và phép chia, và phép nhân và phép chia xảy ra trước phép cộng và phép trừ.


Trong ví dụ sau, phép nhân xảy ra trước phép cộng.

$12 + 5 * 6$

 42

Nếu bạn muốn phép cộng xảy ra trước, bạn có thể sử dụng dấu ngoặc đơn.

$(12 + 5) * 6$

 102

Mỗi biểu thức đều có **một giá trị**. Ví dụ, biểu thức $6 * 7$ có giá trị là 42 .

Lưu ý rằng khi diễn đạt một biểu thức trong Python, bạn cần tuân thủ một quy tắc được gọi là **phong cách viết mã**. Tức bạn cần thể hiện mã của bạn một cách rõ ràng, tránh việc viết dính liền điều này sẽ gây rất nhiều khó khăn trong quá trình tìm lỗi và sửa lỗi sau này. Một lý do khác là mã Python của bạn sẽ rất dễ quan sát, theo dõi, hoặc đọc trong tương lai nếu bạn áp dụng **phong cách viết mã** ngay từ bây giờ.

Xem thêm ví dụ về phong cách viết mã sau đây:

```
((13 ^ 2) * 4) + 5
```

 65

Rõ ràng, mã bên trên sẽ rất dễ đọc và cho chúng ta một kết quả chính xác hơn so với ô mã bên dưới phải không nào. Vậy thì tại sao chúng ta không tập làm quen với phong cách viết mã ngay bây giờ để thể hiện chúng ta là một người cẩn thận và là một lập trình viên chuyên nghiệp.

```
13^2*4+5
```

 0

✓ Các hàm số học

Ngoài các toán tử số học, Python cung cấp **một số hàm** làm việc với các số. Ví dụ, hàm `round()` nhận một số thực và làm tròn nó về số nguyên gần nhất.

```
round(42.4)
```

 42

```
round(42.6)
```

 43

Hàm `abs()` tính toán giá trị tuyệt đối của một số. Đối với một số dương, giá trị tuyệt đối chính là số đó.

```
abs(42)
```

```
↔ 42
```

Đối với một số âm, giá trị tuyệt đối là số dương.

```
abs(-42)
```

```
↔ 42
```

Khi chúng ta sử dụng một hàm như thế này, chúng ta nói rằng chúng ta đang **gọi hàm**. Một biểu thức gọi một hàm được gọi là một lời **gọi hàm**.

Khi bạn gọi một hàm, dấu ngoặc đơn là bắt buộc. Nếu bạn bỏ qua chúng, bạn sẽ nhận được một thông báo lỗi.

Lưu ý: Ô dưới đây sử dụng `%%expect`, đây là một lệnh ma thuật trong Jupyter có nghĩa là chúng ta mong đợi mã trong ô này sẽ tạo ra một lỗi. Để biết thêm về chủ đề này, hãy xem phần giới thiệu về [sổ ghi chép](#).

```
%%expect SyntaxError
```

```
abs 42
```

```
↔ UsageError: Cell magic `%%expect` not found.
```

Bạn có thể bỏ qua dòng đầu tiên của thông báo này; nó không chứa bất kỳ thông tin nào mà chúng ta cần hiểu ngay bây giờ. Dòng thứ hai là mã chứa lỗi, với một dấu mũ (^) bên dưới để chỉ ra vị trí mà lỗi được phát hiện.

Dòng cuối cùng cho biết đây là **một lỗi cú pháp**, có nghĩa là có điều gì đó sai với cấu trúc của biểu thức. Trong ví dụ này, vấn đề là một lời gọi hàm cần có dấu ngoặc đơn.

Hãy xem điều gì xảy ra nếu bạn bỏ qua dấu ngoặc đơn và giá trị.

```
abs
```

```
↔ <function abs(x, /)>
```


Tên hàm tự nó là một biểu thức hợp lệ có giá trị. Khi nó được hiển thị, giá trị cho biết rằng `abs` là một hàm, và nó bao gồm một số thông tin bổ sung mà tôi sẽ giải thích sau.

✓ Chuỗi trong Python

Ngoài các số, Python cũng có thể đại diện cho các chuỗi ký tự, được gọi là **chuỗi** vì các ký tự được xâu lại với nhau như những hạt trên một chiếc dây chuyền. Chính vì thế, một số sách được xuất bản tại Việt Nam thường hay gọi **chuỗi** là xâu, và các phép toán tử được áp dụng trên chuỗi thường được gọi là *các phép toán tử xâu*.

Để viết một chuỗi, chúng ta có thể đặt một chuỗi các ký tự bên trong dấu nháy đơn thẳng. Để hiển thị chuỗi `'Hello'` trên sổ ghi chép, chúng ta thường sử dụng chúng với hàm `print()` như bên dưới:

```
print('Hello')
```

 Hello

Nó cũng hợp lệ khi sử dụng dấu nháy kép.

```
print("world")
```

 world

Dấu nháy kép giúp dễ dàng viết một chuỗi có chứa dấu nháy đơn, vì đây là ký hiệu giống như dấu nháy đơn thẳng.

```
print("it's a small ")
```

 it's a small

Một chuỗi cũng có thể chứa khoảng trắng, dấu câu, và chữ số.

```
print('Well, ')
```

 Well,

Toán tử + hoạt động với các chuỗi; nó nối hai chuỗi lại thành một chuỗi duy nhất, được gọi là **phép nối chuỗi**.

```
print('Well, ' + "it's a small " + 'world.')
```

```
⇒ Well, it's a small world.
```

Toán tử * cũng hoạt động với các chuỗi; nó tạo ra nhiều bản sao của một chuỗi và nối chúng lại với nhau.

```
print('Spam, ' * 4)
```

```
⇒ Spam, Spam, Spam, Spam,
```

Các toán tử số học khác không hoạt động với các chuỗi.

Python cung cấp một hàm tích hợp sẵn gọi là len tính toán độ dài của một chuỗi.

```
len('Spam')
```

```
⇒ 4
```

Lưu ý rằng hàm len() đếm các ký tự giữa các dấu nháy, nhưng không bao gồm dấu nháy.

Khi bạn tạo một chuỗi, hãy chắc chắn sử dụng dấu nháy thẳng. Dấu nháy nghiêng (') sẽ gây ra lỗi cú pháp.

```
%%expect SyntaxError
```

```
`Hello`
```

```
⇒ UsageError: Cell magic `%%expect` not found.
```

Dấu nháy thông minh, còn được gọi là dấu nháy cong là không hợp lệ.

```
%%expect SyntaxError
```

```
'Hello'
```

```
➞ UsageError: Cell magic `%%expect` not found.
```

✓ Giá trị và kiểu dữ liệu

Cho đến thời điểm hiện tại, chúng ta đã thấy ba loại giá trị:

- 2 là một số nguyên,
- 42.0 là một số thực,
- 'Hello' là một chuỗi.

Một loại giá trị được gọi là **kiểu dữ liệu**. Mỗi giá trị đều có một kiểu dữ liệu - hoặc đôi khi chúng ta nói rằng nó **thuộc về** một kiểu nào đó.

Python cung cấp một hàm gọi là `type` để cho bạn biết kiểu dữ liệu của bất kỳ giá trị nào. Kiểu dữ liệu của một số nguyên là `int`.

```
type(2)
```

```
➞ int
```

Kiểu của một số thực là `float`.

```
type(42.0)
```

```
➞ float
```

Và kiểu dữ liệu của một chuỗi là `str`.

```
type('Hello, World!')
```

```
➞ str
```

Các kiểu `int`, `float`, và `str` có thể được sử dụng như các hàm. Ví dụ, `int` có thể nhận một số thực và chuyển đổi nó thành một số nguyên (luôn làm tròn xuống).


```
int(42.9)
```

```
↔ 42
```

Và float có thể chuyển đổi một số nguyên thành giá trị số thực.

```
float(42)
```

```
↔ 42.0
```

Bây giờ, đây là điều có thể gây nhầm lẫn. Nếu bạn đặt một chuỗi các chữ số trong dấu nháy, bạn sẽ nhận được một chuỗi chứ không phải là một số.

```
'126'
```

```
↔ '126'
```

Nó trông giống như một con số, nhưng thực ra đó là một chuỗi.

```
type('126')
```

```
↔ str
```

Nếu bạn cố gắng sử dụng nó như một con số, bạn có thể gặp lỗi.

```
%%expect TypeError
```

```
'126' / 3
```

```
↔ UsageError: Cell magic `%%expect` not found.
```

Ví dụ này tạo ra lỗi `TypeError` (lỗi kiểu dữ liệu), có nghĩa là các giá trị trong biểu thức, được gọi là **toán hạng**, có kiểu không phù hợp. Thông báo lỗi cho biết toán tử `/` không hỗ trợ các kiểu giá trị này, cụ thể là `str` và `int`.

Nếu bạn có một chuỗi chứa các chữ số, bạn có thể sử dụng `int` để chuyển đổi nó thành số nguyên.

```
int('126') / 3
```

```
⇒ 42.0
```

Nếu bạn có một chuỗi chứa các chữ số và dấu thập phân, bạn có thể sử dụng `float` để chuyển đổi nó thành số thực.

```
float('12.6')
```

```
⇒ 12.6
```

Khi bạn viết một số nguyên lớn, bạn có thể muốn sử dụng dấu phẩy giữa các nhóm chữ số, như trong `1,000,000`. Đây là một biểu thức hợp lệ trong Python, nhưng kết quả không phải là một số nguyên.

```
1,000,000
```

```
⇒ (1, 0, 0)
```

Python hiểu `1,000,000` như một dãy các số nguyên được ngăn cách bởi dấu phẩy. Chúng ta sẽ tìm hiểu thêm về loại chuỗi này sau.

Bạn có thể sử dụng dấu gạch dưới (`_`) để làm cho các số lớn dễ đọc hơn.

```
1_000_000
```

```
⇒ 1000000
```

✓ Ngôn ngữ tự nhiên và ngôn ngữ chính thức

Ngôn ngữ tự nhiên là các ngôn ngữ mà con người sử dụng để giao tiếp, như tiếng Anh, tiếng Tây Ban Nha, và tiếng Pháp. Chúng không được con người thiết kế mà phát triển một cách tự nhiên.

Ngôn ngữ chính thức là các ngôn ngữ được con người thiết kế cho các ứng dụng cụ thể. Ví dụ, ký hiệu mà các nhà toán học sử dụng là một ngôn ngữ chính thức, rất hiệu quả trong việc biểu thị các mối quan hệ giữa các con số và ký hiệu. Tương tự, ngôn ngữ lập trình là các ngôn ngữ chính thức được thiết kế để diễn đạt các phép tính toán.

Mặc dù ngôn ngữ chính thức và ngôn ngữ tự nhiên có một số đặc điểm chung, nhưng có những khác biệt quan trọng:

- **Sự mơ hồ:** Ngôn ngữ tự nhiên đầy rẫy sự mơ hồ, mà con người xử lý bằng cách sử dụng các gợi ý ngữ cảnh và thông tin khác. Ngôn ngữ chính thức được thiết kế để gần như hoặc hoàn toàn không mơ hồ, có nghĩa là bất kỳ chương trình nào cũng chỉ có một ý nghĩa duy nhất, bất kể ngữ cảnh.
- **Tính dư thừa:** Để bù đắp cho sự mơ hồ và giảm thiểu hiểu lầm, ngôn ngữ tự nhiên sử dụng tính dư thừa. Kết quả là chúng thường dài dòng. Ngôn ngữ chính thức ít dư thừa hơn và súc tích hơn.
- **Tính chính xác:** Ngôn ngữ tự nhiên đầy những thành ngữ và ẩn dụ. Ngôn ngữ chính thức có nghĩa chính xác theo những gì nó biểu đạt.

Vì chúng ta đều lớn lên với ngôn ngữ tự nhiên, nên đôi khi việc thích nghi với ngôn ngữ chính thức trở nên khó khăn. Ngôn ngữ chính thức có tính cô đọng hơn ngôn ngữ tự nhiên, do đó việc đọc chúng mất nhiều thời gian hơn. Ngoài ra, cấu trúc rất quan trọng, vì vậy không phải lúc nào cũng tốt nhất khi đọc từ trên xuống dưới, từ trái sang phải. Cuối cùng, các chi tiết rất quan trọng. Những lỗi nhỏ về chính tả và dấu câu, điều mà bạn có thể bỏ qua trong ngôn ngữ tự nhiên, có thể tạo ra sự khác biệt lớn trong ngôn ngữ chính thức.

Gỡ lỗi

Lập trình viên thường mắc lỗi. Vì những lý do thú vị, lỗi lập trình được gọi là **lỗi** và quá trình tìm kiếm, sửa chữa chúng được gọi là **gỡ lỗi**.

Lập trình, đặc biệt là gỡ lỗi, đôi khi có thể khiến bạn trải qua những cảm xúc mạnh. Nếu bạn đang vật lộn với một lỗi khó, bạn có thể cảm thấy tức giận, buồn bã, hoặc xấu hổ.

Chuẩn bị tinh thần cho những phản ứng này có thể giúp bạn đối phó tốt hơn. Một cách tiếp cận là nghĩ về máy tính như một nhân viên với những điểm mạnh như tốc độ và độ chính xác, nhưng cũng có những điểm yếu như thiếu sự thấu hiểu và không có khả năng nắm bắt bức tranh toàn cảnh.

Nhiệm vụ của bạn là trở thành một nhà quản lý giỏi, tìm cách tận dụng điểm mạnh và giảm thiểu điểm yếu. Hãy sử dụng cảm xúc của mình để tập trung vào vấn đề mà không để chúng ảnh hưởng đến khả năng làm việc hiệu quả của bạn.

Học cách gỡ lỗi có thể gây bực bội, nhưng đó là một kỹ năng quý giá và hữu ích cho nhiều hoạt động khác ngoài lập trình. Ở cuối mỗi chương, sẽ có một phần như phần này với các gợi ý của tôi về việc gỡ lỗi. Hy vọng chúng sẽ giúp ích cho bạn!

Thuật ngữ

- **arithmetic operator - toán tử số học:** Một ký hiệu, như $+$ và $*$, biểu thị một phép toán số học như phép cộng hoặc phép nhân.
- **integer - số nguyên:** Một kiểu đại diện cho các số nguyên.
- **floating-point - số thực:** Một kiểu đại diện cho các số có phần thập phân.
- **integer division - phép chia nguyên:** Một toán tử $//$, chia hai số và làm tròn xuống thành số nguyên.
- **expression - biểu thức:** Một sự kết hợp của các biến, giá trị và toán tử.
- **value - giá trị:** Một số nguyên, số thực, hoặc chuỗi — hoặc một trong những loại giá trị khác mà chúng ta sẽ thấy sau.
- **function - hàm:** Một chuỗi các câu lệnh có tên thực hiện một phép toán hữu ích nào đó. Các hàm có thể nhận tham số và có thể hoặc không có kết quả.
- **function call - gọi hàm:** Một biểu thức — hoặc một phần của biểu thức — chạy một hàm. Nó

bao gồm tên hàm theo sau là danh sách tham số trong dấu ngoặc đơn.

- **syntax error - lỗi cú pháp:** Một lỗi trong chương trình khiến nó không thể phân tích — và do đó không thể chạy.
- **string - chuỗi:** Một kiểu đại diện cho các chuỗi ký tự.
- **concatenation - phép nối chuỗi:** Nối hai chuỗi lại với nhau.
- **type - kiểu:** Một danh mục giá trị. Các kiểu dữ liệu mà chúng ta đã thấy cho đến nay là số nguyên (kiểu int), số thực (kiểu float), và chuỗi (kiểu str).
- **operand - toán hạng:** Một trong các giá trị mà một toán tử hoạt động trên đó.
- **natural language - ngôn ngữ tự nhiên:** Bất kỳ ngôn ngữ nào mà con người nói và phát triển một cách tự nhiên.
- **formal language - ngôn ngữ chính thức:** Bất kỳ ngôn ngữ nào mà con người thiết kế cho các mục đích cụ thể, như đại diện cho các ý tưởng toán học hoặc chương trình máy tính. Tất cả các ngôn ngữ lập trình đều là ngôn ngữ chính thức.
- **bug - lỗi:** Một lỗi trong chương trình.
- **debugging - gỡ lỗi:** Quá trình tìm kiếm và sửa chữa lỗi.
- **code - mã:** Các lệnh hoặc đoạn văn bản được viết bằng một ngôn ngữ lập trình để chỉ thị cho máy tính thực hiện một công việc cụ thể. Trong nhiều trường hợp, chúng ta vẫn có thể gọi nó là mã lệnh hoặc mã lập trình.

✓ Bài tập

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

```
%xmode Verbose
```

✓ Hỏi trợ lý ảo

Khi bạn làm việc qua cuốn sách này, có nhiều cách để bạn sử dụng trợ lý ảo hoặc chatbot để giúp bạn học tốt hơn.

- Nếu bạn muốn tìm hiểu thêm về một chủ đề trong chương, hoặc có điều gì đó không rõ ràng, bạn có thể yêu cầu nó giải thích.
- Nếu bạn gặp khó khăn với bất kỳ bài tập nào, bạn có thể hỏi nó để được giúp đỡ.

Trong mỗi chương, tôi sẽ gợi ý những bài tập bạn có thể thực hiện với trợ lý ảo, nhưng tôi khuyến khích bạn hoàn thành các bài tập trước khi nhờ sự trợ giúp của một trợ lý ảo và xem điều gì hiệu quả với bạn.

Dưới đây là một số chủ đề bạn có thể hỏi một trợ lý ảo:

- Trước đó, tôi đã đề cập đến các toán tử bitwise nhưng không giải thích tại sao giá trị của $7 \wedge 2$ là 5. Hãy thử hỏi *Các toán tử bitwise trong Python là gì?* hoặc *Giá trị của $7 \text{ XOR } 2$ là gì?*
- Tôi cũng đã đề cập đến thứ tự thực hiện phép toán. Để biết thêm chi tiết, hãy hỏi *Thứ tự thực hiện phép toán trong Python là gì?*
- Hàm `round`, mà chúng ta đã sử dụng để làm tròn một số thực đến số nguyên gần nhất, có thể nhận một tham số thứ hai. Hãy thử hỏi *Các tham số của hàm `round` là gì?* hoặc *Làm thế nào để làm tròn π đến ba chữ số thập phân?*
- Có một toán tử số học khác mà tôi chưa đề cập; hãy thử hỏi *Toán tử modulo trong Python là gì?*

Hầu hết các trợ lý ảo đều biết về Python, vì vậy chúng thường trả lời các câu hỏi như thế này một cách đáng tin cậy. Nhưng hãy nhớ rằng những công cụ này cũng có thể mắc lỗi. Nếu bạn nhận được mã từ một chatbot, hãy kiểm tra lại nó!

✓ Bài tập 1

Bạn có thể thắc mắc hàm `round` hoạt động như thế nào nếu một số kết thúc bằng 0.5. Câu trả lời là nó đôi khi làm tròn lên và đôi khi làm tròn xuống. Hãy thử các ví dụ sau và xem liệu bạn có thể tìm ra quy tắc mà nó tuân theo không:

`round(42.5)`

Quy tắc làm tròn của python khi kết thúc bằng 0.5 thì sẽ làm tròn đến số nguyên gần nhất

`round(43.5)`

Nếu bạn cảm thấy tò mò, hãy hỏi một trợ lý ảo: *Nếu một số kết thúc bằng 0.5, Python làm tròn lên hay xuống?*

Bài tập 2

Khi bạn tìm hiểu về một tính năng mới, hãy thử nghiệm và cố ý mắc lỗi. Bằng cách đó, bạn sẽ học được các thông báo lỗi, và khi thấy chúng lần nữa, bạn sẽ biết chúng có nghĩa là gì. Thật tốt hơn khi mắc lỗi ngay bây giờ một cách có chủ đích hơn là sau này một cách vô tình.

1. Bạn có thể sử dụng dấu trừ để tạo ra một số âm, như `-2`. Điều gì sẽ xảy ra nếu bạn đặt dấu cộng trước một số? Thử ví dụ với `+2`.
2. Còn với `2++2` thì sao? Hãy xem kết quả.
3. Nếu bạn có hai giá trị mà không có toán tử nào ở giữa, như `4 2`, điều gì sẽ xảy ra?
4. Nếu bạn gọi một hàm như `round(42.5)`, điều gì sẽ xảy ra nếu bạn bỏ qua một hoặc cả hai dấu ngoặc?

Bài tập 3

Nhớ rằng mỗi biểu thức có một giá trị, mỗi giá trị có một kiểu, và chúng ta có thể sử dụng hàm `type` để tìm ra kiểu của bất kỳ giá trị nào.

Giá trị của các biểu thức sau có kiểu gì? Hãy đoán tốt nhất cho mỗi biểu thức, sau đó sử dụng `type` để xác định.

- `765` `int`
- `2.718` `float`
- `'2 pi'` `str`
- `abs(-7)` `int`
- `abs(-7.0)` `float`
- `abs`
- `int`
- `type`

Bài tập 4

Các câu hỏi sau đây giúp bạn thực hành viết các biểu thức số học:

1. Có bao nhiêu giây trong 42 phút 42 giây?
2. Có bao nhiêu dặm trong 10 km? Gợi ý: có 1.61 km trong một dặm.
3. Nếu bạn chạy một cuộc đua 10 km trong 42 phút 42 giây, tốc độ trung bình của bạn là bao nhiêu giây mỗi dặm?
4. Tốc độ trung bình của bạn là bao nhiêu phút và giây mỗi dặm?
5. Tốc độ trung bình của bạn là bao nhiêu dặm mỗi giờ?

Nếu bạn đã biết về biến số, bạn có thể sử dụng chúng cho bài tập này.

Nếu không, bạn có thể thực hiện bài tập mà không cần biến số — và sau đó chúng ta sẽ tìm hiểu về chúng ở chương tiếp theo.

