



Trong chương 2, chúng ta đã thực hiện một số phép tính cơ bản với Python và bạn đã tìm hiểu về biến. Trong chương này, chúng ta sẽ làm việc với các thành phần khác trong chương trình Python: chuỗi, danh sách, tuple, và map. Bạn sẽ sử dụng chuỗi để hiển thị thông báo trong chương trình của mình (như các thông điệp “Get Ready” và “Game Over” trong trò chơi). Bạn cũng sẽ khám phá cách các danh sách, tuple, và map được sử dụng để lưu trữ tập hợp các thứ.

Chuỗi

Trong thuật ngữ lập trình, chúng ta thường gọi văn bản là chuỗi. Khi bạn nghĩ về chuỗi như một tập hợp các chữ cái, thuật ngữ này có vẻ hợp lý. Tất cả các chữ cái, số, và ký tự trong cuốn sách này có thể được xem là một chuỗi. Tên của bạn có thể là một



chuỗi, và địa chỉ của bạn cũng vậy. Thậm chí, chương trình Python đầu tiên chúng ta tạo trong Chương 1 đã sử dụng một chuỗi: “Hello World.”

TẠO CHUỖI

Trong Python, chúng ta tạo chuỗi bằng cách đặt dấu ngoặc kép xung quanh văn bản. Ví dụ: chúng ta có thể sử dụng biến fred (vốn không hữu ích lắm) từ Chương 2 và gán cho nó một chuỗi để làm nhãn, như sau:

```
fred = "Why do gorillas have big nostrils? Big fingers!!"
```

Sau đó, để xem nội dung bên trong fred, chúng ta có thể nhập print(fred), như sau:

```
>>> print(fred)
Why do gorillas have big nostrils? Big fingers!!
```

Bạn cũng có thể sử dụng dấu nháy đơn để tạo một chuỗi, như sau:

```
>>> fred = 'What is pink and fluffy? Pink fluff!!!'
>>> print(fred)
What is pink and fluffy? Pink fluff!!
```

Tuy nhiên, nếu bạn cố gắng nhập nhiều hơn một dòng văn bản cho chuỗi của mình chỉ bằng một dấu nháy đơn (') hoặc dấu nháy kép ("), hoặc nếu bạn bắt đầu bằng một loại dấu nháy và kết thúc bằng loại khác, bạn sẽ nhận được thông báo lỗi trong Python shell. Ví dụ, hãy nhập dòng sau:

```
>>> fred = "How do dinosaurs pay their bills?"
```

Bạn sẽ thấy kết quả như sau:

```
SyntaxError: EOL while scanning string literal
```

Đây là một thông báo lỗi phản nản về cú pháp vì bạn đã không tuân thủ quy tắc kết thúc chuỗi bằng dấu nháy đơn hoặc dấu nháy kép.

Cú pháp (syntax) có nghĩa là cách sắp xếp và thứ tự của từ trong một câu, hoặc trong trường hợp này, là cách sắp xếp và thứ tự của từ và ký hiệu trong chương trình. Vì vậy, `SyntaxError` có nghĩa là bạn đã làm gì đó không đúng theo thứ tự mà Python mong đợi, hoặc bạn đã bỏ sót điều gì đó mà Python đang mong đợi.

Giải pháp: Để sử dụng nhiều dòng trong chuỗi (được gọi là chuỗi nhiều dòng), bạn dùng ba dấu nháy đơn (`'''`) và nhấn Enter giữa các dòng, như sau:

```
>>> fred = '''How do dinosaurs pay their bills?
With tyrannosaurus checks!'''
```

Bây giờ, hãy in nội dung của `fred` để xem liệu điều này có hoạt động không:

```
>>> print(fred)
How do dinosaurs pay their bills?
With tyrannosaurus checks!
```

XỬ LÝ VẤN ĐỀ VỚI CHUỖI

Bây giờ, hãy xem xét một ví dụ phức tạp về chuỗi, điều này sẽ khiến Python hiển thị thông báo lỗi:

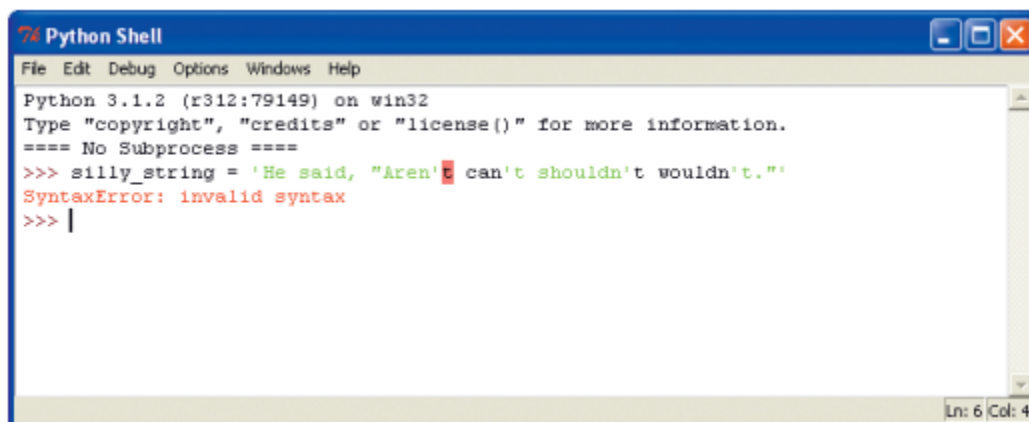
```
>>> silly_string = 'He said, "Aren't can't shouldn't
wouldn't."'
SyntaxError: invalid syntax
```

Trong dòng đầu tiên, chúng ta cố gắng tạo một chuỗi (được định nghĩa là biến `silly_string`) được bao quanh bởi dấu nháy đơn, nhưng lại chứa sự kết hợp của dấu nháy đơn trong các từ *can't*, *shouldn't*, và *wouldn't*, cùng với dấu nháy kép. Thật hỗn loạn!

Hãy nhớ rằng Python không thông minh như con người, vì vậy tất cả những gì nó thấy là một chuỗi chứa `He said, "Aren`, sau đó là một loạt ký tự khác mà nó không mong đợi. Khi Python gặp một dấu nháy (dấu nháy đơn hoặc kép), nó sẽ mong đợi chuỗi bắt đầu ngay sau dấu nháy đầu tiên và kết thúc sau dấu nháy phù hợp tiếp theo (dấu nháy đơn hoặc kép) trên cùng một dòng.

Trong trường hợp này, điểm bắt đầu chuỗi là dấu nháy đơn trước từ **He**, và điểm kết thúc chuỗi (theo Python) là dấu nháy đơn sau chữ **n** trong từ **Aren**.

IDLE sẽ làm nổi bật vị trí mà lỗi xảy ra:



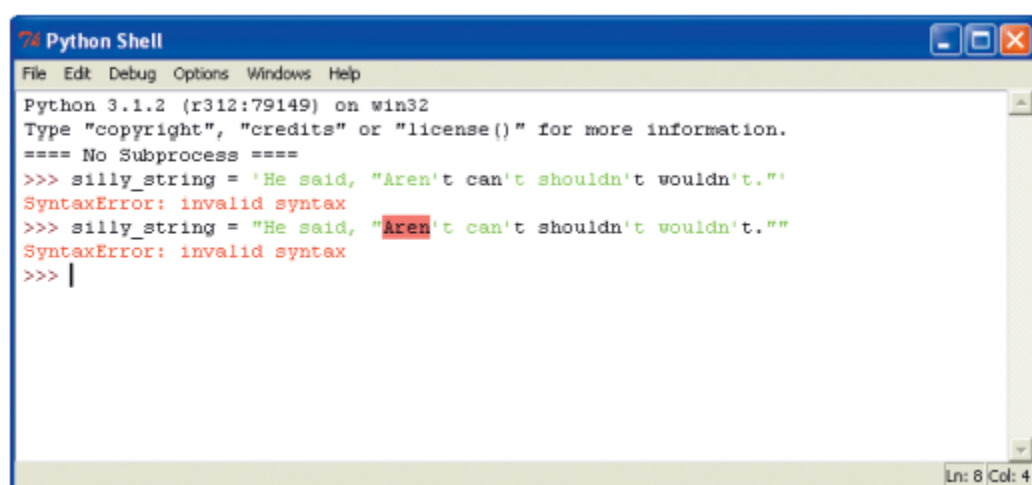
Dòng cuối cùng trong IDLE cho chúng ta biết loại lỗi đã xảy ra—trong trường hợp này là lỗi cú pháp (**syntax error**).

Sử dụng dấu nháy kép thay vì dấu nháy đơn vẫn sẽ dẫn đến lỗi:

```
>>> silly_string = "He said, "Aren't can't shouldn't wouldn't.""
```

```
SyntaxError: invalid syntax
```

Ở đây, Python thấy một chuỗi được giới hạn bởi dấu nháy kép, chứa các ký tự **He said**, (và một khoảng trắng). Mọi thứ sau chuỗi đó (từ **Aren't** trở đi) gây ra lỗi.

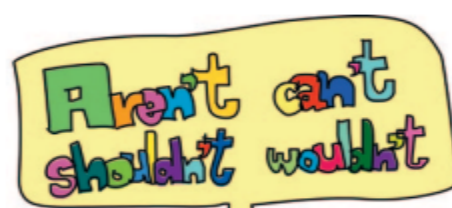
A screenshot of a Python Shell window titled "Python Shell". The window has a menu bar with "File", "Edit", "Debug", "Options", "Windows", and "Help". The main text area shows the following text:

```
Python 3.1.2 (r312:79149) on win32
Type "copyright", "credits" or "license()" for more information.
==== No Subprocess ====
>>> silly_string = 'He said, "Aren't can't shouldn't wouldn't."'
SyntaxError: invalid syntax
>>> silly_string = "He said, "Aren't can't shouldn't wouldn't."
SyntaxError: invalid syntax
>>> |
```

The status bar at the bottom right indicates "Ln: 8 Col: 4".

Điều này là vì, từ quan điểm của Python, tất cả những thứ bổ sung đó không nên có ở đó. Python tìm kiếm dấu nháy phù hợp tiếp theo và không biết bạn muốn nó làm gì với bất kỳ thứ gì theo sau dấu nháy đó trên cùng một dòng.

Giải pháp cho vấn đề này là sử dụng chuỗi nhiều dòng, mà chúng ta đã học trước đó, bằng cách dùng ba dấu nháy đơn (""") giúp chúng ta kết hợp cả dấu nháy đơn và dấu nháy kép trong chuỗi mà không gây lỗi. Thực tế, nếu sử dụng ba dấu nháy đơn, chúng ta có thể đặt bất kỳ sự kết hợp nào của dấu nháy đơn và dấu nháy kép vào trong chuỗi (miễn là không cố gắng đặt ba dấu nháy đơn vào trong chuỗi). Đây là phiên bản chuỗi không có lỗi của chúng ta:



```
silly_string = '''He said, "Aren't can't shouldn't  
wouldn't.''''
```

Nhưng khoan đã, còn nhiều điều nữa. Nếu bạn thực sự muốn sử dụng dấu nháy đơn hoặc dấu nháy kép để bao quanh một chuỗi trong Python, thay vì dùng ba dấu nháy đơn, bạn có thể thêm một dấu gạch chéo ngược (\) trước mỗi dấu nháy trong chuỗi. Đây gọi là escaping. Đây là cách để nói với Python, "Vâng, tôi biết tôi có dấu nháy trong chuỗi của mình, và tôi muốn bạn bỏ qua chúng cho đến khi bạn thấy dấu nháy kết thúc."

Việc sử dụng escaping có thể làm cho chuỗi khó đọc hơn, vì vậy tốt hơn là nên sử dụng chuỗi nhiều dòng. Tuy nhiên, bạn vẫn có thể gặp phải những đoạn mã sử dụng escaping, vì vậy hiểu được lý do tại sao có dấu gạch chéo ngược là điều tốt.

Dưới đây là một vài ví dụ về cách escaping hoạt động:

```
(1) >>> single_quote_str = 'He said, "Aren\'t can\'t  
shouldn\'t wouldn\'t.''  
  
(2) >>> double_quote_str = "He said, \"Aren't can't  
shouldn't wouldn't.\""  
  
>>> print(single_quote_str)  
He said, "Aren't can't shouldn't wouldn't."  
  
>>> print(double_quote_str)  
He said, "Aren't can't shouldn't wouldn't."
```

Đầu tiên, tại **1**, chúng ta tạo một chuỗi với dấu nháy đơn, sử dụng dấu gạch chéo ngược (\) trước dấu nháy đơn bên trong chuỗi đó. Tại **2**, chúng ta tạo một chuỗi với dấu nháy kép và sử dụng dấu gạch chéo ngược (\) trước các dấu nháy

kép trong chuỗi. Trong các dòng tiếp theo, chúng ta in các biến mà chúng ta vừa tạo. Lưu ý rằng ký tự gạch chéo ngược không xuất hiện trong các chuỗi khi chúng ta in chúng ra.

NHÚNG GIÁ TRỊ VÀO CHUỖI

Nếu bạn muốn hiển thị một thông báo sử dụng nội dung của một biến, bạn có thể nhúng các giá trị vào chuỗi bằng cách sử dụng `%s`, đây giống như một dấu hiệu cho một giá trị mà bạn muốn thêm vào sau. (Việc nhúng giá trị là cách nói của lập trình viên để "chèn giá trị vào"). Ví dụ, để Python tính toán hoặc lưu trữ số điểm bạn ghi được trong một trò chơi, và sau đó thêm nó vào một câu như "I scored ____ points," bạn sử dụng `%s` trong câu thay cho giá trị, và sau đó thông báo cho Python biết giá trị đó, như sau:

```
>>> myscore = 1000
>>> message = 'I scored %s points'
>>> print(message % myscore)
I scored 1000 points
```

Ở đây, chúng ta tạo biến `myscore` với giá trị 1000 và biến `message` với một chuỗi chứa các từ "I scored %s points", trong đó `%s` là một dấu chấm cho số điểm. Ở dòng tiếp theo, chúng ta gọi `print(message)` với ký tự `%` để thông báo cho Python thay thế `%s` bằng giá trị được lưu trữ trong biến `myscore`. Kết quả khi in thông điệp này sẽ là "I scored 1000 points". Chúng ta không cần phải sử dụng một biến cho giá trị này. Ta cũng có thể thực hiện ví dụ tương tự và chỉ cần sử dụng `print(message % 1000)`.

Chúng ta cũng có thể truyền vào các giá trị khác cho dấu chấm `%s`, sử dụng các biến khác nhau, như trong ví dụ sau:

```
>>> joke_text = '%s: a device for finding furniture in
the dark'
```

```
>>> bodypart1 = 'Knee'
>>> bodypart2 = 'Shin'
>>> print(joke_text % bodypart1)
Knee: a device for finding furniture in the dark
>>> print(joke_text % bodypart2)
Shin: a device for finding furniture in the dark
```

Ở đây, chúng ta tạo ba biến. Biến đầu tiên, `joke_text`, bao gồm chuỗi với dấu `%`s. Các biến khác là `bodypart1` và `bodypart2`. Chúng ta có thể in biến `joke_text`, và một lần nữa sử dụng toán tử `%` để thay thế nó bằng nội dung của các biến `bodypart1` và `bodypart2`, từ đó tạo ra các thông điệp khác nhau.



Bạn cũng có thể sử dụng nhiều dấu chấm (placeholders) trong một chuỗi, như sau:

```
>>> nums = 'What did the number %s say to the number %s? Nice belt!!'
>>> print(nums % (0, 8))

What did the number 0 say to the number 8? Nice belt!!
```

Khi sử dụng nhiều dấu chấm (placeholders), hãy chắc chắn rằng bạn đóng gói các giá trị thay thế trong dấu ngoặc đơn, như trong ví dụ. Thứ tự của các giá trị là thứ tự mà chúng sẽ được sử dụng trong chuỗi.

NHÂN CHUỖI

Cái gì là 10 nhân với 5? Đáp án là 50, tất nhiên rồi. Nhưng 10 nhân với `a` thì sao? Đây là câu trả lời của Python:

```
>>> print(10 * 'a')
```

```
aaaaaaaaaaaa
```

Các lập trình viên Python có thể sử dụng phương pháp này để căn chỉnh các chuỗi với một số khoảng trắng cụ thể khi hiển thị thông báo trong shell, ví dụ như vậy. Vậy còn việc in một chữ cái trong shell thì sao? Bạn có thể chọn **File -> New Window** và nhập mã sau:

```
spaces = ' ' * 25
```

```
print('%s 12 Butts Wynd' % spaces)
```

```
print('%s Twinklebottom Heath' % spaces)
```

```
print('%s West Snoring' % spaces)
```

```
print()
```

```
print()
```

```
print('Dear Sir')
```

```
print()
```

```
print('I wish to report that tiles are missing from  
the')
```

```
print('outside toilet roof.')
```

```
print('I think it was bad wind the other night that  
blew them away.')
```

```
print()
```

```
print('Regards')
```

```
print('Malcolm Dithering')
```

Khi bạn đã nhập mã vào cửa sổ shell, chọn **File -> Save As** và đặt tên cho tệp của bạn là myletter.py.

*Từ bây giờ, khi bạn thấy Save As: somefilename.py ở trên một đoạn mã, bạn sẽ biết rằng bạn cần chọn **File -> New Window**, nhập mã vào cửa sổ xuất hiện và sau đó lưu lại như chúng ta đã làm trong ví dụ này.*

Trong dòng đầu tiên của ví dụ này, chúng ta tạo biến spaces bằng cách nhân một ký tự khoảng trắng với 25. Sau đó, chúng ta sử dụng biến đó trong ba dòng tiếp theo để căn chỉnh văn bản sang phía bên phải của shell. Bạn có thể thấy kết quả của các câu lệnh print dưới đây:



```
Python Shell
File Edit Debug Options Windows Help
>>>
12 Butts Wynd
Twinklebottom Heath
West Snoring

Dear Sir

I wish to report that tiles are missing from the
outside toilet roof.
I think it was bad wind the other night that blew
them away.

Regards
Malcolm Dithering
>>>
>>>
>>>
>>>
>>>
Ln: 68 Col: 4
```

Ngoài việc sử dụng phép nhân để căn chỉnh, chúng ta cũng có thể sử dụng nó để làm đầy màn hình với những thông báo gây phiền phức. Hãy thử ví dụ này cho chính bạn:

```
>>> print(1000 * 'snirt')
```

DANH SÁCH MẠNH HƠN CHUỖI



“Spider legs, toe of frog, eye of newt, bat wing, slug butter, and snake dandruff” không phải là một danh sách mua sắm bình thường (trừ khi bạn là một phù thủy), nhưng chúng ta sẽ sử dụng nó như là ví dụ đầu tiên về sự khác biệt giữa chuỗi và danh sách.

Chúng ta có thể lưu trữ danh sách các món đồ này trong biến `wizard_list` bằng cách sử dụng một chuỗi như sau:

```
>>> wizard_list = 'spider legs, toe of frog, eye of newt, bat wing, slug butter, snake dandruff'

>>> print(wizard_list)

spider legs, toe of frog, eye of newt, bat wing, slug butter, snake dandruff
```

Tuy nhiên, chúng ta cũng có thể tạo một danh sách, một loại đối tượng Python "ma thuật" mà chúng ta có thể thao tác. Đây là cách các món đồ này sẽ trông như thế nào khi được viết dưới dạng danh sách:

```
>>> wizard_list = ['spider legs', 'toe of frog', 'eye of newt', 'bat wing', 'slug butter', 'snake dandruff']

>>> print(wizard_list)

['spider legs', 'toe of frog', 'eye of newt', 'bat wing', 'slug butter', 'snake dandruff']
```

Việc tạo một danh sách mất nhiều công sức hơn so với việc tạo một chuỗi, nhưng danh sách lại hữu ích hơn vì chúng ta có thể thao tác với nó. Ví dụ, chúng ta có thể in món đồ thứ ba trong `wizard_list` (eye of newt) bằng cách nhập vị trí của nó trong danh sách (gọi là chỉ số vị trí) trong dấu ngoặc vuông (`[]`), như sau:

```
>>> print(wizard_list[2])
```

```
eye of newt
```

Chà? Nó không phải là món thứ ba trong danh sách sao? Đúng vậy, nhưng danh sách bắt đầu từ chỉ số vị trí 0, vì vậy món thứ nhất có chỉ số 0, món thứ hai có chỉ số 1, và món thứ ba có chỉ số 2. Điều này có thể không hợp lý với con người, nhưng lại hợp lý với máy tính.

Chúng ta cũng có thể dễ dàng thay đổi một món đồ trong danh sách hơn là trong một chuỗi. Có thể thay vì "eye of newt", chúng ta cần "snail tongue". Đây là cách chúng ta sẽ làm điều đó với danh sách:

```
>>> wizard_list[2] = 'snail tongue'
```

```
>>> print(wizard_list)
```

```
['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'snake dandruff']
```

Điều này thay thế món đồ ở vị trí chỉ số 2 (trước đây là "eye of newt") thành "snail tongue".

Một tùy chọn khác là hiển thị một phần của các mục trong danh sách. Chúng ta làm điều này bằng cách sử dụng dấu hai chấm (:) trong dấu ngoặc vuông. Ví dụ, nhập đoạn mã sau để xem các món đồ từ thứ ba đến thứ năm trong danh sách (một bộ nguyên liệu tuyệt vời cho một chiếc sandwich tuyệt hảo):



```
>>> print(wizard_list[2:5])  
['snail tongue', 'bat wing', 'slug butter']
```

Viết [2:5] tương đương với việc nói, "Hiển thị các món đồ từ vị trí chỉ số 2 đến vị trí chỉ số 5 (nhưng không bao gồm vị trí 5)"—nói cách khác, là các món đồ ở vị trí chỉ số 2, 3, và 4.

Danh sách có thể được sử dụng để lưu trữ đủ loại mục, chẳng hạn như các số:

```
>>> some_numbers = [1, 2, 5, 10, 20]
```

Chúng cũng có thể chứa các chuỗi:

```
>>> some_strings = ['Which', 'Witch', 'Is', 'Which']
```

Danh sách có thể chứa sự pha trộn giữa số và chuỗi:

```
>>> numbers_and_strings = ['Why', 'was', 6, 'afraid',  
'of', 7, 'because', 7, 8, 9]  
  
>>> print(numbers_and_strings)  
['Why', 'was', 6, 'afraid', 'of', 7, 'because', 7, 8,  
9]
```

Danh sách thậm chí có thể lưu trữ các danh sách khác:

```
>>> numbers = [1, 2, 3, 4]

>>> strings = ['I', 'kicked', 'my', 'toe', 'and',
               'it', 'is', 'sore']

>>> mylist = [numbers, strings]

>>> print(mylist)

[[1, 2, 3, 4], ['I', 'kicked', 'my', 'toe', 'and',
               'it', 'is', 'sore']]
```

Ví dụ về danh sách chứa danh sách này tạo ra ba biến: numbers với bốn số, strings với tám chuỗi, và mylist chứa numbers và strings. Danh sách thứ ba (mylist) chỉ có hai phần tử vì đó là một danh sách các tên biến, không phải là nội dung của các biến đó.

THÊM ITEMS VÀO DANH SÁCH

Để thêm các mục vào danh sách, ta sử dụng hàm append. Hàm là một đoạn mã yêu cầu Python thực hiện một tác vụ nào đó. Trong trường hợp này, append thêm một mục vào cuối danh sách.

Ví dụ, để thêm một "bear burp" (một món chắc chắn tồn tại) vào danh sách mua sắm của phù thủy, ta làm như sau:

```
>>> wizard_list.append('bear burp')

>>> print(wizard_list)

['spider legs', 'toe of frog', 'snail tongue', 'bat
wing', 'slug butter', 'snake dandruff', 'bear burp']
```

Bạn có thể tiếp tục thêm các món đồ kỳ diệu vào danh sách của phù thủy bằng cách sử dụng `append`:

```
>>> wizard_list.append('mandrake')
>>> wizard_list.append('hemlock')
>>> wizard_list.append('swamp gas')
```

Kết quả danh sách bây giờ sẽ là:

```
>>> print(wizard_list)

['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'snake dandruff', 'bear burp', 'mandrake', 'hemlock', 'swamp gas']
```

XOÁ ITEMS KHỎI DANH SÁCH

Để xóa một mục khỏi danh sách, ta sử dụng lệnh `del` (viết tắt của "delete"). Ví dụ, để xóa món đồ thứ sáu trong danh sách của phù thủy, là "snake dandruff", ta làm như sau:

```
>>>del wizard_list[5]

>>>print(wizard_list)
```

```
['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'bear burp', 'mandrake', 'hemlock', 'swamp gas']
```

Lưu ý rằng chỉ số bắt đầu từ 0, vì vậy wizard_list[5] thực sự ám chỉ món đồ thứ sáu trong danh sách.

Và đây là cách để loại bỏ các mục mà chúng ta vừa thêm vào (mandrake, hemlock và swamp gas):

```
>>>del wizard_list[8]

>>>del wizard_list[7]

>>>del wizard_list[6]

>>>print(wizard_list)

['spider legs', 'toe of frog', 'snail tongue', 'bat wing', 'slug butter', 'bear burp']
```

DANH SÁCH TOÁN HỌC

Chúng ta có thể nối các danh sách lại với nhau bằng cách sử dụng dấu cộng (+), giống như cộng các số, ví dụ:

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = ['I', 'tripped', 'over', 'and', 'hit', 'the', 'floor']
>>> print(list1 + list2)
[1, 2, 3, 4, 'I', 'tripped', 'over', 'and', 'hit', 'the', 'floor']
```

Chúng ta cũng có thể cộng hai danh sách và gán kết quả vào một biến khác:

```
>>> list1 = [1, 2, 3, 4]
>>> list2 = ['I', 'ate', 'chocolate', 'and',
'I', 'want', 'more']
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4, 'I', 'ate', 'chocolate',
'and', 'I', 'want', 'more']
```

Chúng ta cũng có thể nhân một danh sách với một số. Ví dụ, để nhân list1 với 5, ta viết list1 * 5:

```
>>> list1 = [1, 2]
>>> print(list1 * 5)
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

Điều này thực sự đang yêu cầu Python lặp lại list1 năm lần, kết quả là 1, 2, 1, 2, 1, 2, 1, 2, 1, 2.

Mặt khác, phép chia (/) và phép trừ (-) chỉ đưa ra lỗi, như trong các ví dụ sau:

```
>>> list1 / 20
Traceback (most recent call last):
  File "", line 1, in
    list1 / 20
TypeError: unsupported operand type(s) for /: 'list'
and 'int'
```

```
>>> list1 - 20
Traceback (most recent call last):
  File "", line 1, in
    list1 - 20
TypeError: unsupported operand type(s) for -: 'list'
and 'int'
```

Nhưng tại sao lại như vậy? Việc nối các danh sách với dấu cộng (+) và lặp lại các danh sách với dấu sao (*) là những thao tác khá đơn giản. Những khái niệm này cũng khá hợp lý trong thực tế. Ví dụ, nếu tôi đưa cho bạn hai danh sách mua sắm giấy và nói, "Hãy cộng hai danh sách này lại," bạn có thể viết tất cả các mục trong một tờ giấy khác theo thứ tự, nối chúng lại với nhau. Điều tương tự cũng xảy ra nếu tôi nói, "Nhân danh sách này với 3." Bạn có thể tưởng tượng việc viết ra một danh sách của tất cả các mục ba lần trên một tờ giấy khác.

Nhưng làm sao bạn có thể chia một danh sách? Ví dụ, hãy xem xét cách bạn chia một danh sách gồm sáu số (từ 1 đến 6) thành hai phần. Dưới đây là ba cách khác nhau:

[1, 2, 3]	[4, 5, 6]
[1]	[2, 3, 4, 5, 6]
[1, 2, 3, 4]	[5, 6]

Liệu chúng ta sẽ chia danh sách ở giữa, chia sau phần tử đầu tiên, hay chỉ chọn một vị trí ngẫu nhiên và chia nó ở đó? Không có câu trả lời đơn giản, và khi bạn yêu cầu Python chia một danh sách, nó cũng không biết phải làm gì. Đó là lý do tại sao Python phản hồi với một lỗi.

Điều tương tự cũng xảy ra khi bạn cố gắng thêm bất cứ thứ gì khác ngoài một danh sách vào một danh sách. Bạn không thể



làm điều đó. Ví dụ, đây là điều xảy ra khi chúng ta thử thêm số 50 vào list1:

```
>>> list1 + 50
Traceback (most recent call last):
  File "", line 1, in
    list1 + 50
TypeError: can only concatenate list (not "int") to
list
```

Tại sao lại có lỗi ở đây? Vậy thêm 50 vào một danh sách có nghĩa là gì? Có phải là thêm 50 vào mỗi phần tử? Nhưng nếu các phần tử không phải là số thì sao? Có phải là thêm số 50 vào cuối hoặc đầu danh sách không?

Trong lập trình máy tính, các lệnh phải hoạt động một cách nhất quán mỗi khi bạn nhập chúng. Máy tính, vì là một cỗ máy đơn giản, chỉ nhìn nhận mọi thứ dưới dạng trắng đen. Yêu cầu nó đưa ra một quyết định phức tạp, và nó sẽ không thể hiểu được, dẫn đến lỗi.

TUPLES

Một tuple giống như một danh sách nhưng sử dụng dấu ngoặc đơn, như trong ví dụ sau:

```
>>> fibs = (0, 1, 1, 2, 3)
>>> print(fibs[3])
```

2

Ở đây, chúng ta định nghĩa biến fibs là các số 0, 1, 1, 2 và 3. Sau đó, giống như một danh sách, chúng ta in ra phần tử tại vị trí chỉ mục 3 trong tuple bằng cách sử dụng print(fibs[3]).

Điểm khác biệt chính giữa tuple và danh sách là một tuple không thể thay đổi sau khi bạn đã tạo ra nó. Ví dụ, nếu chúng ta cố gắng thay thế giá trị đầu tiên trong tuple `fibs` bằng số 4 (giống như khi thay đổi giá trị trong `wizard_list`), chúng ta sẽ nhận được một thông báo lỗi:

```
>>> fibs[0] = 4
```

```
Traceback (most recent call last):
```

```
File "", line 1, in
```

```
fibs[0] = 4
```

```
TypeError: 'tuple' object does not support item  
assignment
```

Bạn sẽ sử dụng một tuple thay vì một danh sách khi bạn cần một cấu trúc dữ liệu không thay đổi. Nếu bạn tạo một tuple với hai phần tử bên trong, nó sẽ luôn giữ nguyên hai phần tử đó, không thay đổi. Điều này đặc biệt hữu ích khi bạn muốn đảm bảo rằng dữ liệu của mình không bị thay đổi trong suốt quá trình sử dụng.

CÁC BẢN ĐỒ TRONG PYTHON SẼ KHÔNG GIÚP BẠN TÌM ĐƯỜNG

Trong Python, một map (hay còn gọi là dict, viết tắt của từ dictionary) là một tập hợp các đối tượng, giống như danh sách và tuple. Sự khác biệt giữa maps và danh sách hoặc tuple là mỗi mục trong map có một key (khóa) và một value (giá trị) tương ứng.

Ví dụ, giả sử chúng ta có một danh sách những người và môn thể thao yêu thích của họ. Chúng ta có thể lưu trữ thông tin này trong một danh sách Python, với tên của người đó và môn thể thao yêu thích của họ, như sau:

```
favorite_sports = ['Ralph Williams, Football',
```

```
'Michael Tippett, Basketball',  
'Edward Elgar, Baseball',  
'Rebecca Clarke, Netball',  
'Ethel Smyth, Badminton',  
'Frank Bridge, Rugby']
```

Nếu tôi hỏi bạn môn thể thao yêu thích của Rebecca Clarke, bạn có thể lướt qua danh sách và tìm thấy câu trả lời là netball. Nhưng nếu danh sách chứa 100 người (hoặc nhiều hơn), việc tìm kiếm sẽ trở nên khó khăn và mất thời gian.

Bây giờ, nếu chúng ta lưu trữ cùng thông tin này trong một map, với tên người làm khóa và môn thể thao yêu thích làm giá trị, mã Python sẽ trông như thế này:



```
>>> favorite_sports = {'Ralph Williams': 'Football',  
                        'Michael Tippett': 'Basketball',  
                        'Edward Elgar': 'Baseball',  
                        'Rebecca Clarke': 'Netball',  
                        'Ethel Smyth': 'Badminton',  
                        'Frank Bridge': 'Rugby'}
```

Ở đây, chúng ta sử dụng dấu hai chấm (:) để tách mỗi khóa khỏi giá trị của nó, và mỗi khóa và giá trị được bao quanh bởi dấu nháy đơn. Lưu ý rằng các mục trong

map được bao quanh bởi dấu ngoặc nhọn {}, thay vì dấu ngoặc đơn hoặc ngoặc vuông.

Kết quả là một bản đồ (mỗi khóa trở đến một giá trị cụ thể), như trong bảng dưới đây:

Bảng 3-1: Các khóa trở đến các giá trị trong bản đồ các môn thể thao yêu thích

Key	Value
Ralph Williams	Football
Michael Tippett	Basketball
Edward Elgar	Baseball
Rebecca Clarke	Netball
Ethel Smyth	Badminton
Frank Bridge	Rugby

Bây giờ, để lấy thông tin môn thể thao yêu thích của Rebecca Clarke, chúng ta truy cập vào bản đồ `favorite_sports` bằng tên của cô ấy làm khóa, như sau:

```
>>> print(favorite_sports['Rebecca Clarke'])
```

Netball

Và kết quả là môn thể thao yêu thích của cô ấy là "Netball".

Để xóa một giá trị trong bản đồ, ta sử dụng khóa của nó. Ví dụ, đây là cách xóa Ethel Smyth:

```
>>> del favorite_sports['Ethel Smyth']
```

```
>>> print(favorite_sports)
```

```
{'Rebecca Clarke': 'Netball', 'Michael Tippett': 'Basketball', 'Ralph Williams': 'Football', 'Edward Elgar': 'Baseball', 'Frank Bridge': 'Rugby'}
```

Để thay thế một giá trị trong bản đồ, ta cũng sử dụng khóa của nó:

```
>>> favorite_sports['Ralph Williams'] = 'Ice Hockey'
>>> print(favorite_sports)

{'Rebecca Clarke': 'Netball', 'Michael Tippett': 'Basketball', 'Ralph Williams': 'Ice Hockey', 'Edward Elgar': 'Baseball', 'Frank Bridge': 'Rugby'}
```

Chúng ta thay thế môn thể thao yêu thích "Football" bằng "Ice Hockey" bằng cách sử dụng khóa "Ralph Williams".

Như bạn có thể thấy, làm việc với bản đồ (maps) khá giống với làm việc với danh sách (lists) và bộ dữ liệu (tuples), ngoại trừ việc bạn không thể kết hợp các bản đồ với toán tử cộng (+). Nếu bạn thử làm vậy, bạn sẽ nhận được thông báo lỗi:

```
>>> favorite_sports = {'Rebecca Clarke': 'Netball',
                        'Michael Tippett': 'Basketball',
                        'Ralph Williams': 'Ice Hockey',
                        'Edward Elgar': 'Baseball',
                        'Frank Bridge': 'Rugby'}

>>> favorite_colors = {'Malcolm Warner' : 'Pink polka dots',
                        'James Baxter' : 'Orange stripes',
                        'Sue Lee' : 'Purple paisley'}
```

```
>>> favorite_sports + favorite_colors

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'dict'
and 'dict'
```

Việc kết hợp các bản đồ không hợp lý với Python, vì vậy nó sẽ ném ra lỗi.

NHỮNG GÌ BẠN ĐÃ HỌC ĐƯỢC

Trong chương này, bạn đã học cách Python sử dụng chuỗi (strings) để lưu trữ văn bản, và cách sử dụng danh sách (lists) và bộ dữ liệu (tuples) để xử lý nhiều mục. Bạn đã thấy rằng các mục trong danh sách có thể thay đổi, và bạn có thể kết hợp một danh sách với danh sách khác, nhưng các giá trị trong bộ dữ liệu không thể thay đổi. Bạn cũng đã học cách sử dụng bản đồ (maps) để lưu trữ các giá trị với các khóa (keys) xác định chúng.

CÂU ĐÓ LẬP TRÌNH

Dưới đây là một số thí nghiệm bạn có thể thử. Các câu trả lời có thể được tìm thấy tại <http://python-for-kids.com/>.

#1: Những sở thích yêu thích

Hãy tạo một danh sách về sở thích yêu thích của bạn và gán cho biến tên games. Sau đó, hãy tạo một danh sách về những món ăn yêu thích của bạn và gán cho biến tên foods. Kết hợp hai danh sách này và đặt kết quả vào biến favorites. Cuối cùng, hãy in biến favorites.

#2: Đếm số người tham chiến

Nếu có 3 tòa nhà với 25 ninja ẩn nấp trên mỗi mái nhà và 2 đường hầm với 40 samurai ẩn nấp trong mỗi đường hầm, thì tổng số ninja và samurai chuẩn bị chiến đấu là bao nhiêu? (Bạn có thể làm điều này với một phương trình trong Python shell.)

#3: Lời chào!

Tạo hai biến: một biến chứa tên của bạn và một biến chứa họ của bạn. Sau đó, tạo một chuỗi và sử dụng các dấu chèn (placeholders) để in tên của bạn kèm theo một thông điệp, ví dụ như “Chào bạn, Brando Ickett!”

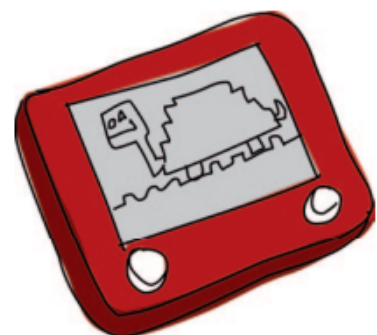


Trong Python, con *turtle* giống như một con rùa trong thế giới thực. Chúng ta biết một con rùa là loài bò sát di chuyển rất chậm và mang trên lưng ngôi nhà của mình. Trong thế giới Python, một con rùa là một mũi tên màu đen di chuyển chậm trên màn hình. Thực ra, vì con rùa Python để lại dấu vết khi di chuyển trên màn hình, nó thực tế giống một con Ốc sên hoặc một con sên hơn là một con rùa.

Turtle là một công cụ tuyệt vời để học một số kiến thức cơ bản về đồ họa máy tính. Vì vậy, trong chương này, chúng ta sẽ sử dụng một Python turtle để vẽ một số hình dạng và đường đơn giản.

SỬ DỤNG MÔ-ĐUN TURTLE TRONG PYTHON

Một mô-đun trong Python là một cách cung cấp mã nguồn hữu ích để được sử dụng bởi chương trình khác (mô-đun có thể chứa các hàm mà chúng ta có thể sử dụng). Chúng ta sẽ tìm hiểu thêm về mô-đun trong



Chương 7. Python có một mô-đun đặc biệt gọi là **turtle** mà chúng ta có thể sử dụng để học cách máy tính vẽ hình ảnh trên màn hình. Mô-đun turtle là một cách lập trình đồ họa vector, cơ bản là vẽ bằng các đường thẳng, chấm và đường cong đơn giản.

Hãy xem cách mô-đun turtle hoạt động. Đầu tiên, mở Python shell bằng cách nhấp vào biểu tượng trên màn hình (hoặc nếu bạn đang sử dụng Ubuntu, chọn **Applications** → **Programming** → **IDLE**). Tiếp theo, yêu cầu Python sử dụng mô-đun turtle bằng cách nhập lệnh sau:

```
>>> import turtle
```

Việc nhập lệnh này thông báo cho Python rằng bạn muốn sử dụng mô-đun turtle.

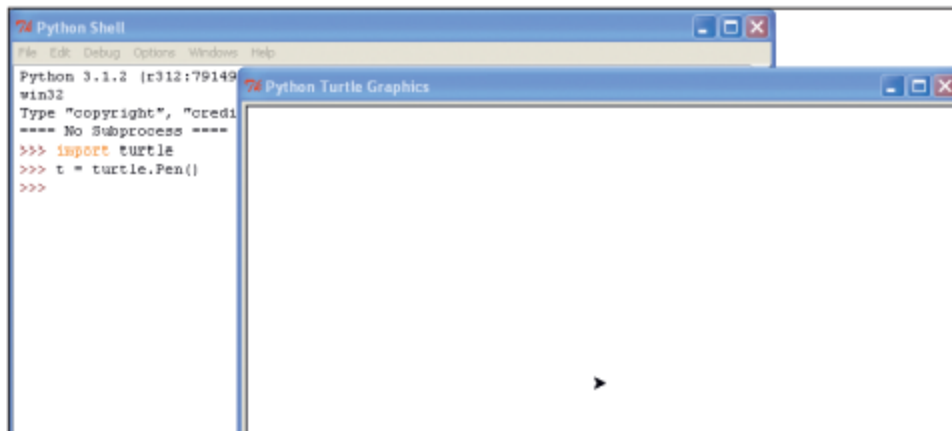
Lưu ý: Nếu bạn đang sử dụng Ubuntu và gặp lỗi ở bước này, bạn có thể cần cài đặt tkinter. Để làm điều đó, hãy mở Ubuntu Software Center và nhập python-tk vào ô tìm kiếm. “Tkinter – Writing Tk Applications with Python” sẽ xuất hiện trong cửa sổ. Hãy nhấp Install để cài đặt gói này.

TẠO MỘT CANVAS

Bây giờ, sau khi đã nhập mô-đun turtle, chúng ta cần tạo một canvas—một không gian trắng để vẽ, giống như một bức tranh của họa sĩ. Để làm điều này, chúng ta gọi hàm Pen từ mô-đun turtle, hàm này tự động tạo ra một canvas. Nhập lệnh này vào Python shell:

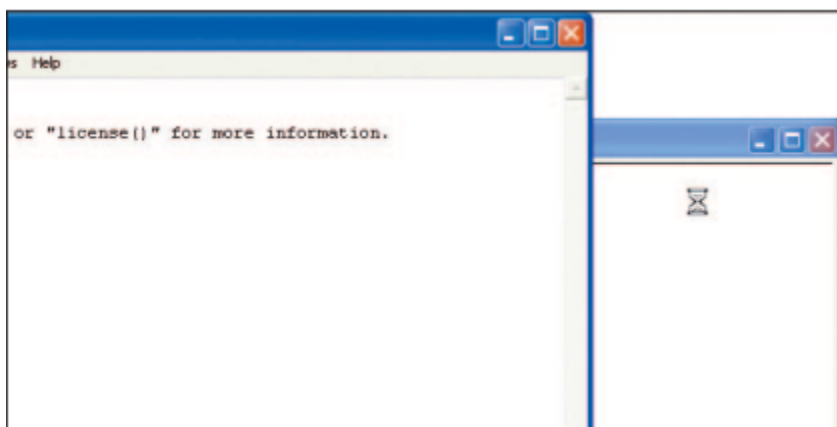
```
>>> t = turtle.Pen()
```

Bạn sẽ thấy một hộp trống (canvas), với một mũi tên ở giữa, trông như thế này:



Mũi tên ở giữa màn hình chính là con turtle, và đúng là nó không giống như một con rùa chút nào.

Nếu cửa sổ Turtle xuất hiện phía sau cửa sổ Python Shell, bạn có thể thấy rằng nó không hoạt động đúng cách. Khi di chuyển chuột vào cửa sổ Turtle, con trỏ sẽ chuyển thành hình đồng hồ cát, như thế này:



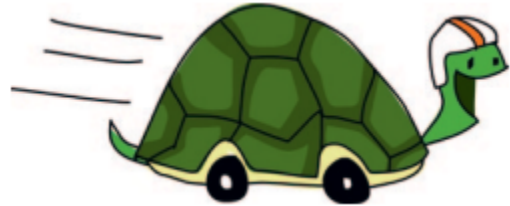
Điều này có thể xảy ra vì một số lý do: bạn chưa mở shell từ biểu tượng trên màn hình (nếu bạn đang sử dụng Windows hoặc Mac), bạn đã nhấp vào IDLE (Python GUI) trong menu Start của Windows, hoặc IDLE không được cài đặt đúng cách. Hãy thử thoát và khởi động lại shell từ biểu tượng trên màn hình. Nếu điều đó không thành công, hãy thử sử dụng Python console thay vì shell, như sau:

- Trong Windows, chọn **Start** → **All Programs**, sau đó trong nhóm **Python 3.2**, nhấp vào **Python (command line)**.

- Trong Mac OS X, nhấp vào biểu tượng **Spotlight** ở góc trên bên phải màn hình và nhập **Terminal** vào ô tìm kiếm. Sau đó nhập **python** khi terminal mở ra.
- Trong Ubuntu, mở terminal từ menu **Applications** và nhập **python**.

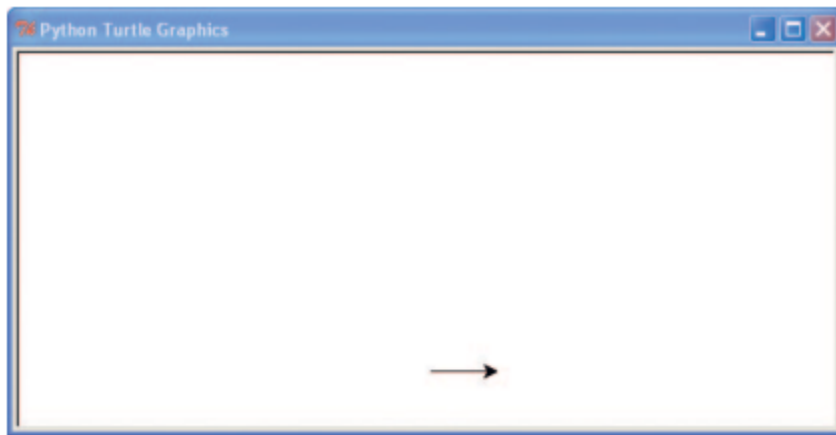
DI CHUYỂN TURTLE

Bạn gửi các lệnh cho turtle bằng cách sử dụng các hàm có sẵn trên biến `t` mà chúng ta vừa tạo, tương tự như việc sử dụng hàm `Pen` trong mô-đun `turtle`. Ví dụ, lệnh `forward` yêu cầu turtle di chuyển về phía trước. Để yêu cầu turtle tiến 50 pixel, hãy nhập lệnh sau:



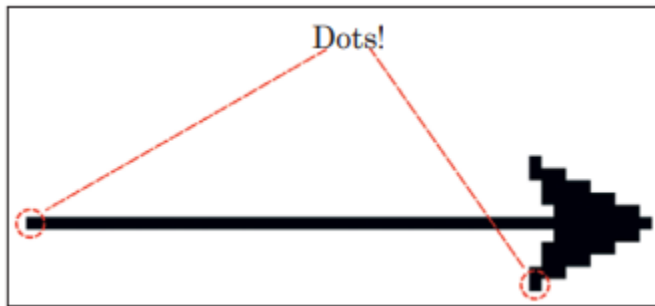
```
>>> t.forward(50)
```

Bạn sẽ thấy một chuyển động như sau:



Con turtle đã di chuyển về phía trước 50 pixel. Một pixel là một điểm duy nhất trên màn hình - yếu tố nhỏ nhất có thể được biểu diễn. Mọi thứ bạn nhìn thấy trên màn hình máy tính của mình đều được tạo thành từ các pixel, những chấm nhỏ hình vuông. Nếu bạn có thể phóng to lên canvas và đường đi mà turtle vẽ ra,

bạn sẽ thấy mũi tên đại diện cho con đường của turtle chỉ là một loạt các pixel. Đó chính là đồ họa máy tính đơn giản.



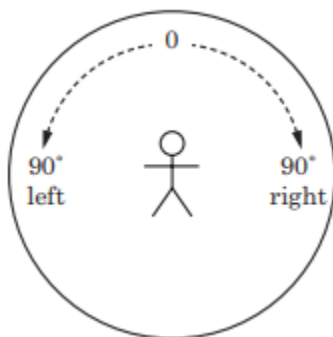
Bây giờ, chúng ta sẽ yêu cầu turtle quay sang trái 90 độ với lệnh sau:

```
>>> t.left(90)
```

Nếu bạn chưa học về độ (degree), đây là cách bạn có thể hiểu về chúng. Hãy tưởng tượng bạn đang đứng ở trung tâm của một vòng tròn:

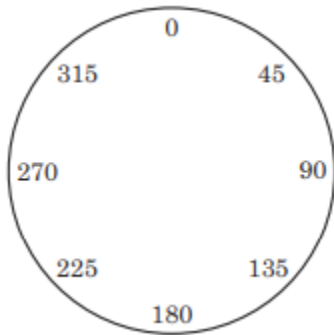
- Hướng bạn đang đối diện là 0 độ.
- Nếu bạn giơ tay trái ra, đó là 90 độ về bên trái.
- Nếu bạn giơ tay phải ra, đó là 90 độ về bên phải.

Bạn có thể thấy sự quay 90 độ sang trái hoặc phải ở đây:



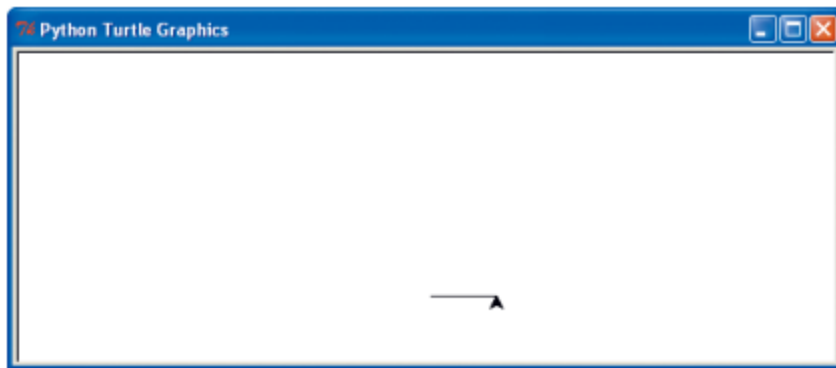
Nếu bạn tiếp tục di chuyển quanh vòng tròn theo hướng bên phải từ nơi tay phải của bạn đang chỉ, 180 độ sẽ là phía sau bạn, 270 độ sẽ là hướng mà tay trái

của bạn đang chỉ, và 360 độ sẽ là vị trí bạn bắt đầu; các độ từ 0 đến 360. Các độ trong một vòng tròn đầy đủ khi quay sang phải có thể thấy như sau, mỗi lần quay là 45 độ:



Khi turtle của Python quay sang trái, nó xoay để hướng về phía mới (giống như bạn xoay cơ thể để đối diện với hướng mà tay bạn đang chỉ 90 độ về phía trái).

Lệnh `t.left(90)` sẽ làm mũi tên quay lên (vì nó ban đầu hướng sang phải):



Lưu ý: Khi bạn gọi lệnh `t.left(90)`, điều đó tương đương với việc gọi `t.right(270)`. Điều này cũng đúng với lệnh `t.right(90)`, tương đương với việc gọi `t.left(270)`. Hãy tưởng tượng vòng tròn và làm theo các độ.

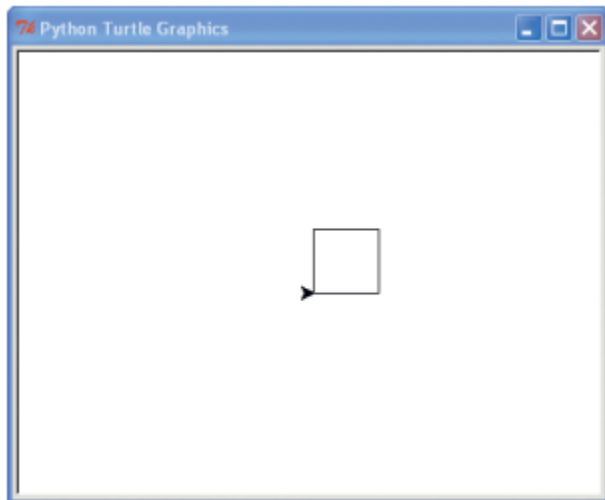
Bây giờ, chúng ta sẽ vẽ một hình vuông. Thêm mã sau vào các dòng mã bạn đã nhập trước đó:

```
>>> t.forward(50)
```

```
>>> t.left(90)
```

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
```

Turtle của bạn nên đã vẽ một hình vuông và hiện tại nó đang hướng theo cùng một hướng mà nó bắt đầu.



Để xóa canvas, bạn nhập lệnh reset. Lệnh này sẽ xóa canvas và đưa turtle trở lại vị trí ban đầu.

```
>>> t.reset()
```

Bạn cũng có thể sử dụng lệnh clear, lệnh này chỉ xóa màn hình mà không làm thay đổi vị trí của turtle.

```
>>> t.clear()
```

Chúng ta cũng có thể xoay turtle sang phải hoặc di chuyển nó lùi lại. Lệnh `up` giúp nâng bút lên khỏi trang (nói cách khác, yêu cầu turtle ngừng vẽ), còn lệnh `down` giúp bắt đầu vẽ lại. Các lệnh này được viết tương tự như các lệnh khác mà chúng ta đã sử dụng.

Hãy thử vẽ thêm một hình bằng một số lệnh này. Lần này, chúng ta sẽ yêu cầu turtle vẽ hai đường thẳng. Nhập mã sau:

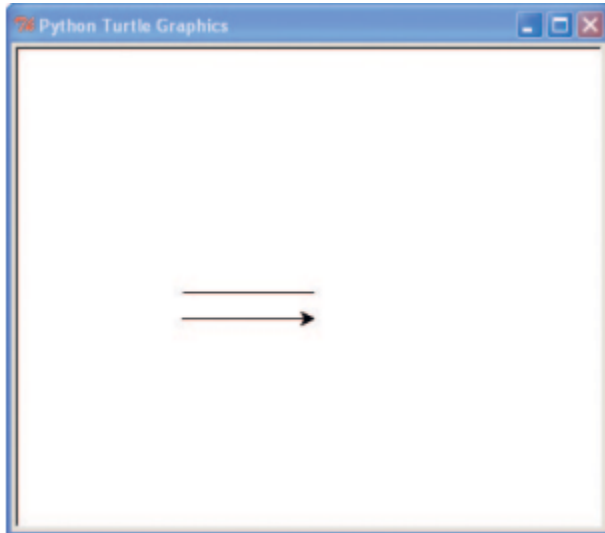
```
>>> t.reset()
>>> t.backward(100)
>>> t.up()
>>> t.right(90)
>>> t.forward(20)
>>> t.left(90)
>>> t.down()
>>> t.forward(100)
```

Đầu tiên, chúng ta gọi `t.reset()` để làm mới canvas và di chuyển turtle về lại vị trí bắt đầu. Tiếp theo, chúng ta di chuyển turtle lùi lại 100 pixel với lệnh `t.backward(100)`, và sau đó sử dụng `t.up()` để nhấc bút lên và ngừng vẽ.

Sau đó, với lệnh `t.right(90)`, chúng ta xoay turtle sang phải 90 độ để nó hướng xuống, về phía dưới màn hình, và với lệnh `t.forward(20)`, chúng ta di chuyển về phía trước 20 pixel. Không có gì được vẽ ra vì chúng ta



đã dùng lệnh `up` ở dòng thứ ba. Chúng ta xoay turtle sang trái 90 độ với lệnh `t.left(90)` để nó hướng sang phải, rồi sau đó với lệnh `down`, chúng ta yêu cầu turtle đặt bút xuống và bắt đầu vẽ lại. Cuối cùng, chúng ta vẽ một đường thẳng về phía trước, song song với đường thẳng đầu tiên đã vẽ, với lệnh `t.forward(100)`. Hai đường thẳng song song mà chúng ta vẽ sẽ trông như sau:



Điều bạn đã học được

Trong chương này, bạn đã học cách sử dụng mô-đun turtle của Python. Chúng ta đã vẽ một số đường thẳng đơn giản, sử dụng các lệnh xoay trái và phải cùng với các lệnh di chuyển về phía trước và lùi lại. Bạn đã tìm hiểu cách ngừng vẽ của turtle bằng lệnh `up`, và bắt đầu vẽ lại với lệnh `down`. Bạn cũng đã khám phá cách turtle xoay theo độ.

Các câu đố lập trình

Hãy thử vẽ một số hình dưới đây bằng cách sử dụng turtle. Các đáp án có thể được tìm thấy tại <http://python-for-kids.com/>.

#1: Một Hình Chữ Nhật Tạo một canvas mới bằng cách sử dụng hàm `Pen` của mô-đun turtle và sau đó vẽ một hình chữ nhật.

#2: Một Hình Tam Giác Tạo một canvas khác, và lần này vẽ một hình tam giác. Hãy tham khảo lại biểu đồ của vòng tròn với các độ ("Di chuyển Turtle" trên trang 46) để nhắc lại hướng xoay của turtle sử dụng độ.

#3: Một Hình Hộp Không Có Góc Viết một chương trình để vẽ bốn đường thẳng như hình dưới đây (kích thước không quan trọng, chỉ cần đúng hình dạng):

