

Bí kíp luyện Lập trình nhập môn với Python

Vũ Quốc Hoàng
(Nguyễn Thị Bích Quyên soạn LaTeX phần I)

Bản thảo phần I, ngày 30/04/2020.
Hoan nghênh mọi ý kiến đóng góp!
(<https://github.com/vqhBook/python>)

Mục lục

1	Khởi động Python	1
1.1	Cài đặt Python	1
1.2	Làm quen Python	5
	Tóm tắt	7
	Bài tập	8
2	Tính toán đơn giản	11
2.1	Toán tử và biểu thức	11
2.2	Biến và lệnh gán	15
2.3	Giá trị luận lý và toán tử so sánh	18
2.4	Lỗi	19
2.5	Phong cách viết	22
	Tóm tắt	23
	Bài tập	23
3	Tính toán nâng cao	27
3.1	Hàm dựng sẵn	27
3.2	Module và thư viện	31
3.3	Từ khóa	34
3.4	Chế độ tương tác và phiên làm việc	35
3.5	Tra cứu	36
	Tóm tắt	38
	Bài tập	39
4	Bắt đầu lập trình	45
4.1	Chương trình và mã nguồn	45
4.2	Chế độ chương trình và người dùng	48
4.3	Dữ liệu	49
4.4	None	51
4.5	Chuỗi	52
4.6	Tiếng Việt và Unicode	54
	Tóm tắt	55
	Bài tập	56

5	Case study 1: Vẽ rùa	61
5.1	Khởi động con rùa	61
5.2	Bắt con rùa bò nhiều hơn	64
5.3	Bắt con rùa bò nhiều hơn nữa	65
5.4	Tô màu	68
5.5	Chế độ trình diễn	69
	Tóm tắt	70
	Bài tập	70
6	Phá vỡ đơn điệu với lệnh chọn	77
6.1	Luồng thực thi và tiến trình	77
6.2	Lệnh chọn và khối lệnh	78
6.3	if lồng	81
6.4	Toán tử điều kiện và toán tử luận lý	82
6.5	Cuộc bỏ trốn khỏi cửa sổ của con rùa	85
6.6	Bài toán, thuật toán và mã giả	86
6.7	Lệnh pass và chiến lược thiết kế chương trình từ trên xuống	89
	Tóm tắt	89
	Bài tập	90
7	Vượt qua hữu hạn với lệnh lặp	93
7.1	Lặp số lần xác định với lệnh for	93
7.2	Công việc được tham số hóa và biến lặp	95
7.3	Lặp linh hoạt với lệnh while	97
7.4	Vòng lặp lồng	99
7.5	Điều khiển chi tiết lệnh lặp	101
7.6	Lặp vô hạn	103
7.7	Duyệt chuỗi	105
	Tóm tắt	107
	Bài tập	107
8	Tự viết lấy hàm	113
8.1	Định nghĩa hàm/gọi hàm và tham số/đối số	113
8.2	Hàm tính toán, thủ tục và lệnh return	115
8.3	Cách truyền đối số và đối số mặc định	117
8.4	Hàm cũng là đối tượng	119
8.5	Biểu thức lambda và hàm vô danh	121
8.6	Lập trình hướng sự kiện	122
8.7	Lệnh biểu thức, hiệu ứng lề và docstring	124
8.8	Tự viết lấy module	126
	Tóm tắt	129
	Bài tập	130

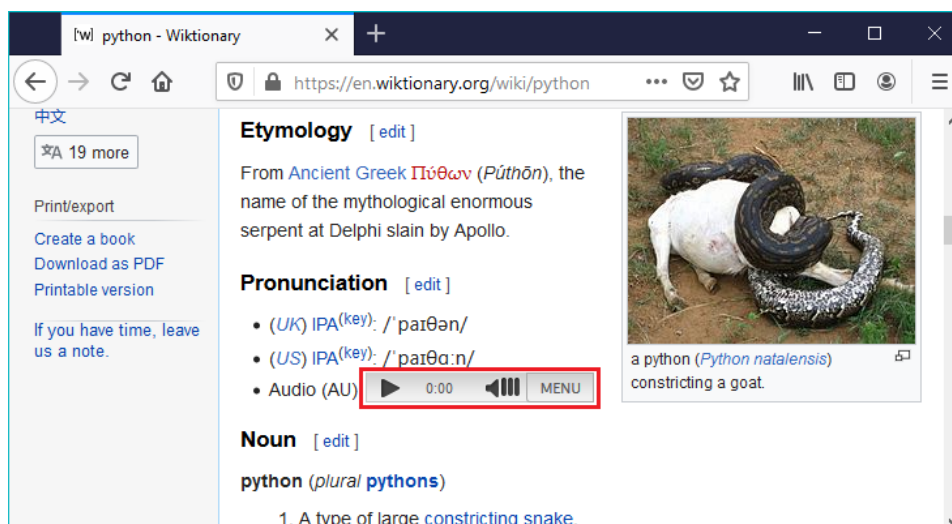
Bài 1

Khởi động Python

“Hành trình vạn dặm bắt đầu từ một bước chân!” Câu nói này của Lão Tử chính là để chỉ hành trình chinh phục Python của ta, bắt đầu từ bước quan trọng nhất, **cài đặt Python** (Python installation). Bài học đầu tiên này hướng dẫn bạn chuẩn bị môi trường làm việc với Python. Hơn nữa, bạn cũng sẽ chào hỏi và làm quen Python.

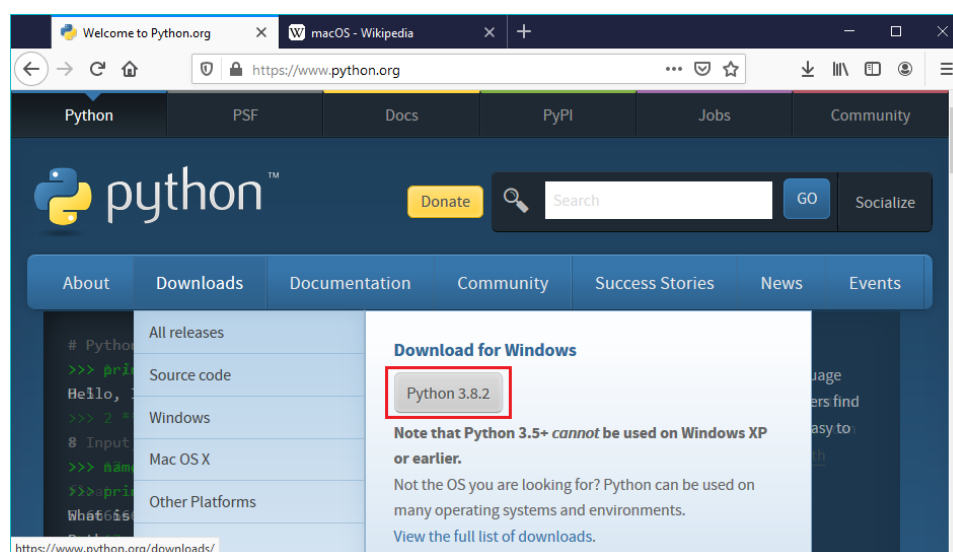
1.1 Cài đặt Python

Trước tiên, bạn cần đọc cho đúng từ Python. Từ Python có 2 cách đọc: kiểu Anh là “bai thon” còn kiểu Mỹ là “bai thon”. Bạn có thể tra từ python trong Wiktionary (<https://en.wiktionary.org/wiki/python>) và nghe phát âm như hình dưới. Cũng trong trang này, python là từ chỉ một loài rắn lớn, tuy nhiên, tên Python của ta lại có nguồn gốc khác.¹



¹Tương truyền, khi Guido van Rossum khai sinh Python cũng là lúc ông đang xem vở “Monty Python’s Flying Circus”, ông cần một cái tên “không đụng hàng” và kì bí, tên Python được chọn.

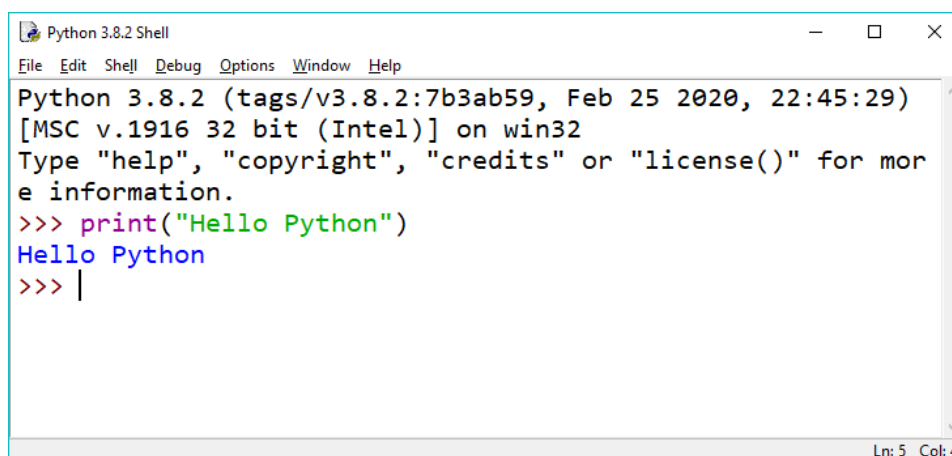
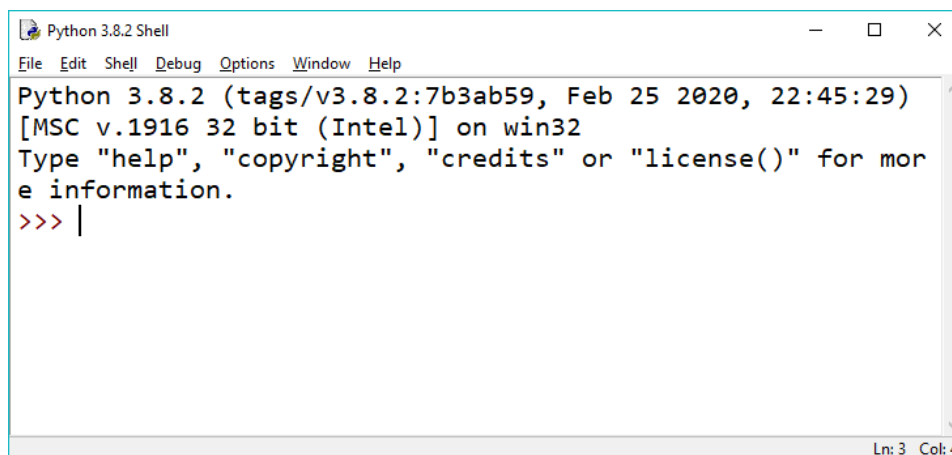
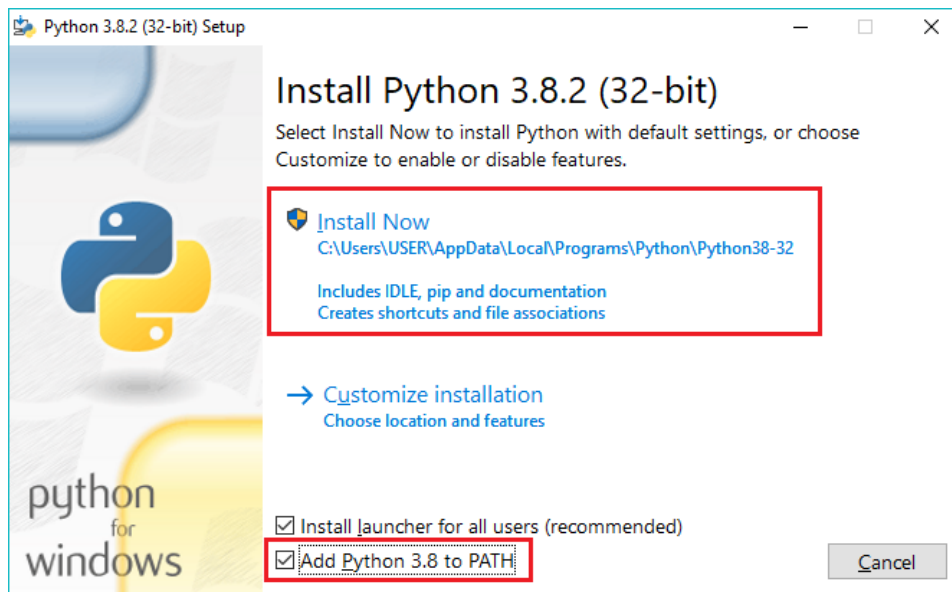
Để cài đặt Python, bạn vào trang chủ của Python ở địa chỉ <https://www.python.org/>, trong thẻ Downloads bạn sẽ thấy link tải file cài đặt Python. Nếu bạn dùng máy Windows thì nhấp nút “Python 3.8.2” như hình dưới. Lưu ý, con số 3.8.2 là số **phiên bản** (version) của Python, nó sẽ thay đổi về sau (càng ngày càng lớn hơn) thành 3.x.y gì đó (hoặc thậm chí là 4.x.y). Nếu máy bạn dùng hệ điều hành khác (Linux, Mac OS, ...) hoặc bạn muốn cài phiên bản Python khác thì có thể nhấp vào thẻ Downloads (<https://www.python.org/downloads/>). Nói chung, có hai nhóm phiên bản Python khác nhau đáng kể là Python 2 (2.x.y) và Python 3 (3.x.y). Các phiên bản trong cùng một nhóm thì không khác nhau nhiều lắm. Tài liệu này dùng phiên bản mới nhất, **Python 3**.



Phần sau đây hướng dẫn bạn cài Python cho Windows. Sau khi tải file cài đặt (file python-3.8.2.exe), bạn nhấp đúp vào file đã tải và nhấn nút Run sẽ thấy hộp thoại cài đặt như hình dưới. Bật lựa chọn “Add Python ... to PATH” và nhấp “Install Now”. Nhắc lại, bạn cần bật lựa chọn “Add Python ... to PATH” để thuận tiện cho việc dùng Python sau này. Sau đó đợi Python cài đặt (có thể phải nhấn nút Yes/OK gì đó). Khi cài đặt xong, Python sẽ thông báo là cài đặt thành công (setup was successful), nhấn nút Close và bạn có thể bắt đầu dùng Python trên máy của mình.

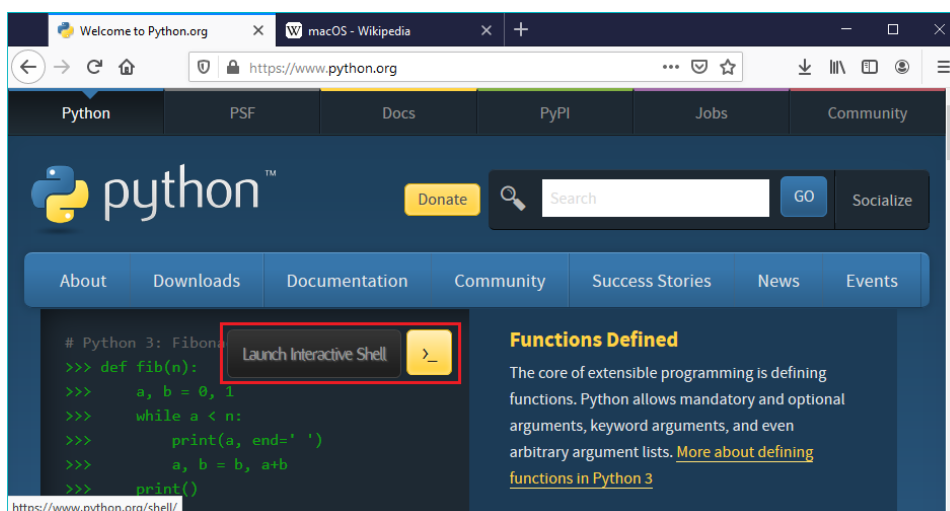
Để chạy Python, từ nút Windows trên thanh Taskbar, bạn nhấp chạy “IDLE (Python 3.8)”. IDLE hiện ra và sẵn sàng giúp Python nhận lệnh như hình dưới.² Cửa sổ IDLE này còn được gọi là **Python Shell** và kí hiệu >>> được gọi là **dấu đợi lệnh** (prompt). Bạn nhập yêu cầu, tức là ra lệnh, và Python **thực hiện/thực thi** (execute). Thử **lệnh** (statement) đầu tiên: `print("Hello Python")`, nghĩa là bạn gõ lệnh đó (gõ đúng y chang các kí tự, kể cả chữ hoa chữ thường) và nhấn phím Enter. Python sẽ thực hiện lệnh này với kết quả là dòng chữ Hello Python được xuất ra như hình dưới.

²IDLE là viết tắt của Integrated Development and Learning Environment.

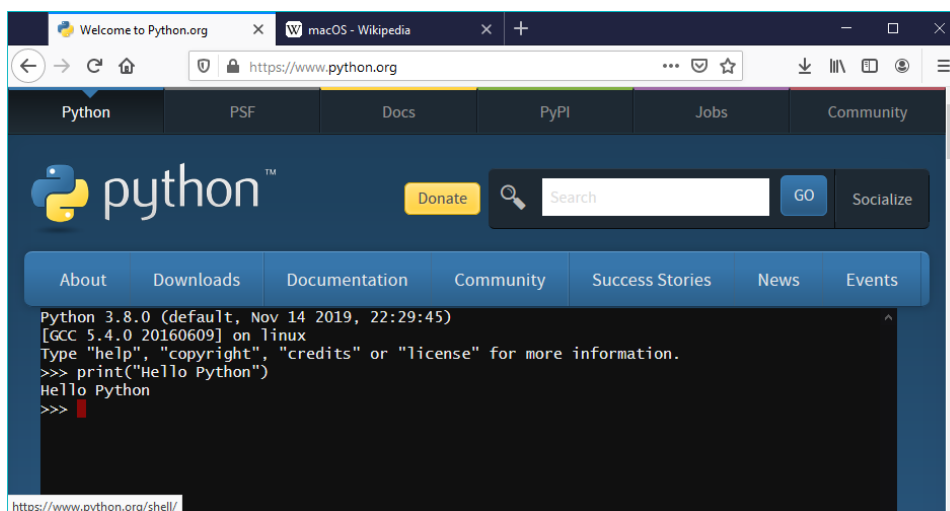


Chúc mừng!!! Bạn đã cài đặt thành công Python và thực hiện suôn sẻ lệnh đầu tiên trên IDLE. Thuật ngữ kỹ thuật gọi IDLE là một **môi trường phát triển tích hợp** (Integrated Development Environment, viết tắt IDE) cho Python, nghĩa là một tập các công cụ giúp bạn làm việc với Python. Có nhiều IDE như vậy, mà đa số là tiện lợi và nhiều chức năng hơn IDLE. Tuy nhiên, với sự đơn giản và sẵn dùng của mình, bạn *nên bắt đầu học Python bằng IDLE*.

Một lựa chọn khác, làm việc với Python mà không cần cài đặt, là dùng **Python trực tuyến** (online Python). Với máy tính kết nối mạng và một trình duyệt Web, bạn có thể thử bắt đầu với Python trực tuyến được cung cấp từ chính trang chủ của Python. Vào trang <https://www.python.org/> và nhấp nút “Launch Interactive Shell” như hình dưới.



Trong Python Shell (cửa sổ đen thui), bạn gõ yêu cầu và Python thực hiện như trên IDLE.



Như vậy, có nhiều cách dùng Python, quan trọng hơn vẫn là những yêu cầu ta đưa cho nó. Các yêu cầu đó (cùng với kết quả mà Python thực hiện) được mô tả như sau trong tài liệu này:

```
1 >>> print("Hello Python")
Hello Python
```

Lưu ý, các số nhỏ ngoài lề chỉ số dòng của lệnh, bạn không gõ chúng cũng như không gõ dấu nhắc lệnh và các kết quả mà Python xuất ra.

1.2 Làm quen Python

Bạn đã biết cách dùng Python (cài đặt và dùng IDLE hoặc dùng Python online). Bây giờ ta hãy làm quen Python chút nhé. Trước hết, Python không phải là thần thánh! Với “Hello Python” ở trên, tôi không mong đợi Python trả lời là “Hello Hoàng”. Đơn giản, nó không biết tôi là ai. Thậm chí, ta cũng đừng mong Python trả lời là “Hi you”. Python không trò chuyện hay tâm tình với ta. Python không phải là bạn bè!

Python là đây đó! Ta bảo gì, Python làm nấy. Ta ra lệnh, Python thực hiện. *Python là một công cụ hay một động cơ thực thi!* Điều tuyệt vời là Python không nề hà, không ngại khó, lại rất mạnh mẽ với trí nhớ vô biên và khả năng tính toán cực nhanh. Điều khó khăn là ta phải biết cách ra lệnh cho Python. Python không hiểu Tiếng Việt, hiểu “sơ sơ” Tiếng Anh. Đúng hơn, Python chỉ hiểu tiếng của nó. Tiếng gì? Tiếng Python! Chữ Python, như vậy, được dùng với 2 nghĩa là **động cơ Python** (Python engine) và **ngôn ngữ Python** (Python language).³

Học Python là học cách dùng ngôn ngữ Python để yêu cầu động cơ Python thực hiện các công việc mình mong đợi. Ta đã dùng **lệnh xuất** (output statement) `print("Hello Python")` viết theo ngôn ngữ Python để yêu cầu động cơ Python xuất ra dòng chữ Hello Python. Cũng yêu cầu này, nếu bạn nói một cách thắm thiết hơn là Xuất ra dòng chữ "Hello Python" cái coi! thì Python chẳng hiểu. Thử há:

```
1 >>> Xuất ra dòng chữ "Hello Python" cái coi.
SyntaxError: invalid syntax
```

Như bạn thấy, Python trả lời là `SyntaxError: invalid syntax` mà ý đại khái là “nói gì ...éo hiểu”. Đây không phải lỗi của Python mà là lỗi của ta vì đã không nói ngôn ngữ của Python. Rõ ràng, *khi gặp thông báo này (SyntaxError), bạn cần kiểm tra và gõ lại lệnh cho đúng để Python có thể hiểu và thực thi.*

Bạn có thể cho rằng ngôn ngữ này quá cứng nhắc, thiếu cảm xúc, kiểu như ngôn ngữ của máy móc hay robot. Đúng hơn, ngôn ngữ này rất đơn giản, ngắn gọn, không gây mơ hồ hay nhầm lẫn như ngôn ngữ của ta (Tiếng Việt, Tiếng Anh, ...).

³Thuật ngữ kĩ thuật là **trình thông dịch Python** (Python interpreter) và **ngôn ngữ lập trình Python** (Python programming language) mà ta sẽ rõ hơn sau.

Với mục đích mô tả các yêu cầu, nó hay và tốt hơn ngôn ngữ của ta nhiều. Bạn coi lại đi, câu “Xuất ra dòng chữ "Hello Python" cái coi!” nó mới dài dòng làm sao. Chỉ có 2 thứ trong câu đó thôi. Làm gì? Xuất. Xuất gì? “Hello Python”. Đẹp ba cái thứ dư thừa và vớ vẩn kia (trong việc mô tả yêu cầu) ta còn đúng lại là: `print("Hello Python")`.⁴

Thế tôi muốn xuất ra tên mình thay vì Python thì làm sao?

```
1 >>> print("Chào Hoàng")
Chào Hoàng
```

Chế lại chút thôi.⁵ Nếu nói việc viết các lệnh để Python thực hiện các công việc nào đó là **lập trình** (programming) thì lập trình chỉ đơn giản vậy thôi!

Thế tôi muốn xuất ra tên mình trang trọng trong một cái khung thì sao?

```
1 >>> print("====="); print("| Chào Hoàng |");
↪ print("=====")
=====
| Chào Hoàng |
=====
```

Ta bảo Python xuất ra 3 dòng chữ bằng 3 lệnh. **Dấu chấm phẩy (;)** được dùng để phân cách các lệnh. Sáng tạo đó. Lập trình là vậy đó! Lưu ý là bạn gõ cả 3 lệnh trên một dòng như số dòng cho thấy. Kí hiệu ↪ cho biết việc xuống dòng trong khung hình trên chỉ là để tiện trình bày, bạn phải gõ trên một dòng trong IDLE.

Thậm chí, nếu hiểu rõ Python hơn, công việc trên có thể viết gọn lại là:

```
1 >>> print("=====\n| Chào Hoàng |\n=====")
=====
| Chào Hoàng |
=====
```

Ta chỉ dùng 1 lệnh, nhưng phải biết là `print` không chỉ yêu cầu xuất ra một dòng chữ mà là một văn bản, hay **chuỗi** (string) như cách “dân chuyên nghiệp” thường gọi, mà dòng chữ chỉ là một trường hợp. Ở đây, văn bản của ta gồm 3 dòng với kí hiệu đặc biệt `\n` báo hiệu cho việc xuống dòng (xem Bài tập 1.4). Lập trình Python là vậy đó!

Có cách viết nào khác không? Có luôn:

```
1 >>> print("=====", "| Chào Hoàng |",
↪ "=====", sep="\n")
=====
| Chào Hoàng |
=====
```

⁴Thậm chí trong Python 2 bạn chỉ viết `print "Hello Python"`.

⁵Hy vọng bạn thấy chỗ tôi đã chế, tức là đã sửa. Bạn cũng cần bộ gõ Tiếng Việt (như UniKey) để có thể gõ các kí tự có dấu Tiếng Việt.

Ở đây, ta cần biết là `print` có thể nhận nhiều chuỗi cần xuất và sẽ xuất ra tất cả các chuỗi đó, phân cách bởi chuỗi do `sep` qui định (tức là chuỗi sau `sep=`). Lập trình Python là vậy đó!

Còn cách viết nào nữa không? Còn luôn:

```
1 >>> print("""
2 =====
3 | Chào Hoàng |
4 =====
5 """)

=====
| Chào Hoàng |
=====
```

Dấu **3 nháy** (triple-quote) `"""` (3 ký tự `"` viết liền) giúp ta mô tả một chuỗi mà ta có thể gõ trải dài trên nhiều dòng (xem Bài tập 1.2), còn dấu **nháy kép** (double quote) `"` chỉ giúp ta mô tả chuỗi gõ trên 1 dòng (xem Bài tập 1.1).

Còn nữa không? Chắc còn á, bạn tự khám phá đi! Dĩ nhiên, việc xuất ra đôi ba dòng chữ thì không có gì ghê gớm hay hấp dẫn. *Python có thể làm rất rất rất nhiều thứ nếu ta biết cách bảo nó*. Nếu bạn muốn làm những việc khó hơn, có ích hơn và thú vị hơn thì tiếp tục nào!

Các minh họa trên cũng cho thấy cách bạn học lập trình (và lập trình Python): *bắt chước, thử nghiệm và sáng tạo*. Mới đầu nên bắt chước, sau đó thử nghiệm nhiều, và càng về sau càng phải sáng tạo. Điều quan trọng, bạn *học lập trình bằng cách luyện tập và trải nghiệm lập trình*: bạn không chỉ đọc, nghĩ mà phải bắt tay vào gõ, chạy, thử (nghiệm), (khám) phá, (sáng) tạo, ...

Tóm tắt

- Python IDE là tập các công cụ giúp ta làm việc với Python và IDLE là cái đơn giản mà ta có thể dùng để học Python. Ngoài ra, ta cũng có thể dùng Python trực tuyến mà không cần cài đặt.
- Python 3 là phiên bản Python mới nhất, đang được dùng nhiều, khác với Python 2 là phiên bản đã lỗi thời.
- Lập trình Python là viết các lệnh theo ngôn ngữ Python để yêu cầu động cơ Python thực thi các công việc mong đợi nào đó.
- Cách học lập trình (Python) là bắt chước, thử nghiệm và sáng tạo qua việc luyện tập và trải nghiệm lập trình (Python).
- Để yêu cầu Python xuất ra một chuỗi, ta có thể dùng lệnh xuất `print("... chuỗi cần xuất...")` và các biến thể của nó.
- Để yêu cầu Python thực hiện “một lượt” nhiều lệnh, ta có thể dùng dấu chấm phẩy (;) phân cách giữa các lệnh.

Bài tập

1.1 Chuỗi 1 dòng. Python cho phép ta mô tả chuỗi ngắn bằng cặp nháy kép "...", hoặc cặp **nháy đơn** (single quote) '...'. Mặc dù, ta *nên dùng dấu nháy kép* như nhiều ngôn ngữ lập trình hay dùng, nhưng trong trường hợp bản thân chuỗi có dấu nháy kép thì ta nên dùng cặp nháy đơn để “bọc” chúng như sau:

```
1 >>> print("Ta nên dùng cặp nháy kép(...) cho chuỗi")
  SyntaxError: invalid syntax
2 >>> print('Ta nên dùng cặp nháy kép(...) cho chuỗi')
  Ta nên dùng cặp nháy kép(...) cho chuỗi
3 >>> print("Python is an easy to learn, "
4       'powerful programming language.')
  Python is an easy to learn, powerful programming language.
```

Mình họa cũng cho thấy cách viết các chuỗi dài nhưng kết xuất chỉ trên 1 dòng bằng cách viết nhiều chuỗi kề nhau.

Viết lệnh để Python xuất ra văn bản sau (có dấu nháy kép):

```
Python is the "most powerful language you can still read".
- Paul Dubois
```

1.2 Chuỗi nhiều dòng. Python cho phép mô tả chuỗi dài gõ trên nhiều dòng bằng cặp dấu 3-nháy ("""" hoặc ''') như minh họa sau:

```
1 >>> print('''\
2 Python is an:
3   - interpreted
4   - high-level
5   - general-purpose
6 programming language, \
7 created by Guido van Rossum.''' )
  Python is an:
    - interpreted
    - high-level
    - general-purpose
  programming language, created by Guido van Rossum.
```

Nếu không muốn xuống dòng trong chuỗi, ta có thể dùng dấu \ như minh họa.

Viết lệnh để Python xuất ra:

```
"Dưới cầu nước chảy trong veo
Bên cầu tơ liễu bóng chiều thướt tha."
Truyện Kiều - Nguyễn Du.
```

1.3 Thiền trong Python. Viết lệnh để Python xuất ra:⁶

⁶Đây là những khẩu quyết của “Thiền trong Python” (Zen of Python, https://en.wikipedia.org/wiki/Zen_of_Python). Python thuộc lòng nó và bạn có thể bắt Python đọc ra bằng lệnh: `import this`. Bạn không cần phải thuộc nó để có thể lập trình Python đâu nhé!

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
...  
Tốt gỗ hơn tốt nước sơn!!!
```

1.4 Dãy kí tự thoát và chuỗi thô. Python dùng **dãy kí tự thoát** (escape sequence) để mô tả các kí hiệu đặc biệt trong chuỗi. Chẳng hạn, `\n` mô tả xuống dòng như ta đã biết trong bài học, `\"` mô tả kí tự `"`, `\t` mô tả kí tự Tab (cách nhiều khoảng trắng), ... như minh họa sau:

```
1 >>> print("Ta nên dùng cặp nháy kép(\"...\") cho chuỗi")  
Ta nên dùng cặp nháy kép("...") cho chuỗi  
2 >>> print("Python is an:\n\t- interpreted\n\t-  
  ↳ high-level\n\t- general-purpose\n\t- programming language,  
  ↳ created by Guido van Rossum.")  
Python is an:  
    - interpreted  
    - high-level  
    - general-purpose  
programming language, created by Guido van Rossum.
```

Trường hợp “tình cờ” có dãy kí tự thoát trong chuỗi mà ta không muốn Python xem nó như là kí hiệu đặc biệt thì ta có thể dùng **chuỗi thô** (raw string) bằng cách đặt kí tự `r` (hoặc `R`) ngay trước chuỗi. Khi gặp chuỗi thô, Python dùng nó “y xì” như cách nó được viết mà không phân tích thêm (và do đó bỏ “tác dụng” của các dãy kí tự thoát) như minh họa sau:

```
1 >>> print("\n is the escape sequence of newline")  
  
 is the escape sequence of newline  
2 >>> print(r"\n is the escape sequence of newline")  
\n is the escape sequence of newline
```

Ôi, chỉ là mô tả cái chuỗi thôi mà sao rắc rối quá vậy! Bài tập này được cố ý đưa vào ngay từ bài học đầu để bạn thấy rằng việc lập trình ... “cũng” rắc rối. Python đã cố gắng loại bỏ nhiều vấn đề phiền toái cho ta nhưng ta vẫn không thể tránh khỏi vài chi tiết cụ thể, vụn vặt. Đó là bản chất của lập trình!

Viết lệnh để Python xuất ra bảng sau (dùng các kí hiệu `|`, `-`, ... “mô phỏng” cái bảng như cái khung tên trong bài học):

Dãy kí tự thoát	Ý nghĩa
\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\n	ASCII Linefeed (LF)
\t	ASCII Horizontal Tab (TAB)

1.5 Chạy Python từ cửa sổ lệnh. Trên Windows bạn có thể chạy Python từ cửa sổ lệnh **Command Prompt**. Trước hết, mở Command Prompt (có thể bằng cách gõ cmd trong nút Search trên thanh Taskbar rồi nhấn Enter), sau đó gõ python và nhấn Enter để chạy Python như hình dưới. Sau khi Python được chạy, ta có thể dùng Python như trong IDLE.

```

Microsoft Windows [Version 10.0.17134.228]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\USER>python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.
Type "help", "copyright", "credits" or "license" for more informa
>>> print("Hello Python")
Hello Python
>>> quit()

C:\Users\USER>

```

Lưu ý, nếu Command Prompt không chạy được Python (thông báo đại khái là “python’ is not recognized ...”) thì có thể bạn đã quên bật lựa chọn “Add Python ... to PATH” khi cài đặt Python. Bạn chỉ cần cài lại Python với lựa chọn “Add Python ... to PATH” là được. Sau khi dùng Python xong, bạn có thể gõ lệnh quit() để kết thúc Python (hoặc đóng cửa sổ Command Prompt).

Chạy lại các ví dụ trên bằng Command Prompt. Lưu ý là cửa sổ Command Prompt không hỗ trợ đồ họa, font chữ, Tiếng Việt, ... tốt như IDLE.

1.6 Đọc IDLE, thực hiện hay tùy chỉnh các chức năng đơn giản sau:

- Cắt, sao chép và dán lệnh. (Gợi ý: có thể dùng IDLE như các chương trình thao tác văn bản khác, có thể bôi chọn văn bản bằng chuột, nhấn Ctrl+C để Copy và Ctrl+V để Paste, ...)
- Chọn font chữ và tăng/giảm kích thước font chữ của IDLE. (Gợi ý: vào mục menu Options → Configure IDLE, thẻ Font/Tabs trong hộp thoại Setting.)
- Chọn màu sắc và theme cho IDLE. (Gợi ý: thẻ Highlights trong Settings.)
- Lưu lại nội dung cửa sổ IDLE (các lệnh cùng với các kết xuất) bằng File → Save (phím tắt Ctrl+S), trong hộp thoại “Save As” chọn “Save as type” là “Text files”, gõ tên file trong “File name” và nhấn nút Save.

Bài 2

Tính toán đơn giản

Một trong những công việc căn bản và quan trọng của con người là **tính toán** (computation). Thuật ngữ này có nghĩa rất rộng, ám chỉ những quá trình gồm các bước thao tác trên các đối tượng số hoặc không phải số. Có thể nói, Toán học, khoa học, kỹ thuật ra đời và phát triển từ nhu cầu tính toán. **Máy tính** (computer) được xem là một trong những phát minh lớn nhất giúp thực hiện tự động việc tính toán và thuật ngữ **điện toán** (computing) ám chỉ việc tính toán bằng máy tính. Ta sẽ thấy rằng Python là một công cụ mạnh mẽ và tiện dụng của điện toán. Bài này hướng dẫn dùng Python làm công cụ **tính toán số học** (arithmetic calculation) đơn giản, tức là dùng Python như một **máy tính số học** (calculator).¹

2.1 Toán tử và biểu thức

Để bắt đầu, ta thử làm bài toán sau:²

Tính nhanh: $15.64 + 25.100 + 36.15 + 60.100$

Dấu chấm (.) kí hiệu cho **phép nhân** (multiplication) và dấu cộng (+) kí hiệu cho **phép cộng** (addition).

Ta dùng Python để tính nhanh. Mở Python, gõ yêu cầu tính toán và nhận kết quả như sau:

```
1 >>> 15*64 + 25*100 + 36*15 + 60*100
10000
```

Như vậy, kết quả là 10000. Nếu không kể thời gian khởi động Python, thời gian ta gõ yêu cầu là khoảng 10 giây, thời gian Python tính là khoảng “một phần triệu nháy mắt”, thì tổng thời gian tính là khoảng 10 giây. Quá nhanh!

Có thể bạn không hài lòng về cách làm này, nhưng để lại đó đã, trước hết, ta có một loại yêu cầu rất quan trọng mà ta có thể bảo Python làm, đó là tính toán số học.

¹Ta hay gọi là máy tính bỏ túi, mặc dù ít khi ta bỏ túi.)

²SGK Toán 8 Tập 1.

Ta mô tả yêu cầu này bằng một **biểu thức** (expression) gồm các **số** (number) và các **toán tử** (operator) là kí hiệu mô tả các **phép toán** (operation). Việc tính toán còn được gọi là **lượng giá** (evaluation) và kết quả tính ra còn được gọi là **giá trị** (value) của biểu thức. Chẳng hạn, biểu thức trên có giá trị là 10000. Lưu ý, Python dùng toán tử $*$ kí hiệu cho phép nhân chứ không dùng dấu chấm (.) như ta hay dùng.

Trở lại, bạn có thể cãi rằng, cách tính nhanh phải là (dấu \times kí hiệu cho phép nhân):

$$\begin{aligned} &15 \times 64 + 25 \times 100 + 36 \times 15 + 60 \times 100 \\ &= 15 \times (64 + 36) + 25 \times 100 + 60 \times 100 \\ &= 15 \times 100 + 25 \times 100 + 60 \times 100 \\ &= (15 + 25 + 60) \times 100 \\ &= 100 \times 100 \\ &= 10000 \end{aligned}$$

Quá nguy hiểm ... nhưng ... không nhanh! Bạn nghĩ rằng sau một thời gian dài nghỉ hè, bạn có thể dễ dàng làm như vậy không? Giả sử một vài số trong đó không còn “đẹp” nữa, chẳng hạn, thay vì 15×64 là 16×64 , thì bạn còn làm được như vậy không?

Tương tự, bạn có thể tính nhanh biểu thức sau không?³

$$74^2 + 24^2 - 48.74$$

Làm như sau:

```
1 >>> 74**2 + 24**2 - 48*74
2500
```

Ta cũng chỉ cần đưa biểu thức để nhờ Python lượng giá. Lưu ý, toán tử $**$ (2 kí tự $*$ viết liền) kí hiệu cho **phép mũ** (exponentiation) hay phép **lũy thừa** (power). Dĩ nhiên, để mô tả 74^2 ta cũng có thể dùng biểu thức $74*74$. Toán tử trừ ($-$) kí hiệu cho **phép trừ** (subtraction) như thông thường.

Bạn có thể tính nhanh theo cách “vi diệu” gì đó không? Tôi cũng có thể làm được, cũng không khó mấy, nhưng không cần thiết. Toán học đôi khi bị lợi dụng; mẹo mực thường bị nhầm lẫn với trí tuệ. Dĩ nhiên, ranh giới giữa mảnh lời và khéo léo, mẹo vặt và trí khôn thường không rõ ràng. Thời đại tính tay đã qua lâu, thời đại của calculator cũng sắp qua, *giờ là thời đại của computer và của Python!*⁴

Các biểu thức trên làm việc với **số nguyên** (integer). Bây giờ ta qua **số thực** (real number) và đó là lúc mà **dấu chấm thập phân** (decimal point) (.) xuất hiện. Một hình vuông có cạnh dài 5 cm thì diện tích là 25 cm^2 (chính là 5^2 , mà viết trong Python là $5**2$). Vậy hình vuông có diện tích 30 cm^2 thì cạnh dài bao nhiêu? Lớn hơn 5 (vì $5^2 = 25 < 30$) nhưng nhỏ hơn 6 (vì $6^2 = 36 > 30$), tức là khoảng “5 chấm mấy” đó. Cụ thể là bao nhiêu? Toán ghi là $\sqrt{30}$ để chỉ con số thực mà bình phương lên được 30. Nhưng là bao nhiêu? Python cho biết:

³Cũng trong SGK Toán 8 Tập 1.

⁴Nhân tiện, những cuộc thi trên Casio gì đó nên bỏ đi mà thay bằng Python!


```
1 >>> 30 ** (1/2)
5.477225575051661
```

Nhờ Toán ta biết $\sqrt{30} = 30^{\frac{1}{2}}$ vì $(30^{\frac{1}{2}})^2 = 30^{\frac{1}{2} \times 2} = 30^1 = 30$. Dĩ nhiên, vì $\frac{1}{2} = 0.5$ nên ta cũng có thể viết là $30 ** 0.5$. Như vậy, Python dùng toán tử $/$ kí hiệu cho **phép chia** (division). Đặc biệt, Python dùng dấu chấm (.) để phân cách **phần nguyên** (integer part) và **phần lẻ** (fractional part) trong **biểu diễn thập phân** (decimal representation) của số thực chứ không dùng dấu phẩy (,) như Việt Nam ta hay dùng.

Lưu ý, cặp ngoặc tròn ở trên là rất quan trọng, nếu không có nó, ta sẽ được kết quả khác (không đúng mong đợi) như sau:

```
1 >>> 30 ** 1/2
15.0
```

Tại sao? Đó là vì Python ưu tiên thực hiện phép mũ (**) trước phép chia (/) (do đó biểu thức trên được Python hiểu là $\frac{30^1}{2}$), ta còn nói phép mũ có **độ ưu tiên** (precedence) cao hơn phép chia. Độ ưu tiên của các toán tử giúp Python xác định rõ ràng thứ tự thực hiện các phép toán: *toán tử có độ ưu tiên cao hơn sẽ được thực hiện trước*. Bạn đã biết một phần của qui tắc ưu tiên này qua câu cửa miệng “nhân chia trước, cộng trừ sau”. Dĩ nhiên, Python cần tính giá trị trong các cặp ngoặc tròn trước như thông thường.

Một điều lưu ý quan trọng nữa là thứ tự mà Python thực hiện trên các toán tử có cùng độ ưu tiên. Theo bạn, Python sẽ lượng giá biểu thức $4 - 3 - 2$ ra giá trị mấy? Là -1, vì trong 2 toán tử trừ (-), toán tử đầu được thực hiện trước, tức là biểu thức trên được Python hiểu là $(4 - 3) - 2$ chứ không phải là $4 - (3 - 2)$. Ta nói toán tử trừ (-) có tính **kết hợp trái** (left-associative). Ngược lại, theo bạn, Python sẽ lượng giá biểu thức $4 ** 3 ** 2$ ra giá trị mấy? Là 262144, vì trong 2 toán tử mũ (**), toán tử sau được thực hiện trước, tức là biểu thức trên được Python hiểu là $4 ** (3 ** 2)$ chứ không phải là $(4 ** 3) ** 2$. Ta nói toán tử mũ (**) có tính **kết hợp phải** (right-associative).

Sẵn nói về các khái niệm liên quan đến toán tử,⁵ bạn cần biết rằng toán tử trừ (-) được gọi là **toán tử 2 ngôi** (binary operator), tức có **số ngôi** (arity) là 2, vì nó cần 2 **toán hạng** (operand). Ta viết nó dưới dạng $x - y$ và Python lượng giá bằng cách tính giá trị của toán hạng x , toán hạng y , rồi thực hiện phép trừ để được giá trị cuối cùng của biểu thức. Các toán tử +, *, /, ** cũng là các toán tử 2 ngôi. **Toán tử 1 ngôi** (unary operator) hay gặp là toán tử đối, cũng kí hiệu là -,⁶ nhưng được viết ở dạng $-x$, chỉ có 1 toán hạng.⁷ Như thông thường, toán tử này giúp thực hiện

⁵Tôi sẽ cố gắng dùng ít thuật ngữ và khái niệm nhưng bạn cũng nên nắm vững những thuật ngữ và khái niệm cơ bản nhất để có thể nói chuyện với “người trong nghề”, để đọc tài liệu và có nền tảng vững chắc để tiến xa hơn.

⁶Do đó, toán tử này còn được gọi là **toán tử trừ 1 ngôi** (unary minus operator).

⁷Lưu ý, dấu - cũng được dùng cho toán tử trừ 2 ngôi. Hiện tượng “**lạm dụng kí hiệu**”, tức là dùng cùng kí hiệu cho các mục đích khác nhau, xuất hiện nhiều trong ngôn ngữ tự nhiên, trong Toán và cả trong Python. Ngữ cảnh sẽ giúp ta và Python xác định rõ mục đích của kí hiệu đó.

phép đối (negation), tức là “đổi dấu” giá trị số. Lưu ý, toán tử đối có độ ưu tiên cao hơn toán tử trừ nhưng thấp hơn toán tử mũ. Chẳng hạn, biểu thức $-1 - 2$ được Python hiểu là $(-1) - 2$, có giá trị -3 còn biểu thức $-1 ** 2$ được Python hiểu là $-(1 ** 2)$, có giá trị -1.

Sau đây là bảng tóm tắt các toán tử hay gặp của Python. Các toán tử này được gọi chung là các **toán tử số học** (arithmetic operator) vì nó giúp ta thực hiện các tính toán trên số như thông thường. Ta sẽ gặp nhiều toán tử Python nữa.

Độ ưu tiên (Precedence)	Toán tử (Operator)	Số ngôi (Arity)	Tính kết hợp (Associativity)
1 (cao nhất)	mũ (**)	2	Phải
2	đối (-)	1	(Phải)
3	nhân (*), chia (/) chia nguyên (//) chia lấy dư (%)	2	Trái
4	cộng (+), trừ (-)	2	Trái

Đối với số nguyên, ngoài phép chia thông thường $5 / 2$ được 2.5 thì **phép chia nguyên** (floor division) chỉ giữ lại phần nguyên, $5 // 2$ được 2 và **phép chia lấy dư** (modulo) giữ lại **phần dư** (remainder), $5 \% 2$ được 1, cũng thường được dùng. Chẳng hạn, số nguyên chẵn là số chia 2 dư 0. Bạn cũng thử các minh họa sau để nắm rõ hơn về 2 phép chia này:⁸

```
1 >>> 123 // 100; 123 % 100 // 10; 123 % 10
1
2
3
```

Với các biểu thức “hỗn hợp” chứa cả giá trị nguyên lẫn thực, Python “chuyển” các giá trị nguyên thành thực rồi lượng giá và cho kết quả là giá trị thực như minh họa sau:⁹

```
1 >>> print(1.0 * 2, 2/2 * 2)
2.0 2.0
2 >>> print(2 ** 100, 2.0 ** 100)
1267650600228229401496703205376 1.2676506002282294e+30
```

Lưu ý là phép chia thường / cho kết quả thực và Python làm việc với số nguyên “cực tốt” (chính xác tuyệt đối với số lớn bất kỳ) nhưng “khá tệ” với số thực (không

⁸Tôi viết các lệnh trên một dòng để “tiết kiệm giấy”! Bạn nên thử viết riêng mỗi lệnh và quan sát kết quả. Không nên “bắt chước quá máy móc”. Hơn nữa, các toán tử này cũng dùng được trên số thực. Bạn thử nghiệm xem sao. “*Bắt chước và thử nghiệm*” nhiều hơn từ giờ!

⁹Không chỉ xuất chuỗi, print cũng có thể được dùng để xuất số (thực ra là mọi “thứ”). Và không chỉ xuất ra 1 mà có thể nhiều “thứ” phân cách bằng dấu phẩy như ta đã biết ở Phần 1.2 với chuỗi. Bạn có thể không dùng print vì Python tự động xuất kết quả nhưng đây là một “thủ đoạn” khác tôi dùng để tiết kiệm giấy! Như đã nói, bạn không nên bắt chước quá máy móc.

chính xác trong “nhiều” trường hợp). Cách viết $1.2676506002282294e+30$ trong kết xuất ở trên mô tả số thực $1.2676506002282294 \times 10^{30}$, là **kí hiệu khoa học** (scientific notation) hay dùng để mô tả các số thực rất lớn (hoặc rất nhỏ). Kết quả này cho thấy 2^{100} là con số nguyên có 31 chữ số thập phân mà nếu dùng số nguyên thì Python cho kết quả chính xác từng chữ số, còn nếu dùng số thực thì chỉ chính xác khoảng 17 chữ số đầu, còn gọi là **chữ số có nghĩa** (significant digit), mà thôi.¹⁰

2.2 Biến và lệnh gán

Bây giờ ta thử làm bài toán Lớp 8 khác như sau:

Đặt $A = 2 - \sqrt{3}$ và $B = 2 + \sqrt{3}$. Chứng minh rằng A, B là hai số nghịch đảo nhau.

Làm Toán nhé. Ta có:

$$A \times B = (2 - \sqrt{3}) \times (2 + \sqrt{3}) = 2^2 - (\sqrt{3})^2 = 4 - 3 = 1$$

Vậy A, B nghịch đảo nhau.

Làm Python nhé. Ta có:

```
1 >>> A = 2 - 3**0.5; B = 2 + 3**0.5
2 >>> A * B
1.0000000000000004
```

Lưu ý, tích của A, B không “hoàn hảo” là 1, phần lẻ rất nhỏ nhưng khác 0. Tôi đã nói rằng cách làm bằng Tin (dùng Python) là tốt hơn Toán (biến đổi với các mẹo mực) trong bài toán tính nhanh trước. Trong bài toán này ta lại thấy cách làm bằng Toán (dùng hằng đẳng thức và tính chất của căn) lại tốt hơn Tin (dùng Python). Vậy rốt cuộc là sao? *Mỗi cách đều có ưu điểm và khuyết điểm, cách tốt nhất là cách phù hợp với tình huống cần giải quyết.* Toán là lý thuyết; Tin là thực hành. Toán là tưởng tượng; Tin là thực tế... Và ... ta cần cả hai! Điều quan trọng cần học ở đây là: tư duy **Tin học** (Informatics), tư duy **giải quyết vấn đề** (problem solving) bằng máy tính, tư duy **khoa học máy tính** (computer science), tư duy **lập trình máy tính** (computer programming) hay *tư duy Python là tư duy thực tế, cụ thể, rõ ràng và chi tiết!*

Điều quan trọng nữa mà cách làm Python trên cho thấy là ta có thể đặt các **kí hiệu** (notation) như A, B ở trên trong Python. Thực tế, việc dùng kí hiệu hay **đặt tên** (naming) là phổ biến và hầu như không thể tránh khỏi trong mọi hoạt động giao tiếp. Trong Python, ta có thể kí hiệu hay đặt tên cho giá trị của một biểu thức bằng **lệnh gán** (assignment statement) như sau:

`<Name> = <Expr>`

Khi thực thi lệnh này, Python lượng giá biểu thức `<Expr>` để được giá trị `<Value>`, tạo **tên** (name)¹¹ `<Name>` và đặt tên đó kí hiệu cho, còn gọi là **tham chiếu** (refer)

¹⁰Nếu việc đếm số lượng chữ số của số 2^{100} gây khó dễ cho bạn thì bạn có thể dùng lệnh `len(str(2**100))` để Python “đếm giùm”. Bạn sẽ rõ lệnh này sau.

¹¹Thuật ngữ kĩ thuật là **định danh** (identifier).

hay **kết buộc** (bind) hay **trỏ** (point) đến, <Value>. Ta cũng nói <Value> được **gán** (assign) cho <Name> và <Name> được **định nghĩa** (define) là <Value>.

Sau khi <Name> được định nghĩa thì ta có thể dùng nó: khi <Name> xuất hiện trong các biểu thức thì giá trị mà nó tham chiếu đến (<Value>) sẽ được Python dùng. Lưu ý, như vậy, dấu bằng (=) được Python dùng với ý nghĩa là “được gán bằng” hay “được định nghĩa là” hay “kí hiệu cho” chứ không mang nghĩa “so sánh bằng”.¹²

Ta sẽ thấy rằng Python cho phép đặt tên cho hầu như tất cả mọi thứ. Trường hợp tên kí hiệu cho một giá trị thì được gọi là (tên) **biến** (variable). Các calculator “xịn” cũng thường cung cấp khả năng này qua các nút nhớ như X, Y hay A, B, C, ... nhưng khá hạn chế. Với Python, ta có thể dùng bao nhiêu tên và tên dài thế nào cũng được, miễn là ta đặt tên đúng qui tắc. Qui tắc đặt tên đúng là tên bao gồm các kí tự là chữ cái (bao gồm các chữ cái Tiếng Việt và các tiếng khác), kí số, dấu gạch dưới (_), nhưng không được bắt đầu là kí số (và do đó không có khoảng trắng hay các dấu như các toán tử, dấu câu, ...).

Thật ra, *việc đặt tên là cả một nghệ thuật*. Ngoài việc phải đặt tên đúng qui tắc, ta nên đặt tên ngắn gọn như trong Toán nhưng cũng cần rõ ràng, đầy đủ để gợi nhớ đến ý nghĩa của nó (tức là đọc tên thì biết nó kí hiệu cho cái gì).¹³ Cũng lưu ý là Python **phân biệt chữ hoa chữ thường** (case-sensitive), tức là x và X hay hoàng và Hoàng là các tên khác nhau.

Sở dĩ tên kí hiệu cho một giá trị thường được gọi là biến, nghĩa là có thể thay đổi, vì tên đó có thể được đặt lại để kí hiệu cho một giá trị khác. Khi đó, ta còn nói, tên được định nghĩa lại. Ngược lại, chuỗi số ta gõ trong Python được gọi là **hằng** (literal) vì nó kí hiệu cho một giá trị cố định. Chẳng hạn, thử minh họa sau:

```
1 >>> a = 10; b = 2.5; c = "Hello"
2 >>> print(a, b, c)
    10 2.5 Hello
3 >>> b = a; c = b; a = a + 20
4 >>> print(a, b, c)
    30 10 10
```

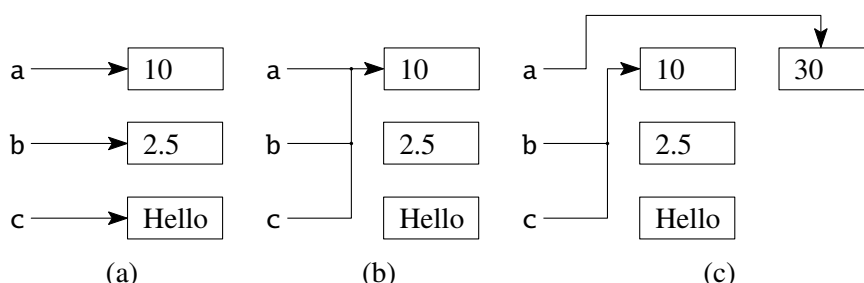
Dòng đầu tiên gồm 3 lệnh (phân cách bởi dấu ; như ta đã biết). Lệnh gán đầu tiên định nghĩa tên (hay biến) a là số nguyên 10, được mô tả bởi **hằng số nguyên** (integer literal) 10.¹⁴ Tương tự, lệnh gán thứ 2 gán cho biến b giá trị là số thực 2.5 do **hằng số thực** (floating point literal) 2.5 mô tả và lệnh gán thứ 3 gán cho biến c

¹²Dấu = trong Toán được dùng cho nhiều mục đích (lạm dụng kí hiệu), trong đó có trường hợp là “so sánh bằng” và cũng có trường hợp là “kí hiệu cho” như ta đã thấy trong bài toán trên. Kĩ hơn, trong Toán, người ta thường dùng dấu := hoặc $\stackrel{\text{def}}{=}$ với ý nghĩa là “kí hiệu cho” hay “được định nghĩa là”.

¹³Rõ ràng, ta nên tránh dùng các tên như “Người mà ai cũng biết là ai” (thay bằng Voldemort chẳng hạn) vì sẽ phải gõ một nghĩ!:)

¹⁴Bạn cần nhận ra khác biệt tinh tế chỗ này, dãy gồm 2 kí tự viết liên tiếp 10 mà bạn gõ trong lệnh được gọi là hằng, còn cái mà nó mô tả là giá trị 10.

giá trị là chuỗi Hello do **hằng chuỗi** (string literal) "Hello" mô tả. Hình (a) dưới đây mô tả “tình hình” sau khi Python thực thi 3 lệnh ở Dòng 1.



Trong lệnh xuất ở Dòng 2, ta dùng đến 3 biến a, b, c. Khi đó, Python sẽ lấy tương ứng 3 giá trị mà 3 biến này tham chiếu đến và xuất ra như kết quả sau Dòng 2 cho thấy. Dòng 3 gồm 3 lệnh. Lệnh đầu tiên định nghĩa lại (tức gán lại) biến b bằng giá trị mà a đang tham chiếu đến. Nói cách khác, sau lệnh này, b và a cùng tham chiếu đến giá trị 10 (là giá trị mà a đang tham chiếu). Lệnh thứ 2 định nghĩa lại biến c bằng giá trị mà b đang tham chiếu đến. Sau lệnh này, cả 3 biến a, b, c đều cùng tham chiếu đến giá trị 10 như Hình (b) trên. Lệnh thứ 3 trong Dòng 3 rất thú vị: biến a xuất hiện ở hai bên dấu gán (=). Python lượng giá biểu thức bên phải dấu gán trước, như vậy, a (bên phải) được dùng, là giá trị 10 (giá trị biến a đang trỏ đến). Sau đó, kết quả lượng giá biểu thức bên phải là 30 được dùng để định nghĩa lại biến a (bên trái). Như vậy, sau lệnh này, a trỏ đến giá trị 30. Ta còn nói biến a cập nhật giá trị mới là 30 (thay cho giá trị cũ là 10). Hình (c) trên cho thấy tình hình sau khi Python thực thi 3 lệnh ở Dòng 3.¹⁵ Từ đó, kết quả xuất ra của lệnh xuất ở Dòng 4 là dễ hiểu.

Rõ ràng *thứ tự thực hiện các lệnh là rất quan trọng*. Chẳng hạn, giả sử cũng 3 lệnh ở Dòng 3 nhưng đổi lại theo thứ tự là:

```
3 >>> c = b; a = a + 20; b = a
```

thì kết quả xuất ra của lệnh xuất ở Dòng 4, theo bạn, là gì? Bạn đoán thử (tốt nhất là vẽ hình ra) rồi so với kết quả mà Python xuất ra. Nếu khác thì bạn nên xem kĩ lại trước khi đi tiếp nhé. (Tạm dừng ... lấy bút giấy ... vẽ hình ... chạy thử ...)

Mô típ cập nhật giá trị cho một biến dựa trên giá trị cũ của nó rất hay gặp nên bạn cần quen thuộc. Chẳng hạn, ta có các cách tính 2^{32} như sau:

[illegible]

¹⁵Bạn có thể phân vân về các giá trị (các ô) “không dùng nữa” là 2.5 và Hello ở Hình (c). Vâng, chúng được gọi là “rác” và Python tự động “dọn rác” thường xuyên! (xem Phần ??.)

```

4 >>> x = 2; x = x * x; x = x * x; x = x * x; \
5     x = x * x; x = x * x; x
4294967296

```

Nếu không được dùng phép mũ mà chỉ dùng phép nhân thì bạn làm sao tính 2^{32} ?¹⁶ Bạn có thể nhân 32 lần con số 2 (tức là thực hiện 31 phép nhân) như Dòng 2-3 hoặc bạn có thể chỉ dùng 5 phép nhân như Dòng 4-5. Thử tưởng tượng, bạn cần tính 2^{64} , với cách đầu bạn cần 63 phép nhân, với cách sau bạn chỉ cần thêm 1 phép nhân ($2^{32} \times 2^{32} = 2^{(32+32)} = 2^{64}$). Quá khác biệt đúng không nào!

Khi gõ biểu thức quá dài, ta có thể “xuống dòng” bằng cách đặt nó trong cặp ngoặc tròn như ở Dòng 2-3. Cặp ngoặc tròn không làm thay đổi biểu thức nhưng nó “ép buộc” Python cho ta xuống dòng vì Python phải “khớp” dấu) với dấu (. Cách khác, ta dùng dấu \ để “yêu cầu” Python cho ta xuống dòng như ở Dòng 4-5. Trường hợp ta xuống dòng mà không để trong cặp ngoặc tròn hoặc dùng dấu \ thì Python sẽ thông báo `SyntaxError` như ở Phần 1.2.

Python có một cái tên đặc biệt là `_` dùng để kí hiệu cho kết quả tính toán gần nhất (và dĩ nhiên giá trị của `_` sẽ thay đổi liên tục khi ta yêu cầu Python tính toán). Kí hiệu này tương tự phím `Ans` trên các calculator. Dùng kí hiệu này, số 2^{32} có thể được tính một cách kì bí như sau:

```

1 >>> 2; _ * _; _ * _; _ * _; _ * _; _ * _
2
4
16
256
65536
4294967296

```

2.3 Giá trị luận lý và toán tử so sánh

Trở lại bài toán chứng minh 2 số nghịch đảo ở phần trên. Ta có thể làm trong Python như sau:

```

1 >>> A = 2 - 3**0.5; B = 2 + 3**0.5; A * B
1.0000000000000004
2 >>> A * B == 1
False

```

Ta thậm chí có thể yêu cầu Python kiểm tra xem tích của A, B có là 1 không bằng cách dùng dấu `==`, lưu ý là 2 dấu bằng (=) viết liền (phân biệt với 1 dấu = trong lệnh gán). Mục đích của ta là yêu cầu Python kiểm tra xem một khẳng định nào đó là **đúng** (true) hay **sai** (false) và Python sau khi kiểm tra sẽ báo True nếu

¹⁶Bạn nên biết, thời gian Python thực hiện phép mũ lâu hơn nhiều so với phép nhân.

đúng và False nếu sai; cụ thể, ta yêu cầu Python kiểm tra “tích của A với B có bằng 1 hay không” và Python báo là “không”.

Nếu nhìn rộng hơn thì ta cũng đang yêu cầu Python tính (tức lượng giá) nhưng kết quả không phải là số như thông thường mà là **giá trị luận lý** (boolean value). Vì Python vẫn xem là tính nên $A * B == 1$ vẫn là một biểu thức và được gọi là **biểu thức luận lý** (boolean expression) để phân biệt với biểu thức số học là biểu thức có giá trị số mà ta đã quen thuộc. Và do đó, $==$ là toán tử, gọi là **toán tử so sánh bằng** (equality comparison operator).

Như mong đợi, Python có các **toán tử so sánh** (comparison operator) khác mà ta hay gặp trong Toán là: $<$ (nhỏ hơn), $<=$ (nhỏ hơn hoặc bằng, \leq), $>$ (lớn hơn), $>=$ (lớn hơn hoặc bằng, \geq), $!=$ (khác, \neq). *Tất cả các toán tử so sánh này đều có độ ưu tiên như nhau và thấp hơn các toán tử số học.* Chẳng hạn, thử minh họa sau:

```
1 >>> print(1 <= 1 - 1e-10, 1 != 1.0, (1e10 - 1)/1e10 >= 1.0)
False False False
2 >>> print(1.00000 > 1 > 0.9999, 1 < 2 < 3 < 4 < 5)
False True
```

Lưu ý, như Dòng 2 cho thấy, ta có thể kiểm tra nhiều so sánh “cùng lúc” bằng cách “nối” các so sánh như trong Toán.

2.4 Lỗi

Ta thường xuyên mắc lỗi khi viết câu Tiếng Việt, Tiếng Anh, ...; viết lệnh Python cũng vậy. Và cũng như việc diễn đạt nội dung bằng ngôn ngữ giữa người viết và người đọc nói chung, việc viết lệnh cho Python thực hiện (người viết là ta và người đọc là Python) thường gặp 3 loại lỗi phân theo giai đoạn và mức độ nghiêm trọng.

Loại lỗi đầu tiên, sớm nhất và đơn giản nhất, là lỗi viết không đúng qui định của ngôn ngữ mà ở đây là ngôn ngữ Python. Qui định của ngôn ngữ thường được gọi là **ngữ pháp** (grammar) hay **cú pháp** (syntax) nên loại lỗi này được gọi là **lỗi cú pháp** (syntax error). Python thông báo `SyntaxError` khi lệnh có lỗi cú pháp như ta đã thấy trong Phần 1.2. Sau đây là các ví dụ khác:

```
1 >>> 74 * * 2
SyntaxError: invalid syntax
2 >>> 1 = 1
SyntaxError: cannot assign to literal
3 (a = 1) > 0
SyntaxError: invalid syntax
```

Vì không viết liền 2 ký tự $*$ nên ta bị lỗi cú pháp (các toán tử $//$, $==$, $!=$, $<=$, $>=$ cũng vậy). IDLE bôi màu chỗ sai, cụ thể là ký hiệu $*$ thứ 2, để ta biết chỗ cần sửa. Tại sao việc viết rời 2 dấu $*$ lại bị lỗi cú pháp? Nếu viết sát 2 dấu $*$ thì đó là toán tử mũ nên Dòng 1 là biểu thức hợp lệ. Ngược lại, nếu viết rời 2 dấu $*$ thì mỗi dấu $*$ được Python hiểu là một toán tử nhân nên Dòng 1 không là biểu thức hợp lệ.

Ở Dòng 2, ta thiếu mất một dấu = để “so sánh bằng” nên bị lỗi cú pháp. Trong trường hợp này, Python không thể hiểu một dấu = là phép gán được vì bên trái dấu = không phải là tên mà là số. Điều này được thể hiện trong mô tả “chẩn đoán lỗi” đi kèm với thông báo `SyntaxError (cannot assign to literal)`.¹⁷

Dòng 3 có lỗi cú pháp vì dấu = không phải là toán tử nên `(a = 1)` không phải là biểu thức. Dấu = là một thành phần của lệnh gán mà ta có thể xem nó như là “dấu câu”.¹⁸

Rõ ràng, ta cần loại bỏ tất cả các lỗi cú pháp (bằng cách viết đúng cú pháp) thì Python mới chấp nhận thực hiện lệnh. Lúc mới học Python, ta thường bị lỗi cú pháp (tương tự việc ta hay bị lỗi khi mới học viết) nhưng lỗi này sẽ ít dần và không còn nữa khi ta quen thuộc Python.¹⁹

Nếu Python chấp nhận thực thi lệnh nhưng bị lỗi khi thực thi thì ta có loại lỗi thứ hai. Loại lỗi này, do đó, được gọi là **lỗi thực thi** (runtime error). Một lỗi hay gặp như vậy là “**lỗi chia cho 0**” (`ZeroDivisionError`) như minh họa sau:²⁰

```
1 >>> 10 // (2**64 - 2**2**6)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    10 // (2**64 - 2**2**6)
ZeroDivisionError: integer division or modulo by zero
```

Các phép chia `//` và `%` cũng có thể bị lỗi tương tự.²¹

Một lỗi thực thi khác hay gặp với các tên đó là việc gõ sai (hay nhầm) tên. Thử minh họa sau:

```
1 >>> name = "hoàng"
2 >>> print("Hello", name)
Hello hoàng
3 >>> print("Hello", Name)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print("Hello", Name)
NameError: name 'Name' is not defined
```

¹⁷Mô tả chẩn đoán lỗi thường không tốt trong nhiều trường hợp vì, nói chung, thường chỉ có một cách đúng nhưng có nhiều cách sai nên Python không đủ thông minh và cũng không đủ thông tin để đoán được là sai theo cách nào.

¹⁸Điều này khác với một số ngôn ngữ như C/C++, trong đó, dấu = là một toán tử và `(a = 1) > 0` là một biểu thức hợp lệ.

¹⁹Có thể nói, Python cực kì khắc khe với các lỗi cú pháp: Python không chấp nhận dù chỉ một lỗi nhỏ nhất. Điều này khác với ngôn ngữ tự nhiên, nhiều lỗi được du di bỏ qua, nhưng đây lại là một trong những nhược điểm của ngôn ngữ tự nhiên vì có rất nhiều cách du di khác nhau, tạo nên những mơ hồ trong việc hiểu nội dung được mô tả.

²⁰Phần đầu của thông báo (Traceback ...) hơi lằng nhằng mà ta sẽ tìm hiểu sau.

²¹Bạn có thể thắc mắc là tại sao Python vẫn thực thi khi biết rằng không thể chia cho 0? Python không phải thần thánh! Không dễ, thậm chí là không thể biết được là có bị chia cho 0 hay không trong các tình huống thực thi khó.

Ta đã dùng biến `name` để chứa tên của một người. Lệnh xuất ở Dòng 2 dùng đúng tên biến là `name` nên không vấn đề gì nhưng ở Dòng 3 ta dùng sai tên (`Name` thay cho `name`). Đây không phải là lỗi cú pháp vì không có vấn đề gì về cú pháp trong các lệnh (lệnh xuất ở Dòng 3 “cùng dạng” với lệnh xuất ở Dòng 2). Tuy nhiên, khi Python thực thi lệnh xuất ở Dòng 3 nó sẽ phải “tra” giá trị mà biến `Name` tham chiếu đến và phát hiện tên này chưa được định nghĩa (chưa được gán) trước đó (nhớ là Python phân biệt chữ hoa chữ thường). Lỗi thực thi này được Python gọi là `NameError` và thông báo lỗi cho biết chi tiết (`is not defined` nghĩa là chưa được định nghĩa). Lưu ý, Python không thể biết được ý định của ta là dùng tên `name` nhưng viết nhầm thành `Name`.²² Không có khái niệm **lỗi chính tả** (spelling error), tức là lỗi viết nhầm, trong Python như ta hay nói trong ngôn ngữ tự nhiên.

Với lỗi cú pháp, Python không chịu thực thi (vì không hiểu được lệnh); với lỗi thực thi, Python thực thi nhưng bị lỗi (và dừng thực thi với thông báo lỗi). Còn loại lỗi thứ ba, tinh vi và nguy hiểm hơn nhiều, là **lỗi logic** (logical error). Với loại lỗi này, Python vẫn thực thi nhưng không dừng và không thông báo lỗi, nghĩa là không có dấu hiệu gì để biết là có lỗi. Chẳng hạn, tôi đã nói rằng Python dùng dấu chấm thập phân cho số thực chứ không dùng “dấu phẩy thập phân” như tiếng Việt. Nếu bạn vẫn ngoan cố dùng thì sao? Thử há:

```
1 >>> a = 2,5
2 >>> print(a)
(2, 5)
3 >>> a == 2.5
False
4 >>> a = 2.5
5 >>> print(a)
2.5
```

Python vẫn chạy và không thông báo lỗi gì. Python không biết lỗi nào xảy ra. Chính xác hơn, với Python, không có lỗi gì trong lệnh gán ở Dòng 1. Vấn đề là có sự “hiểu lầm” giữa người viết (là ta) và người đọc (là Python). Ta viết một đẳng (ta viết 2, 5 để mô tả số thực có phần nguyên là 2 và phần lẻ là một nửa), Python hiểu một nẻo (Python hiểu 2, 5 là cặp số nguyên gồm số thứ nhất là 2 và số thứ hai là 5). Do đó kết quả `False` cho so sánh ở Dòng 3 là “phải” rồi. Tuy nhiên, như đã nói nhiều lần, Python không có lỗi, ta cần phải viết theo ngôn ngữ và cách hiểu của Python. Nhân tiện, dấu phẩy (,) được Python dùng với ý chung là phân cách các thành phần của “cấu trúc” nào đó. Chẳng hạn, nó được dùng để phân cách số thứ nhất là 2 với số thứ hai là 5 trong cặp số (2, 5) hay phân cách các “đối số” trong lệnh xuất như `print(a, b, c)`.²³

Lưu ý, lệnh gán ở Dòng 4 “có vẻ như” không ổn. Nếu “thực sự” ta muốn gán thì không có vấn đề gì, còn như ta muốn kiểm tra giá trị của biến `a` có bằng 2.5 hay

²²Python không phải thần thánh!

²³Ta sẽ hiểu hơn về cặp số và “đối số” sau.

không (như so sánh ở Dòng 3) mà viết thiếu một dấu = (lẽ ra là ==) thì Python sẽ gán lại giá trị cho a (thay vì so sánh) nên đây là một lỗi logic.

Các lỗi thực thi và lỗi logic được gọi chung là **bug**. Công việc tìm và xóa bug trong các lệnh Python được gọi là **debug**. Đây là một phần quan trọng của việc lập trình Python (tức là việc viết các lệnh “đúng”). Ta sẽ rèn luyện thường xuyên kỹ năng này qua các bài học.

2.5 Phong cách viết

Nếu như việc viết đúng (không lỗi) là rất quan trọng thì *việc viết đẹp, rõ ràng, dễ nhìn cũng quan trọng không kém*. Ở phần trên, bạn đã thấy rằng không được dùng khoảng trắng giữa 2 kí tự * khi mô tả toán tử mũ. Ngược lại, bạn được phép dùng và nên dùng khoảng trắng trong cách viết `15*64 + 25*100`. Có kí tự trắng (tương ứng với phím Space) trước và sau toán tử + và không nên có kí tự trắng xung quanh toán tử *. Mục đích của việc này là để dễ nhìn, nó cho thấy rõ 15 được nhân với 64 và 25 được nhân với 100 trước rồi sau đó kết quả của chúng mới được cộng lại. Trường hợp chỉ có phép nhân thôi thì ta lại nên viết là `15 * 64`, tức là dùng khoảng trắng trước và sau toán tử *. Những quy tắc này không phải bắt buộc, nó phụ thuộc vào sở thích, tính cách của người viết và được gọi là **phong cách viết** (style) (mà trong ngôn ngữ tự nhiên gọi là văn phong).

Nói chung, bạn có quyền dùng phong cách viết riêng của mình nhưng lúc mới học, bạn nên dùng phong cách viết mà **cộng đồng Python** (Python community) hay dùng.²⁴ Tài liệu này dùng phong cách viết đó, gọi là “**PEP 8**”, và bạn cũng nên như vậy, bằng cách bắt chước cách trình bày lệnh của tôi trong các minh họa.

Một cách để hạn chế các lỗi “nhầm tên” là dùng tên có ý nghĩa và thống nhất cách đặt tên (Tiếng Việt hay Tiếng Anh, chữ hoa hay chữ thường, ...). Ta cũng *nên dùng danh từ để đặt tên biến*. Trường hợp tên gồm nhiều thành phần (từ hay tiếng) thì có nhiều cách đặt tên; với tên biến, ta nên dùng chữ thường và kí hiệu gạch dưới (_) để phân cách các thành phần, ví dụ `full_name`, `date_of_birth`, `my_love`, ... hay `họ_và_tên`, `ngày_sinh`, `người_ấy`, ... Nhân tiện, dấu _ cũng có thể được dùng để viết số để đọc như minh họa sau:

```
1 >>> a = 1_000_000_000; b = 12_345.678_9
2 >>> print(a, b)
1000000000 12345.6789
```

Lưu ý, Python cho phép dùng các kí tự Tiếng Việt để đặt tên nhưng ta không nên lạm dụng, nhất là khi làm việc chung với nhiều người (trong đó có người nước ngoài). Nói chung, trong thời đại hội nhập, ta *nên dùng tiếng Anh càng nhiều càng tốt*. Những quy ước này cũng là một phần quan trọng của phong cách viết.

²⁴Suy cho cùng thì mục đích viết các lệnh Python là để Python hiểu rõ yêu cầu và thực thi chứ không thể hiện tính cách, đặc điểm, cảm xúc, ... của người viết. Không có “cái tôi” khi viết lệnh Python nên bạn cần “phong cách viết chung” để mọi người đều dễ đọc, dễ hiểu.

Tóm tắt

- Python là một công cụ giúp thực hiện tự động việc tính toán mà trường hợp đơn giản là tính toán số học.
- Biểu thức mô tả một giá trị là kết quả tính toán từ các số nguyên, số thực, ... với các phép toán được kí hiệu bởi các toán tử. Các toán tử số học hay gặp là: + (cộng), - (trừ), * (nhân), / (chia), // (chia nguyên), % (chia lấy dư), - (đối) và ** (mũ).
- Giá trị luận lý mô tả cho đúng/sai, có/không, được/mất, ... với 2 hằng luận lý là True/False. Python có các toán tử so sánh giúp so sánh các giá trị số như trong Toán: <, <=, >, >=, ==, !=.
- Python dùng dấu chấm thập phân chứ không dùng “dấu phẩy thập phân”.
- Thứ tự thực hiện các phép toán được quyết định bởi các cặp ngoặc tròn, độ ưu tiên và tính kết hợp của các toán tử. Các toán tử số học có độ ưu tiên cao hơn các toán tử so sánh.
- Tư duy lập trình, tư duy Python là tư duy thực tế, cụ thể, rõ ràng và chi tiết.
- Lệnh gán cho phép tạo các tên (biến) kí hiệu cho các giá trị. Các tên cũng có thể được định nghĩa lại để tham chiếu đến giá trị khác hoặc cập nhật giá trị mới dựa trên giá trị cũ.
- Các tên phải được đặt đúng qui tắc và nên ngắn gọn, rõ ràng, gợi nhớ đến ý nghĩa của chúng. Python phân biệt chữ hoa chữ thường và cho phép dùng kí hiệu tiếng Việt. Tên đặc biệt _ (Ans) được Python dùng để tham chiếu đến giá trị của biểu thức vừa tính.
- Hằng là phương tiện của Python giúp mô tả các giá trị cụ thể, cố định.
- Thứ tự của các lệnh là rất quan trọng.
- Có 3 loại lỗi trong Python: lỗi cú pháp xảy ra khi lệnh được viết sai cú pháp, lỗi thực thi xảy ra khi Python thực thi lệnh và bị lỗi, lỗi logic là những hiểu lầm hay bất thường mà Python không phát hiện được.
- Lỗi thực thi và lỗi logic được gọi chung là bug. Tìm và sửa lỗi, tức debug, là kĩ năng quan trọng cần rèn luyện.
- Ta nên viết lệnh đẹp, rõ ràng, dễ nhìn và tuân theo phong cách viết mà cộng đồng Python hay dùng, phong cách PEP 8.

Bài tập

2.1 Dùng Python,²⁵ tính nhanh:²⁶

²⁵Tất cả các bài tập trong bài học này đều dùng Python để làm!

²⁶SGK Toán 8 Tập 1. Tôi viết lại như cách SGK đã viết nhưng bạn cần phải dùng các kí hiệu phù hợp trong Python, chẳng hạn, toán tử nhân là * (thay cho dấu .) hay dấu phân cách thập phân là dấu chấm (thay cho dấu phẩy), ... Bạn cũng biết “tính nhanh” nghĩa là gì rồi đó (xem lại Phần 2.1).

- (a) $45^2 + 40^2 - 15^2 + 80.45$
 (b) $37, 5.6, 6 - 7, 5.3, 4 - 6, 6.7, 5 + 3, 5.37, 5$
 (c) $\left(\frac{3}{4}\right)^5 : \left(\frac{3}{4}\right)^3$

2.2 Tính nhanh giá trị của biểu thức:²⁷

- (a) $M = x^2 + 4y^2 - 4xy$ tại $x = 18$ và $y = 4$
 (b) $N = 8x^3 - 12x^2y + 6xy^2 - y^3$ tại $x = 6$ và $y = -8$

2.3 Đoán tuổi:²⁸

Bạn lấy tuổi của mình: cộng thêm 5, được bao nhiêu đem nhân với 2, lấy kết quả trên cộng với 10, nhân kết quả vừa tìm được với 5, rồi tất cả trừ đi 100. Đọc kết quả sau cùng và tôi có thể đoán được tuổi của bạn!

- (a) Gọi x là tuổi của bạn (gán x là tuổi của bạn), y là kết quả sau cùng (gán y là giá trị tính từ x qua biểu thức mô tả trên), tìm cách tính lại x từ y (tính biểu thức trên y có giá trị như x).
 (b) Không dùng biến, dùng tên đặc biệt `_ (Ans)`, từ tuổi của mình, tính lần lượt từng bước các thao tác trên đến khi được kết quả cuối cùng rồi sau đó là tuổi ban đầu.

Gợi ý: Giả sử bạn 16 tuổi, các bước đầu tiên là:

```
1 >>> 16
    16
2 >>> _ + 5
    21
```

2.4 Chứng minh rằng $55^{n+1} - 55^n$ chia hết cho 54 với:²⁹ (a) $n = 10$. (b) $n = 1000$. (c) $n = 1000000$.³⁰ (d) $n = 10000000$.³¹ (e) Mọi số tự nhiên n .³²

2.5 Không được dùng phép mũ, tính 2^{60} .

Gợi ý: Dùng phép nhân, chia và đặt biến phù hợp.

2.6 Lệnh gán tăng cường.

Một mô típ cập nhật giá trị hay gặp là “cộng thêm vào giá trị cũ”: `<Name> = <Name> + <Expr>`. Python cho phép viết gọn lệnh gán này bằng **lệnh gán tăng cường** (augmented assignment statement) là `<Name> += <Expr>`. Chẳng hạn, với biến a đang tham chiếu đến giá trị 10 thì sau lệnh gán tăng cường `a += 20` (tương đương với lệnh gán thông thường `a = a + 20`), biến a sẽ được “cộng thêm 20 vào giá trị cũ”, tức là tham chiếu đến giá trị mới 30. Tương tự, Python cũng hỗ trợ lệnh gán tăng cường cho các phép toán khác bằng cách dùng các dấu tương ứng `-=`, `*=`, `/=`, `//=`, `%=`, `**=`.

²⁷SGK Toán 8 Tập 1.

²⁸Chỉnh sửa từ SGK Toán 8 Tập 1.

²⁹Chỉnh sửa từ SGK Toán 8 Tập 1. Nhớ lại, số tự nhiên a được gọi là chia hết cho số tự nhiên b ($b > 0$) nếu a chia b dư 0.

³⁰Bạn đời Python xúu nhé!

³¹Bạn đời Python nổi không? Nếu không thì xem cách “ngắt” thực thi Python ở Bài tập 2.8.

³²Làm Toán câu này nhé. Làm Python được không ta? Bài tập này cho thấy sự “vi diệu” của Toán!

Ví dụ, ở Bài tập 2.3, giả sử bạn 16 tuổi, bạn có thể tính kết quả sau cùng bằng cách sau:

```
1 >>> x = 16
2 >>> x += 5; x *= 2; x += 10; x *= 5; x -= 100
3 >>> x
160
```

Phương pháp Horner. Giả sử không được dùng phép mũ, ta tính giá trị của biểu thức $P = x^4 + 4x^3 + 6x^2 + 5x + 2$ tại $x = 5$ bằng cách nào?

```
1 >>> x = 5
2 >>> P = x*x*x*x + 4*x*x*x + 6*x*x + 5*x + 2
3 >>> P
1302
```

Để tính P theo cách trên, ta tốn 9 phép nhân và 4 phép cộng. Ta có thể tính nhanh hơn (dùng ít phép toán cơ bản, cộng trừ nhân chia, hơn) bằng nhận xét:

$$\begin{aligned} P &= x^4 + 4x^3 + 6x^2 + 5x + 2 \\ &= (x^3 + 4x^2 + 6x + 5)x + 2 \\ &= ((x^2 + 4x + 6)x + 5)x + 2 \\ &= (((x + 4)x + 6)x + 5)x + 2 \end{aligned}$$

Cách tính này, được gọi là **phương pháp Horner** (Horner's method, đặt theo tên nhà Toán học Anh William George Horner), chỉ tốn 3 phép nhân và 4 phép cộng.³³ Cách tính Horner có thể được viết bằng cách dùng phép gán tăng cường như sau:

```
1 >>> x = 5; P = x
2 >>> P += 4; P *= x; P += 6; P *= x; P += 5; P *= x; P += 2
3 >>> P
1302
```

Ta đã “hiện thực” công thức nhận xét trên: P nhận giá trị x , cộng thêm 4, nhân thêm x , cộng thêm 6, ...

Tương tự, dùng phép gán tăng cường cho cách tính Horner để tính giá trị của biểu thức $P = x^5 - 4x^3 + (x - 1)^2 - 2$ tại $x = 5$ (và đối chiếu kết quả với cách tính thông thường).

2.7 Phương pháp Newton. Để tính căn bậc 2 của một số dương, ta có thể dùng cách “tính lặp” kì diệu sau đây: đặt x là số cần tính căn, lặp lại việc tính biểu thức Python `_/2 + x/(2*_)`, sau vài lần ta sẽ có giá trị gần với căn của x . Ví dụ, tính $\sqrt{2}$ như sau:

³³Trường hợp biểu thức có các số hạng mũ lớn thì cách tính này giảm rất nhiều phép nhân, chẳng hạn từ 1 triệu chỉ còn 1 ngàn phép nhân. Đây thực sự là một phương pháp tính toán rất hay. (Này mới là tính nhanh nhé!)

```

1 >>> 2 ** 0.5
  1.4142135623730951
2 >>> x = 2
3 >>> _/2 + x/(2*_ )
  2.6213203432709573
4 >>> _/2 + x/(2*_ )
  1.692147311338584
5 >>> _/2 + x/(2*_ )
  1.4370387526790456
6 >>> _/2 + x/(2*_ )
  1.4143948342112873
7 >>> _/2 + x/(2*_ )
  1.4142135739891866
8 >>> _/2 + x/(2*_ )
  1.4142135623730951

```

Các giá trị tính ra “hội tụ” rất nhanh đến $\sqrt{2}$. Cách tính này được gọi là **phương pháp Newton** (Newton’s method, đặt theo tên nhà Toán học và Vật lý Anh Isaac Newton). Lưu ý, trong IDLE bạn có thể “lấy lại lệnh trước đó” như trong Bài tập 2.8 và do đó không cần gõ lại biểu thức mỗi lần tính.

Thử tính căn các số khác (đặt giá trị cho x) bằng cách tính lặp như trên (và đổi chiều kết quả với phép căn thông thường (mũ 1/2)).

2.8 Dọc IDLE, thử các chức năng tiện lợi sau:

- IDLE cho phép ta “ngắt thực thi” khi động cơ Python đang thực thi một công việc nào đó bằng Shell → Interrupt Execution (phím tắt Ctrl+C). Chức năng này hay được dùng khi ta không muốn đợi Python thực thi xong một công việc quá lâu nào đó. Trường hợp chức năng này “không có tác dụng” (động cơ Python vẫn làm miệt mài) thì dùng chức năng mạnh hơn là “khởi động lại” động cơ Python bằng Shell → Restart Shell (Ctrl+F6). Dĩ nhiên, ta cũng có thể đóng IDLE rồi mở lại.
- IDLE nhớ các lệnh mà ta đã gõ trước đó trong một danh sách gọi là History. Ta có thể duyệt qua danh sách này để lấy lại các lệnh đã gõ trước đó bằng Shell → Previous History (Alt+P) hay Shell → Next History (Alt+N). Chức năng này rất tiện lợi khi “tính lặp”, ta lấy lại lệnh vừa thực thi trước đó (Alt+P) và nhấn Enter (mà không phải gõ lại lệnh). Sau khi lấy lại lệnh, ta cũng có thể chỉnh sửa nó trước khi nhấn Enter để Python thực thi.

Ta cũng có thể dùng lại một lệnh trước đó bằng cách nhấp chuột vào dòng đó và nhấn Enter. Trong cửa sổ lệnh Command Prompt, ta dùng phím mũi tên lên (Up), xuống (Down) để lấy lại lệnh.

Bài 3

Tính toán nâng cao

Bài này tiếp tục việc tính toán số học trên Python với các hỗ trợ nâng cao hơn là hàm, module và thư viện. Ta sẽ dùng một vài từ mang nghĩa đặc biệt trong Python là từ khóa và tìm hiểu kĩ hơn về chế độ tương tác và phiên làm việc. Một kĩ năng “mềm” rất quan trọng cũng được đề cập là kĩ năng tra cứu. Phần bài tập hướng dẫn giải quyết vài vấn đề của số thực.

3.1 Hàm dựng sẵn

Trong bài trước, ta đã thấy rằng Python làm việc với số thực “khá tệ”. Thật ra, hầu như tất cả các công cụ trên máy tính đều làm việc với số thực tệ như Python. Lí do là vì số thực có bản chất rất phức tạp, khó xử lý hơn số nguyên nhiều.¹ Biểu diễn thập phân của số $\frac{1}{2}$ là 0.5, hữu hạn, nhưng của số $\frac{1}{3}$ là 0.333333..., vô hạn tuần hoàn. Khó hơn nữa, số $\sqrt{2}$ có biểu diễn vô hạn không tuần hoàn 1.414213... mà Toán gọi là số vô tỉ hay số π có biểu diễn là 3.141592... mà Toán gọi là số siêu việt. Tất cả những gì ta có thể làm được (cho đến giờ) với số thực là biểu diễn gần đúng chúng (hoặc giới hạn phạm vi của chúng như trong Bài tập 3.8 và 3.9). Điều đó cũng có nghĩa là ta phải chấp nhận **sai số** (error) trong nhiều trường hợp. Chẳng hạn, trong Toán (nghĩa là chính xác tuyệt đối), ta có đẳng thức hiển nhiên:

$$(\sqrt{2})^2 = 2$$

Nhưng trong Python (nghĩa là chấp nhận sai số), đẳng thức trên không còn đúng nữa:

```
1 >>> (2**0.5) ** 2 == 2
False
2 >>> (2**0.5) ** 2
2.0000000000000004
```

¹Toán nói rằng tập số thực là vô hạn “không đếm được” (uncountable) còn số nguyên (hay số hữu tỉ) là vô hạn nhưng “đếm được” (countable)!

Sự thật là Python đã xấp xỉ $\sqrt{2}$ bằng con số thực lớn hơn $\sqrt{2}$ “chút xíu” và do đó khi bình phương lên ta được số lớn hơn 2 chút xíu.

Làm sao giải quyết vấn đề này? Ta thay khái niệm hoàn hảo (Toán) “2 số bằng nhau” (tức là trùng nhau) thành khái niệm thực tế hơn (Python) là “2 số rất gần nhau”. Trong Python, $(\sqrt{2})^2$ không trùng với 2 nhưng rất gần 2. Làm sao kiểm tra hai số có rất gần nhau không? Ta tìm khoảng cách giữa hai số: Toán, khoảng cách giữa 2 số thực a, b chính là $|a - b|$ (trị tuyệt đối của a trừ b). Ta đã biết cách trừ trong Python, còn tính trị tuyệt đối thì sao? Làm như sau:

```
1 >>> abs((2**0.5)**2 - 2)
4.440892098500626e-16
```

Như ta đã biết, $4.440892098500626e-16$ trong kết xuất ở trên mô tả cho số thực $4.440892098500626 \times 10^{-16}$. Như vậy, khoảng cách giữa hai số là rất rất nhỏ (xấp xỉ một phần mười triệu tỉ). Vấn đề nữa, khoảng cách nhỏ bao nhiêu thì được xem là rất gần, tức là có thể xem là trùng nhau? Điều này tùy trường hợp, tùy ứng dụng, tùy bạn, ... Chẳng hạn, nếu khoảng cách nhỏ hơn một phần tỉ là “xem như trùng nhau” thì ta có thể kiểm tra $(\sqrt{2})^2 = 2$ trong Python như sau:

```
1 >>> abs((2**0.5)**2 - 2) < 1e-9
True
```

Dĩ nhiên, bạn có thể thay hằng số $1e-9$ bằng $10**-9$ hoặc $1/1e9$ hoặc $1/10**9$. Bài toán chứng minh 2 số nghịch đảo ở Phần 2.2, như vậy, được giải trong Python bằng cách viết:

```
1 >>> A = 2 - 3**0.5; B = 2 + 3**0.5; abs(A*B - 1) < 1e-9
True
```

Điều cần học ở đây là cách ta tính trị tuyệt đối trong Python như trên. Ta không dùng toán tử nữa mà dùng **hàm** (function), cụ thể là hàm có tên `abs`. Hàm số chính là cách mà Toán dùng để mô tả các tính toán phức tạp và chúng cũng được dùng rất nhiều trong Python. Tuy nhiên, *hàm trong Python khác biệt so với hàm số của Toán, chúng linh hoạt hơn và thực tế hơn.*

Cách tốt nhất để hình dung về các hàm (trong Python) là xem chúng như là các **dịch vụ** (service) mà trường hợp hay gặp là các dịch vụ tính toán (mà ta có thể xem là hàm số). Python cung cấp sẵn hầu hết các dịch vụ cần thiết, hay gặp mà nếu cần ta có thể dùng. Làm sao để dùng các dịch vụ này, tức là dùng hàm hay **gọi hàm** (calling function). Đầu tiên, ta cần biết tên dịch vụ tức là **tên hàm** (function name) vì cũng như mọi thứ khác, Python dùng tên để tham chiếu (chứ không dùng mã số như 114 – cứu hỏa, 115 – cứu thương, ...). Kế tiếp, ta phải biết dịch vụ đó cần nhận những cái gì để ta cung cấp khi dùng nó, tức là hàm cần nhận những **đối số** (argument) nào để ta cung cấp khi gọi nó. Chẳng hạn, dịch vụ tính trị tuyệt đối (truy cập bằng tên `abs`) cần một giá trị số (để nó tính trị tuyệt đối), dịch vụ cứu hỏa (truy cập bằng số 114) cần biết địa điểm bị cháy, ...

Cách chung để gọi hàm trong Python là viết:

<Func>(<Args>)

Với <Func> là tên tham chiếu đến hàm cần gọi và <Args> cung cấp các đối cho hàm. Nếu không có đối số thì để trống (nhưng vẫn phải có cặp ngoặc tròn) còn nếu có nhiều đối số thì dùng dấu phẩy (,) phân cách các đối số. Vì các đối số cung cấp giá trị nên chúng là các biểu thức.

Điều nữa, các hàm (dịch vụ) có thể trả về kết quả gì đó hoặc không, gọi là **giá trị trả về** (return value) của hàm. Ví dụ, dịch vụ tính trị tuyệt đối sẽ trả về trị tuyệt đối còn dịch vụ cứu hỏa có lẽ không trả về gì. *Trường hợp có trả về giá trị thì lời gọi hàm có thể tham gia vào biểu thức; trường hợp không trả về giá trị thì nó đứng một mình và thường được gọi là lệnh.* Khi thực thi lời gọi hàm, Python lượng giá các đối số (nếu có), sau đó thực thi hàm tương ứng và trả về giá trị (nếu có).

Ta đã dùng hàm abs để tính trị tuyệt đối, nay tôi giới thiệu một hàm nữa cũng hay dùng là hàm làm tròn, tên là round (xem thêm Bài tập 3.5). Thử minh họa sau:

```
1 >>> x = 2 ** 0.5; print(x)
1.4142135623730951
2 >>> print(round(x, 6), round(x, 20), round(x))
1.414214 1.4142135623730951 1
```

Hàm round nhận 2 đối số, đối số thứ nhất là giá trị cần làm tròn, đối số thứ hai là số nguyên cho biết chữ số thập phân cần làm tròn. Điều đặc biệt, round cũng có thể nhận 1 đối số mà khi đó nó sẽ tính và trả về số nguyên gần nhất. Hàm round cũng mang lại giải pháp “kiểm tra bằng” cho số thực trong Python:

```
1 >>> A = 2 - 3**0.5; B = 2 + 3**0.5; round(A * B, 9) == 1
True
```

Hàm trong Python rất linh hoạt, như ta đã thấy, hàm round có thể nhận 1 hoặc 2 đối số. Thậm chí có hàm có thể nhận số lượng đối số “tùy ý” như minh họa sau:

```
1 >>> print(max(1, 2), max(1, 2, 4, 3))
2 4
2 >>> min(1, 2, 4, 3, 0.5)
0.5
```

Như tên gọi gợi ý, hàm max trả về giá trị lớn nhất trong số các đối số, còn hàm min trả về giá trị nhỏ nhất. Lưu ý, max và min nhận từ 2 đối số trở lên tùy ý.

Các hàm ta đã dùng tới giờ (abs, round, max, min) đều là các hàm tính toán, chúng cung cấp các dịch vụ tính toán. Các toán tử cũng mô tả các thao tác tính toán. Có khác biệt gì không? Ranh giới giữa chúng rất mờ nhạt như minh họa sau cho thấy:

```
1 >>> print(2 ** 0.5, pow(2, 0.5))
1.4142135623730951 1.4142135623730951
2 >>> print(2 ** 64, pow(2, 64))
18446744073709551616 18446744073709551616
```

Như bạn thấy, Python cung cấp hàm `pow` để thực hiện việc tính mũ. Dĩ nhiên, ta cũng có thể dùng toán tử mũ (`**`) để thực hiện. Toán tử thì đẹp hơn nhưng hạn chế vì chỉ có số lượng ít. Hàm thì rất linh hoạt và mạnh mẽ với số lượng rất nhiều. Một cách hình dung đúng là *toán tử và hàm đều là cách mô tả các thao tác/dịch vụ, cách mô tả bằng toán tử thì đẹp hơn còn cách mô tả bằng hàm thì mạnh mẽ và linh hoạt hơn* (xem thêm Bài tập 3.10). Cũng lưu ý là *thứ tự các đối số rất quan trọng*. Chẳng hạn, với hàm `pow`, đối số thứ nhất là cơ số còn đối số thứ 2 là số mũ.

Ta đã thấy nhiều hàm cung cấp các dịch vụ tính toán nhưng không phải tất cả các hàm đều như vậy. Python có nhiều hàm cung cấp các dịch vụ khác.² Bạn đã dùng một hàm như vậy đây, thậm chí ngay từ bài đầu tiên, hàm `print`. Như ta đã biết, hàm này không tính toán mà xuất ra các chuỗi. Thật ra, `print` rất mạnh, nó có thể xuất ra bất kì giá trị nào (số, chuỗi, luận lý, ...). Nó cũng có thể nhận không hoặc nhiều đối số. Hơn nữa, nó không trả về giá trị gì nên việc dùng nó thường được gọi là lệnh, mà theo ý nghĩa, ta đã gọi là lệnh xuất. Ví dụ:

```
1 >>> print("Căn của", 2, "là", 2 ** 0.5)
Căn của 2 là 1.4142135623730951
```

Hàm `print` cũng có 2 **đối số có tên** (keyword argument)³ là `sep` và `end`. Hơn nữa, 2 đối số này nhận **giá trị mặc định** (default value) tương ứng là " " và "\n". Thử minh họa sau để hiểu rõ hơn:

```
1 >>> print(1, 2); print(3)
1 2
3
2 >>> print(1, 2, sep="\n"); print(3)
1
2
3
3 >>> print(1, 2, end=" "); print(3)
1 2 3
4 >>> print(1, 2, sep="", end=""); print(3)
123
```

Như vậy, `print` lần lượt xuất ra giá trị của các đối số “thông thường”, phân cách bởi chuỗi do đối số `sep` qui định và sau cùng xuất thêm chuỗi do đối số `end` qui định. Để đưa giá trị cho đối số có tên, trong danh sách đối số của lời gọi hàm ta viết:

<Tên đối số>=<Giá trị>

²Suy cho cùng thì thuật ngữ dịch vụ mang nghĩa rất rộng.

³Dịch sát nghĩa là “đối số từ khóa” nhưng cách gọi này không có ý nghĩa gì hết:) Có lẽ, thuật ngữ `named argument` là tốt hơn.

với <Giá trị> là một biểu thức mà kết quả lượng giá của nó sẽ được cung cấp cho đối số tương ứng có tên là <Tên đối số>. Cũng lưu ý là PEP 8 khuyến cáo không nên dùng khoảng trắng trước và sau dấu = này.⁴

Tất cả các hàm ta dùng cho đến lúc này như `print`, `abs`, `round`, ... được gọi là **hàm dựng sẵn** (built-in function). Đây là những hàm quan trọng mà Python cung cấp sẵn để ta có thể dùng bất cứ lúc nào. Tuy nhiên, số lượng các hàm này (và do đó, các dịch vụ tương ứng) là khá hạn chế so với nhu cầu rất đa dạng của ta cho nhiều công việc khác nhau.⁵

3.2 Module và thư viện

Lớp có 40 người, theo bạn có bao nhiêu cách sắp xếp chỗ ngồi? Chỗ thứ nhất có 40 cách (40 người), chỗ thứ 2 có 39 cách (39 đứa còn lại, trừ đứa đã sắp ngồi chỗ thứ nhất), chỗ thứ 3 có 38 cách, ..., chỗ thứ 40 có 1 cách (đứa sau cùng không được lựa chọn). Vậy tổng số cách là $40 \times 39 \times 38 \times \dots \times 2 \times 1$. Giá trị này được kí hiệu là $40!$ và gọi là **giai thừa** (factorial) của 40 hay 40 giai thừa. Nhưng rốt cuộc là bao nhiêu? Ta có thể tính bằng cách nhân từ từ (chịu khó há) hoặc yêu cầu gọn hơn cho Python như sau:

```
1 >>> import math
2 >>> math.factorial(40)
8159152832478977343456112695961158942720000000000
3 >>> 1 / _
1.2256174391283858e-48
```

Bạn đã bật ngửa vì con số khủng này đúng không! Thật sự, $40!$ là con số lớn “ngoài sức tưởng tượng”.⁶ OK! Quan trọng ở đây là cách ta dùng dịch vụ tính giai thừa mà cụ thể là hàm `factorial`. Hàm này không phải là hàm dựng sẵn mà được để trong **gói dịch vụ** (module) `math`. Như vậy, ngoài các hàm dựng sẵn, Python cung cấp rất nhiều hàm nữa trong các module khác nhau. Mỗi module “đóng gói” một tập các hàm liên quan (và các thứ khác nữa).

Để dùng các hàm trong module, trước hết ta cần dùng lệnh

import <Module>

để Python **nạp** (load, import) module có tên là <Module>. Sau đó, khi dùng các hàm, ta viết thêm tên module phía trước cùng với dấu chấm (.) theo dạng:

<Module>.<Func>(<Args>)

⁴Trường hợp dấu = của lệnh gán thì PEP 8 khuyến cáo nên có khoảng trắng trước và sau.

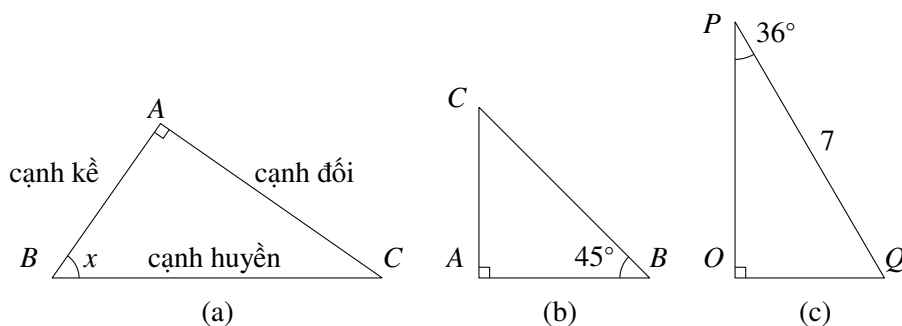
⁵Còn nhiều hàm dựng sẵn quan trọng nữa mà ta sẽ biết sau.

⁶Nghịch đảo của nó là một con số cực nhỏ, nhỏ hơn rất rất nhiều so với xác suất trúng vé số độc đắc. Điều đó cũng cho thấy, việc bạn được xếp ngồi chung bàn với người ấy là một cơ duyên cực kì hiếm nhé!

Dấu chấm, trong trường hợp này, được Python hiểu là “của”, `math.factorial` được hiểu là (hàm) `factorial` của (module) `math`.⁷ Lưu ý là ta chỉ cần nạp module một lần và các hàm của module cũng được dùng cùng một cách như các hàm dựng sẵn.

Phần sau đây minh họa thêm module `math` với các hàm **lượng giác** (trigonometry). Trước hết, ta nhớ lại vài kiến thức từ Toán Lớp 9. Cho tam giác ABC vuông tại A và đặt x là số đo góc \widehat{ABC} như Hình (a) dưới, các tỉ số lượng giác của x được định nghĩa như sau:

$$\begin{aligned}\sin x &= \frac{\text{cạnh đối}}{\text{cạnh huyền}} = \frac{AC}{BC}; & \cos x &= \frac{\text{cạnh kề}}{\text{cạnh huyền}} = \frac{AB}{BC}; \\ \tan x &= \frac{\text{cạnh đối}}{\text{cạnh kề}} = \frac{AC}{AB}; & \cot x &= \frac{\text{cạnh kề}}{\text{cạnh đối}} = \frac{AB}{AC}.\end{aligned}$$



Chẳng hạn, trong tam giác ABC vuông cân tại A ở Hình (b) trên, ta có $\widehat{B} = \widehat{C} = 45^\circ$ và $AC = AB$. Hơn nữa, theo định lý Pytago (Pythagoras) ta có $BC^2 = AC^2 + AB^2 = 2AC^2$ nên $BC = \sqrt{2}AC$. Do đó ta có các tỉ số lượng giác sau cho góc 45° :

$$\begin{aligned}\sin 45^\circ &= \frac{AC}{BC} = \frac{1}{\sqrt{2}} = \frac{\sqrt{2}}{2}; & \cos 45^\circ &= \frac{AB}{BC} = \frac{1}{\sqrt{2}} = \frac{\sqrt{2}}{2}; \\ \tan 45^\circ &= \frac{AC}{AB} = 1; & \cot 45^\circ &= \frac{AB}{AC} = 1.\end{aligned}$$

Ta tính trong Python như sau:⁸

```
1 >>> print(math.sin(math.radians(45)), math.cos(math.pi/4), 1 /
   ↪ 2**0.5)
0.7071067811865475 0.7071067811865476 0.7071067811865475
2 >>> math.tan(math.radians(45)); 1 / _
0.9999999999999999
1.0000000000000002
```

⁷Ta đã biết rằng dấu chấm cũng được dùng làm “dấu chấm thập phân” trong số thực. Cũng vậy, dựa vào ngữ cảnh mà Python phân giải việc lạm dụng kí hiệu này.

⁸Bạn đã phải nạp module `math` (lệnh `import math`) trước khi dùng các lệnh.

Như bạn thấy, các hàm `sin`, `cos`, `tan` (của `math`) giúp tính các tỉ số lượng giác tương ứng. Lưu ý là Python dùng **radian** để đo góc chứ không dùng **độ** (degree).⁹ Do đó ta cần đổi độ sang radian trước bằng hàm `radians`. Minh họa trên cũng cho thấy cách dùng số π bằng tên biến `pi` trong module `math` (nhớ là $45^\circ = \frac{\pi}{4}$ radian). Hơn nữa, Python dùng tên hàm `tan` (chứ không phải `tg`) và không hỗ trợ hàm tính `cotg` nhưng ta có thể dễ dàng tính được vì `cotg` là nghịch đảo của `tg`. Cũng lưu ý về sai số của số thực nên các kết quả xuất ra ở trên “không hoàn hảo”.

Minh họa có ích hơn sau đây giúp ta “giải tam giác” OPQ trong Hình (c) trên (nghĩa là xác định chiều dài các cạnh và số đo các góc của tam giác).

```

1 >>> P = math.radians(36); PQ = 7
2 >>> OP = PQ * math.cos(P); OQ = PQ * math.sin(P)
3 >>> print(round(OP, 3), round(OQ, 3), (OP**2 + OQ**2)**0.5)
5.663 4.114 7.0
4 >>> Q = math.degrees(math.asin(OP/PQ))
5 >>> print(round(Q), round(Q, 2), Q + math.degrees(P))
54 54.0 90.0

```

Sau khi đặt các giá trị như đề cho (Dòng 1), vì $\cos \widehat{P} = \frac{OP}{PQ}$ nên ta có $OP = PQ \cos \widehat{P}$, tương tự ta có $OQ = PQ \sin \widehat{P}$ (Dòng 2). Tính ra ta được $OP \approx 5.663$, $OQ \approx 4.114$, hơn nữa, dùng định lý Pytago ta kiểm lại với cạnh $PQ = 7$ (Dòng 3). Vì tổng 3 góc của một tam giác là 180° nên cách tính \widehat{Q} nhanh nhất là $90^\circ - \widehat{P}$. Tuy nhiên, Python hỗ trợ “tính ngược” số đo góc khi biết tỉ số lượng giác của nó. Chẳng hạn ta đã biết $\sin \widehat{Q} = \frac{OP}{PQ}$, nên từ đó tính ngược ra số đo góc \widehat{Q} bằng hàm `asin`.¹⁰ Như đã lưu ý trên, Python trả về số đo góc theo radian nên ta chuyển nó sang độ bằng hàm `degrees` (Dòng 4). Kết quả ta có $\widehat{Q} = 54^\circ$ như kết xuất của Dòng 5 cho thấy (ta cũng đã kiểm lại tổng của \widehat{Q} và \widehat{P} là 90°).

Thật ra bạn cần biết thêm rằng không phải module (gói dịch vụ) nào cũng có thể dùng như `math`. Có những module mà bạn phải tốn tiền để mua (hoặc người ta cho miễn phí) và bạn phải **cài đặt** (install) thêm vào Python trước khi nạp và dùng. Các module loại này được gọi là **module ngoài** (third-party module)¹¹ để phân biệt với các module như `math` là các module được cung cấp sẵn của Python (bạn không phải cài đặt thêm, nó đi cùng Python). Tập các module cung cấp sẵn này được gọi là **thư viện chuẩn Python** (Python Standard Library). Học lập trình Python, ngoài việc học ngôn ngữ Python, là việc học dùng các hàm trong các module này. Nguyên lý cơ bản là tận dụng tối đa các toán tử, sau đó là các hàm dựng sẵn, sau đó là các hàm trong các module của thư viện chuẩn, sau đó là các hàm trong các module, thư viện ngoài phù hợp khác. Sau đó nữa? Tự viết lấy.¹² Bạn sẽ dùng các module trong

⁹Radian là đơn vị chuẩn để đo góc. Nếu không biết về radian, bạn cứ hình dung nó là một đơn vị khác (ngoài đơn vị độ hay dùng).

¹⁰`asin` là viết tắt của arc sine. Tương tự, ta có thể dùng các hàm `acos` và `atan` để tính ngược số đo góc từ tỉ số lượng giác cos và tg.

¹¹Dịch sát nghĩa là “module của bên thứ 3”. Bạn có biết bên thứ nhất và thứ 2 là ai không?

¹²Không cho thì ta phải tự làm thôi!

thư viện chuẩn (và các thư viện ngoài nổi tiếng khác) và tự viết lấy hàm, module, thư viện trong các bài sau.

3.3 Từ khóa

Trong lệnh nạp module ở trên thì `import` là một **từ khóa** (keyword). Từ khóa là từ mang nghĩa đặc biệt, được dùng như những khẩu hiệu hay tín hiệu cho biết loại lệnh, toán tử, ... Để biết danh sách các từ khóa của Python,¹³ bạn có thể dùng lệnh tra cứu sau:

```
1 >>> help("keywords")
Here is a list of the Python keywords.
Enter any keyword to get more help.

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

Trong danh sách từ khóa này, bạn đã quen thuộc với `import`, `True` và `False`. Bạn sẽ học các từ khóa khác sau nhưng cần nhớ là *không được đặt tên trùng với từ khóa*. Hơn nữa, mặc dù có thể, nhưng bạn cũng *không nên đặt tên trùng với tên các hàm, module dựng sẵn*. Thử minh họa sau:¹⁴

```
1 >>> True = 1
SyntaxError: cannot assign to True
2 >>> print
<built-in function print>
3 >>> print(True)
True
4 >>> print = 1; print
1
5 >>> print(True)
...
TypeError: 'int' object is not callable
```

¹³Danh sách này có thể thay đổi qua các phiên bản Python. Chẳng hạn `print` là từ khóa trong Python 2 nhưng không là từ khóa trong Python 3 (mà là tên hàm dựng sẵn).

¹⁴Tôi sẽ dùng dấu 3 chấm (...) tại những kết xuất không cần thiết như trong kết xuất của Dòng 5. Khi bạn chạy sẽ có kết xuất chi tiết hơn. Đây là một “thủ đoạn” khác để tiết kiệm giấy!

- Dòng 1: lỗi cú pháp vì không được phép đặt tên trùng với từ khóa.
- Dòng 2: `print` là tên tham chiếu đến một hàm dựng sẵn (hàm xuất).
- Dòng 3: dùng `print` và `True` như bình thường.
- Dòng 4: `print` được định nghĩa lại để tham chiếu đến giá trị 1. Nó không còn là tên hàm mà là tên biến!
- Dòng 5: lỗi thực thi `TypeError`, “**lỗi không đúng kiểu**”, do dùng `print` như là một hàm trong khi nó tham chiếu đến số nguyên 1 (ta sẽ tìm hiểu thêm lỗi này sau).

3.4 Chế độ tương tác và phiên làm việc

Ta đã thấy Python cho phép định nghĩa các tên mà sau đó có thể dùng lại, hay ta có thể chỉ nạp một lần module nào đó mà sau đó có thể dùng, hay các hàm dựng sẵn có thể được truy cập bằng tên hàm. Rõ ràng, Python phải có cách nào đó để nhớ những thứ này. Cũng tự nhiên khi Python dùng “**bộ nhớ**” (memory) của nó để nhớ.¹⁵ Ta sẽ tìm hiểu khái niệm cực kì quan trọng này sau (xem Phần ??), ở đây, ta “nghĩa” sơ nó thôi.¹⁶

```

1 >>> dir()
  ['__annotations__', '__builtins__', '__doc__', '__loader__',
  '__name__', '__package__', '__spec__']
2 >>> a = 10; b = 2.5; c = b
3 >>> import math
4 >>> dir()
  [..., 'a', 'b', 'c', 'math']
5 >>> dir(__builtins__)
  [..., 'NameError', 'SyntaxError', ..., 'abs', 'print', ...]
```

Hàm dựng sẵn `dir` cho phép ta “kiểm tra bộ nhớ của Python”. Như ta thấy, lúc mới khởi động (bắt đầu làm việc) thì Python đã biết sẵn một vài thứ,¹⁷ trong đó có `__builtins__` chứa các hàm dựng sẵn.¹⁸ Sau khi ta định nghĩa 3 biến `a`, `b`, `c` và nạp module `math` thì “bộ nhớ Python” có thêm chúng như kết xuất cho thấy.

Bạn có thể tưởng tượng rằng từ khi mới “sinh ra” (tức là ngay sau khi chạy IDLE)¹⁹, Python “biết sẵn” về các hàm (và các thứ khác) dựng sẵn. Sau đó, trong quá trình “sống” và thực thi các lệnh, Python có thể biết (nhớ) thêm nhiều thứ khác như các tên (biến, hàm, module, ...) được định nghĩa. Cũng trong quá trình đó, nếu

¹⁵Bộ nhớ là cốt lõi của mọi hệ “có trạng thái”, không nhớ thì không hoạt động được.

¹⁶Bạn cần chạy minh họa này từ đầu hoặc tắt và mở lại IDLE rồi chạy.

¹⁷Python qui ước dùng các tên đặc biệt `__xyz__` cho các mục đích đặc biệt. Ta cũng nên tránh đặt tên bắt đầu bằng dấu gạch dưới (`_`).

¹⁸Bạn có thể hình dung `__builtins__` là module chứa các hàm dựng sẵn (như `print`, `abs`, ...) mà Python sẽ tự động nạp module này khi bắt đầu làm việc.

¹⁹Thật ra IDLE chỉ là một “vỏ bọc”, cái quan trọng là động cơ Python.

Python dừng (dùng) đến các tên thì nó sẽ lôi ra từ bộ nhớ hay báo lỗi nếu chưa có (tức chưa biết). Khi kết thúc (“chết đi”, tức IDLE bị đóng lại), thì Python hoàn thành sứ mạng (làm đầy tớ) của nó. Ta gọi đây là một **phiên làm việc** (session) của Python và mô tả ở trên là cho một phiên làm việc thông thường.

Phiên làm việc của Python cũng có thể kết thúc “bất ngờ” theo nhiều cách, chẳng hạn, trên IDLE ta có thể kết thúc phiên làm việc hiện hành và khởi động phiên làm việc mới bằng Shell → Restart Shell (Ctrl+F6)²⁰ như minh họa sau:

```

1 >>> a = 10; a
10
>>>
===== RESTART: Shell =====
2 >>> a
...
NameError: name 'a' is not defined

```

Ta định nghĩa biến a và sau đó dùng nó như thông thường. Sau khi khởi động lại, Python kết thúc phiên làm việc cũ và bắt đầu phiên làm việc mới, do đó Python không còn nhớ biến a nữa như thông báo lỗi thực thi NameError cho thấy.

Trong phiên làm việc của mình, Python đợi ta nhập lệnh (thông báo bằng dấu đợi lệnh >>>); sau khi ta nhập lệnh (và nhấn Enter), Python đọc lệnh và phân tích; nếu lệnh nhập không đúng (cú pháp), Python thông báo lỗi cú pháp và tiếp tục đợi lệnh mới; nếu không có lỗi cú pháp, Python thực thi (mà trường hợp lệnh là biểu thức thì Python lượng giá); nếu lệnh yêu cầu xuất kết quả (hoặc lệnh là biểu thức) thì Python xuất kết quả, ngược lại thì không; sau đó, Python lại tiếp tục đợi lệnh kế tiếp. Qui trình lặp lại này được gọi là vòng lặp **Đọc-Thực thi/Lượng giá-Xuất** (Read-Execute/Evaluate-Print loop, viết tắt REPL) và cách thức hoạt động này của Python được gọi là **chế độ tương tác** (interactive mode).

3.5 Tra cứu

Ta đã thấy cách tốt nhất để biết về danh sách các từ khóa của Python là **tra cứu** (search/help), nghĩa là dò tìm thông tin trong một nguồn chi tiết, đầy đủ, khổng lồ.²¹ Cũng như hầu hết các cuốn sách khác, *tài liệu này không phải là tài liệu tra cứu* do các giới hạn về dung lượng và hình thức. Hơn nữa, mục đích chính của tài liệu này là giúp bạn nắm các nguyên lý, kỹ thuật cơ bản và trải nghiệm việc lập trình. Bạn cần một nguồn động (luôn cập nhật), đầy đủ và chi tiết kỹ thuật để tra cứu. Nguồn tốt nhất như vậy chính là Web (qua Internet) và các biến thể của nó.

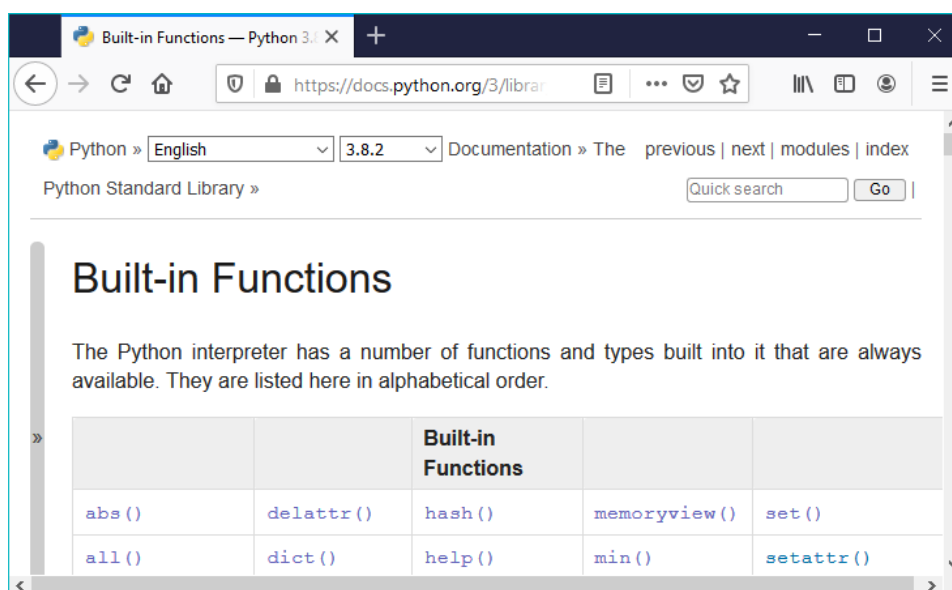
Bạn có thể tra cứu bằng hệ thống tra cứu nội tại của Python với hàm dựng sẵn `help`. Chẳng hạn, bạn đã dùng `help("keywords")` để tra danh sách các từ

²⁰Chức năng này tương tự phím ON/OFF của calculator. Dĩ nhiên, thô bạo hơn, ta có thể bắt đầu phiên làm việc mới bằng cách tắt và chạy lại IDLE.

²¹Thuật ngữ rộng hơn là “**tìm kiếm thông tin**”. Tuy nhiên, ở đây, ta chỉ dùng trường hợp hẹp của nó là tra cứu thông tin, tương tự việc tra từ điển.

khóa, bạn cũng có thể dùng `help()` để được hướng dẫn sử dụng hàm `help`, hay `help(print)` để tra cứu hàm `print`, `help("math")` tra cứu module `math`, ... Tuy nhiên, do hạn chế của IDLE mà cách này không tiện lợi lắm.

Cách khác là dùng hệ thống **tài liệu** (documentation) của Python mà đơn giản nhất là tài liệu cục bộ bằng Help → Python Docs (F1) hay hệ thống Python Docs trực tuyến (<https://docs.python.org/3/>). Tuy nhiên, việc dò tìm trong đồng tài liệu này cũng rất khó khăn. Cách tốt hơn là search Google với cụm từ “Python Docs” đằng trước.²² Chẳng hạn, để tra cứu các hàm dựng sẵn, bạn có thể search Google “Python docs built-in functions”. Trong kết quả tìm kiếm đầu tiên, ta thấy trang Python Docs nhưng cụ thể theo mục cần tìm là hàm dựng sẵn (<https://docs.python.org/3/library/functions.html>) như hình dưới.



Để đọc hiểu các tài liệu tra cứu này, bạn cần biết Tiếng Anh cơ bản và quen thuộc với các thuật ngữ. Chẳng hạn, tìm và nhấp vào link `print()` (<https://docs.python.org/3/library/functions.html#print>) trong bảng trên, từ trang hiện ra, ta thấy rằng ngoài 2 đối số có tên là `sep` và `end` thì `print` còn có 2 cái nữa là `end` và `flush`. Các đối số này nhận các giá trị mặc định và ý nghĩa như thông tin cho biết. Cũng trong trang đó, ta biết các **đối số không tên** (non-keyword argument) sẽ được chuyển thành chuỗi bằng hàm dựng sẵn `str` trước khi in ra. Mà hàm này, nếu muốn, ta có thể tìm hiểu kĩ hơn bằng cách nhấp vào link `str()` (<https://docs.python.org/3/library/stdtypes.html#str>) trong đoạn giải thích. Từ trang mới hiện ra, ta đọc thấy ... Tóm lại là chẳng hiểu bao nhiêu!

Ngoài khó khăn là Tiếng Anh thì tài liệu tra cứu còn dùng nhiều thuật ngữ kĩ thuật mà bạn chưa quen thuộc.²³ Tuy nhiên, suy cho cùng, *nghệ thuật tra cứu* là

²²Hay “Python”, “Python 3”, “Python 3 docs”, ... để định hướng việc tìm kiếm.

²³Để ý là tôi luôn ghi kèm thuật ngữ Tiếng Anh khi trình bày khái niệm mới trong tài liệu này.

nghệ thuật “tìm vàng trong cát”, bạn cần lược qua được các thứ không cần thiết để tìm được đúng nội dung mình cần. Rõ ràng, đây không phải là một kỹ năng dễ dàng, nhưng có thể rèn luyện được và bạn cần (từ từ) rèn luyện vì nó là kỹ năng “mềm” rất quan trọng.

Cách tra cứu tự do và linh hoạt hơn là dùng thông tin của toàn mạng Internet. Chẳng hạn, ta có thể tra hàm `print` bằng cách search Google “Python print”. Kết quả tìm kiếm cho ra nhiều trang “không chính thống”.²⁴ Các trang tốt là W3Schools (<https://www.w3schools.com/>), Stack Overflow (<https://stackoverflow.com/>), CodeProject (<https://www.codeproject.com/>), ... Bạn cũng có thể tìm các website, course, forum, blog, group facebook, ... để học và tra cứu.

Như một bài tập, bạn hãy tra cứu để biết sơ về phong cách viết PEP 8 (xem lại Phần 2.5). Nguồn chính thống là <https://www.python.org/dev/peps/pep-0008/> nhưng nếu bạn hơi “choáng” thì có thể xem ở <https://docs.python.org/3/tutorial/controlflow.html#intermezzo-coding-style> hoặc các nguồn không chính thống khác. Nhớ rằng, bạn chưa biết gì nhiều nên chỉ nghĩa sơ thôi. Bạn sẽ trở lại tham khảo và tra cứu thường xuyên các nguồn tài liệu này để có một phong cách viết tốt.

Tóm tắt

- Python cung cấp nhiều dịch vụ (mà hay dùng là các dịch vụ tính toán) qua các hàm dựng sẵn như: tính trị tuyệt đối (`abs`), làm tròn (`round`), tìm giá trị lớn nhất (`max`), tìm giá trị nhỏ nhất (`min`), tính mũ (`pow`), xuất (`print`). Không nên đặt tên trùng với tên các hàm dựng sẵn.
- Để dùng hàm, ta xác định tên hàm và cung cấp các đối số phù hợp qua lời gọi hàm. Hàm có thể có đối số có tên và nhận giá trị mặc định. Hàm cũng có thể trả về giá trị và tham gia vào biểu thức.
- Python cung cấp rất nhiều hàm khác nhau trong các gói dịch vụ, gọi là module. Tập các module gọi là thư viện. Thư viện chuẩn Python được cung cấp sẵn khi cài đặt Python. Ta cần dùng lệnh `import` để nạp module trước khi dùng.
- Module `math` (trong thư viện chuẩn) cung cấp các hàm (và giá trị) tính toán như: tính giai thừa (`math.factorial`), tính sin (`math.sin`), tính ngược số đo góc từ sin (`math.asin`), đổi độ sang radian (`math.radians`), ...
- Python có các từ dùng với mục đích đặc biệt gọi là từ khóa (như `import` dùng trong lệnh nạp module). Có thể xem danh sách các từ khóa bằng lệnh `help("keywords")`. Không được đặt tên trùng với từ khóa.

²⁴Tôi gọi các trang Python Docs là **chính thống** (official) vì tác giả của các tài liệu này là cha đẻ của Python và “**tổ chức nền tảng Python**” (Python Software Foundation, viết tắt PFS) là tổ chức “quản lý” Python.

- Python hoạt động, tương tác và ghi nhớ theo phiên làm việc. Cách hoạt động trong các minh họa cho đến giờ là theo chế độ tương tác, trong đó, công việc của Python là thực hiện liên tục vòng lặp REP.
- Python ghi nhớ các tên đã định nghĩa (hay các module đã nạp, ...) bằng “bộ nhớ”. Có thể kiểm tra bộ nhớ này bằng hàm dựng sẵn `dir`.
- Tra cứu là kỹ năng mềm quan trọng mà bạn cần thường xuyên rèn luyện. Nguồn tra cứu tốt là Python Docs, W3Schools, Stack Overflow, ... Cách tra cứu nhanh là search Google với cụm từ “Python Docs”, “Python 3” hay “Python” trước từ khóa tra cứu. Để tra cứu và hiểu nội dung, bạn cần kỹ năng đọc Tiếng Anh và quen thuộc với các thuật ngữ Python hay dùng.

Bài tập

3.1 Dùng Python,²⁵ chứng minh các đẳng thức sau:²⁶

$$(a) \left(\frac{2\sqrt{3}-\sqrt{6}}{\sqrt{8}-2} - \frac{\sqrt{216}}{3} \right) \cdot \frac{1}{\sqrt{6}} = -1,5$$

$$(b) \left(\frac{\sqrt{14}-\sqrt{7}}{1-\sqrt{2}} + \frac{\sqrt{15}-\sqrt{5}}{1-\sqrt{3}} \right) : \frac{1}{\sqrt{7}-\sqrt{5}} = -2$$

Lưu ý: module `math` cung cấp hàm `isclose` giúp kiểm tra xem 2 số thực có đủ gần nhau không. Hàm này ngoài nhận 2 số cần kiểm tra thì nhận thêm một đối số có tên là `abs_tol` xác định ngưỡng khoảng cách được xem là “gần”. Chẳng hạn, việc kiểm tra đẳng thức $(\sqrt{2})^2 = 2$ trong bài học có thể được viết gọn như sau:

```
1 >>> import math
2 >>> math.isclose((2**0.5)**2, 2, abs_tol=1e-9)
True
```

Bạn hãy tận dụng hàm `isclose` để làm bài tập này cùng với cách làm “thủ công” như trong bài học.

3.2 Dùng các hàm lượng giác trong module `math`, thực hiện các yêu cầu sau:²⁷

$$(a) \text{ Tính: } \frac{\sin 25^\circ}{\cos 65^\circ}, \tan 58^\circ - \cot 32^\circ$$

$$(b) \text{ Tính các tỉ số lượng giác sau (làm tròn đến chữ số thập phân thứ 4): } \sin 70^\circ 15', \cos 25^\circ 32', \tan 43^\circ 10', \cot 32^\circ 15'.$$

$$(c) \text{ Tìm góc nhọn } x \text{ (làm tròn kết quả đến độ) biết rằng: } \sin x = 0,3495; \cos x = 0,5427; \tan x = 1,5142; \cot x = 3,163.$$

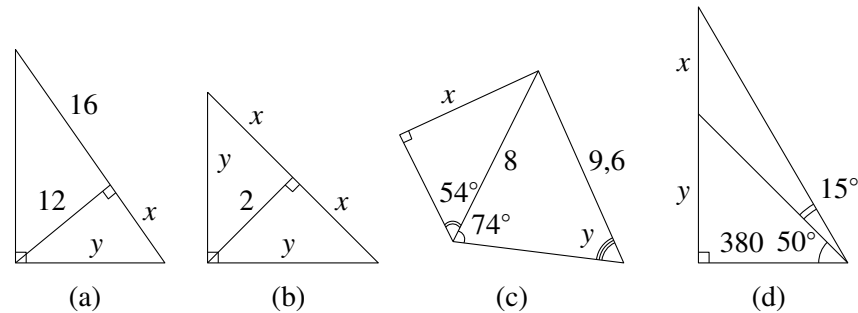
3.3 Tìm x và y trong mỗi hình sau:²⁸

²⁵Tất cả các bài tập trong bài học này đều dùng Python để làm!

²⁶SGK Toán 9 Tập 1. Tôi viết lại như cách SGK đã viết nhưng bạn cần phải dùng các kí hiệu phù hợp trong Python.

²⁷SGK Toán 9 Tập 1.

²⁸Chỉnh sửa từ SGK Toán 9 Tập 1.



3.4 Khai phương. Tính căn bậc 2, còn được gọi là **phép khai phương** (square root), là một phép toán rất hay được dùng mà bạn đã biết cách tính bằng hàm dựng sẵn `pow` hay toán tử mũ (`**`). Module `math` cũng hỗ trợ hàm `sqrt` để khai phương. Hơn nữa, `math` cũng hỗ trợ hàm `pow` để tính mũ.²⁹ Khai phương các số sau bằng 4 cách (trong Python):

(a) 0.0196 (b) 1.21 (c) 2 (d) 3 (e) 4 (f) $\frac{225}{256}$

3.5 Làm tròn đến một chữ số. Làm tròn (rounding) một số là đưa ra “biểu diễn đơn giản hơn” cho số đó. Kết quả làm tròn thường chỉ xấp xỉ giá trị ban đầu, tức là có **sai số** (error). Làm tròn số thực thành số nguyên là công việc hay làm và Python hỗ trợ nó với nhiều hàm khác nhau. Ta đã dùng hàm dựng sẵn `round` để **làm tròn đến số nguyên gần nhất** (rounding to the nearest integer) mà trong trường hợp phần lẻ là 0.5 (có 2 số nguyên gần nhất) thì **làm tròn nửa đến số chẵn** (round half to even) như minh họa sau:

```
1 >>> print(round(1.49), round(1.51))
    1 2
2 >>> print(round(1.5), round(2.5))
    2 2
3 >>> print(round(-1.5), round(-2.5))
   -2 -2
```

Python cũng hỗ trợ 2 cách làm tròn khác là **làm tròn xuống** (rounding down) và **làm tròn lên** (rounding up) với hàm `floor` và `ceil` trong module `math`. Thử minh họa sau:

```
1 >>> import math
2 >>> print(math.floor(1.49), math.floor(1.51))
    1 1
3 >>> print(math.ceil(1.49), math.ceil(1.51))
    2 2
4 >>> print(math.floor(-1.51), math.ceil(-1.49))
   -2 -1
```

(a) Tra cứu hàm `round`, `math.floor` và `math.ceil`.

²⁹ Bạn nên dùng hàm dựng sẵn `pow` hoặc toán tử mũ (`**`) để tính mũ hơn là hàm `pow` của `math`.

- (b) Làm tròn các số sau theo 3 cách làm tròn trên: $\frac{10}{3}$, $\frac{10}{4}$, $\sqrt{2}$, π , e ,³⁰ ϕ .³¹
- (c) Cũng Câu (b) nhưng làm tròn đến chữ số thứ: 1, 3, 6, 9 sau dấu chấm thập phân. *Gợi ý:* có thể nhân 10 hay chia 10 để “dời” dấu chấm thập phân.
- (d) Giả sử giá trị làm tròn đến chữ số thứ 9 sau dấu chấm thập phân theo cách làm tròn gần nhất được dùng làm giá trị “đại diện” cho các số trên. Hãy tính sai số (tức là khoảng cách giữa giá trị làm tròn và giá trị “đại diện”) trong các cách làm tròn lên và xuống đến các chữ số như trong Câu (c).
- (e) Tính phần lẻ của giá trị đại diện của các số trên ở Câu (d). Chẳng hạn, giá trị đại diện (làm tròn gần nhất đến 9 chữ số sau dấu chấm thập phân) của số π là 3.141592654 nên phần lẻ là 0.141592654.

Lưu ý: bạn phải dùng Python để tính nhé.

3.6 Làm tròn bằng phân số. Một cách “làm tròn” khác là dùng phân số để xấp xỉ. Hàm `Fraction` trong module `fractions` giúp tìm phân số xấp xỉ này.³² Hơn nữa, hàm dựng sẵn `float` giúp tìm số thực (gần đúng) tương ứng với một phân số.³³ Thử minh họa sau:³⁴

```
1 >>> import fractions
2 >>> a = fractions.Fraction(1.25); a; print(a)
Fraction(5, 4)
5/4
3 >>> b = a + 1; b; float(b)
Fraction(9, 4)
2.25
```

Như vậy, ta có một loại số trong Python là **phân số** (fraction) hay **số hữu tỉ** (rational number). Trong minh họa trên, `a` trở đến số hữu tỉ có tử là 5 và mẫu là 4 (tức là phân số $\frac{5}{4} = 1.25$). Ta cũng biết thêm là hàm `print` xuất ra “biểu diễn đẹp” của một giá trị, ở trên, phân số có tử là 5 và mẫu là 4 (tức là `Fraction(5, 4)`) có biểu diễn đẹp là `5/4`.³⁵

Làm tròn các số trong Bài tập 3.5b bằng phân số và tính sai số (so với giá trị đại diện ở Bài tập 3.5d).

3.7 Định dạng. Bên cạnh việc làm tròn, ta cũng thường muốn xuất ra (hiển thị, in ấn, báo cáo) các số thực với **số chữ số cố định sau phần thập phân** (fixed-

³⁰Số e được gọi là số Euler, là một hằng số quan trọng khác sau π . Đây là số vô tỉ có giá trị khoảng 2.71828. Trong Python, `math.e` kí hiệu cho giá trị này (dĩ nhiên là giá trị xấp xỉ).

³¹Số ϕ (đọc là phi) được gọi là **tỉ lệ vàng** (golden ratio), là một hằng số quan trọng trong tự nhiên và nghệ thuật. Đây là số vô tỉ có giá trị $\frac{1+\sqrt{5}}{2}$.

³²Đến lúc này, bạn phải tạo thành thói quen tra cứu. Khi gặp một hàm hay module nào mới, bạn phải tập tra cứu nó. Bạn không thể hiểu hết (đúng hơn là chưa hiểu gì nhiều) nhưng phải nghĩa qua nó. Nhớ nghe!!!

³³Tra liền nè!

³⁴Một số calculator cũng có chức năng “chuyển đổi” dạng thập phân và phân số.

³⁵Hiển nhiên, bạn không thể đòi hỏi Python xuất ra là $\frac{5}{4}$ vì Python chỉ dùng các kí tự khi xuất (với hàm `print`). Bạn, thực ra, đã biết vụ “biểu diễn đẹp” này rồi đó, thử các lệnh `s = "Hello"; s; print(s)` và quan sát kết quả.

point notation) (thường là 3, 6, 9 chữ số). Python cung cấp hàm dựng sẵn `format` giúp thực hiện chức năng **định dạng** (format) này. Thử minh họa sau:

```
1 >>> print(round(1.5, 3), round(1.2999, 3))
1.5 1.3
2 >>> format(1.5, ".3f"); print(format(1.2999, ".3f"))
'1.500'
1.300
3 >>> print(format(0.75), format(0.75, "%"), format(0.75,
↪ ".0%"))
0.75 75.000000% 75%
```

Hàm `format` nhận đối số đầu là giá trị cần định dạng và đối số thứ 2 là **chuỗi đặc tả định dạng** (format specifier) xác định cách giá trị được định dạng.³⁶ Hàm trả về chuỗi kết quả sau khi định dạng.

Xuất các số trong Bài tập 3.5b theo định dạng cố định 3, 6, 9 chữ số sau dấu chấm thập phân.

Nhân tiện, bạn cũng có thể định dạng các số nguyên lớn theo nhóm 3 chữ số phân cách bởi dấu phẩy (,) hoặc dấu gạch dưới (_) bằng chuỗi đặc tả định dạng tương ứng như minh họa sau:

```
1 >>> format(1_000_000_00, ",")
'1,000,000,000'
2 >>> format(10**9, ","); print(format(10**9, "_"))
'1,000,000,000'
1_000_000_000
```

3.8 Số thập phân có độ chính xác cố định. Ta đã thấy hạn chế của Python khi làm việc với số thực. Trong trường hợp ta chỉ làm việc với các số thập phân có **độ chính xác cố định** (fixed-precision, fixed-point), nghĩa là cố định số lượng chữ số sau dấu chấm thập phân, thì Python hỗ trợ tốt hơn qua kiểu số `decimal.Decimal`. Các số dạng này được dùng nhiều trong thực tế như mô tả điểm số (2 chữ số thập phân), đo lường (3 hay 6 chữ số), ... và đặc biệt là trong tài chính, tiền tệ. Minh họa sau mô tả rõ hơn:

```
1 >>> 3 * 0.1 == 0.3
False
2 >>> from decimal import Decimal
3 >>> 3 * Decimal("0.1") == Decimal("0.3")
True
4 >>> print(Decimal(0.1), Decimal("0.1"))
0.1000000000000000055511151231257827021181583404541015625 0.1
5 >>> x, y = Decimal("0.1"), Decimal("0.30"); print(x+y, x*y)
```

³⁶Bạn nắm vài chuỗi đặc tả định dạng thông dụng như minh họa trên và các minh họa sẽ gặp. Bạn cũng có thể tra cứu để biết nhiều hơn, chẳng hạn, tại <https://docs.python.org/3/library/string.html#formatspec>.

còn giúp tìm phần nguyên của một phân số mà từ đó ta có thể tìm dạng **hỗn số** (mixed number).

Dùng số Fraction (và số nguyên) tính:³⁸

- (a) $10\frac{3}{7} : 5\frac{3}{14}$
- (b) $\frac{5^4 \cdot 20^4}{25^5 \cdot 4^5}$
- (c) $\left(1 + \frac{2}{3} - \frac{1}{4}\right) \cdot \left(\frac{4}{5} - \frac{3}{4}\right)^2$
- (d) $(-0,375) \cdot 4\frac{1}{3} \cdot (-2)^3$

3.10 Toán tử là hàm. Trong Phần 3.1 tôi đã nói nhiều về khác biệt (và tương đồng) giữa toán tử và hàm mà tổng kết lại “toán tử là hàm được Python hỗ trợ viết đẹp!”. Thật vậy, module `operator` cung cấp các hàm tương ứng với tất cả các toán tử mà Python có (số học, so sánh và các toán tử sẽ học khác). Thử minh họa sau:

```
1 >>> import operator as op
2 >>> print(2 ** 0.5, op.pow(2, 0.5))
1.4142135623730951 1.4142135623730951
3 >>> print(5/2, op.truediv(5, 2), 5//2, op.floordiv(5, 2))
2.5 2.5 2 2
4 >>> print(-(5 - 2), op.neg(op.sub(5, 2)))
-3 -3
5 >>> print(2.0 == 2*1, op.eq(2.0, 2*1))
True True
```

Minh họa cũng cho thấy cách nạp module đặc biệt để “đặt lại tên” cho các module có tên dài (tên `operator` được đặt lại là `op`).

Tương tự, viết lại các minh họa trong Bài 2 bằng cách dùng các hàm trong module `operator` thay cho toán tử.

3.11 Đọc IDLE, thử các chức năng và tra cứu sau:

- Xem thông tin IDLE: Help → About IDLE, trong hộp thoại “About IDLE” đọc thông tin (cũng nhấn các nút như License, ... rồi đọc).³⁹
- Xem hướng dẫn sử dụng IDLE: Help → IDLE Help, trong hộp thoại “IDLE Help”, đọc lướt các chức năng và cách sử dụng (cũng như phím tắt, ...). Cái nào hay, cần thiết thì thử nghiệm và nhớ, còn lại bỏ qua.
- Đọc hệ thống Python Docs cục bộ: Help → Python Docs (F1), trong hộp thoại “Python documentation” đọc qua các nút, các link, search, ... Thử tìm kiếm vài thông tin như các hàm dựng sẵn và module `math`, `decimal`, `fractions`, `operator`.
- Đọc hệ thống tra cứu nội tại của Python: trong Python Shell, dùng hàm `help` thử tìm kiếm vài thông tin như các hàm dựng sẵn, từ khóa và module `math`, `decimal`, `fractions`, `operator`.

³⁸ SGK Toán 7 Tập 1.

³⁹ Bạn không phải nhớ nhưng nghĩa sơ cho biết.

Bài 4

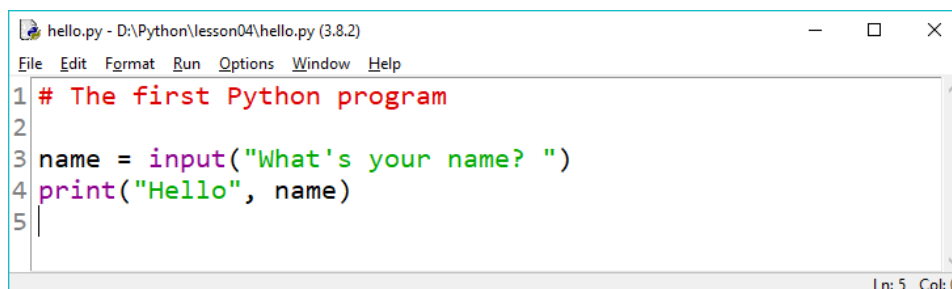
Bắt đầu lập trình

Trong các bài trước, ta chủ yếu dùng Python như một calculator. Ta đã nhờ Python thực hiện các yêu cầu tính toán mà chủ yếu là tính toán số học. Python thực sự mạnh hơn như vậy. Ta có thể nhờ Python làm rất nhiều việc, vượt xa một calculator thông thường, khi ta lập trình với nó. Hãy bắt đầu lập trình với Python từ bài này.

4.1 Chương trình và mã nguồn

Để yêu cầu Python làm các công việc lớn, ta thường phải dùng nhiều lệnh vì các công việc này thường phức tạp, gồm nhiều việc nhỏ, nhiều công đoạn. Hệ quả là ta cần lưu trữ các lệnh này lại trong một tập tin và yêu cầu Python thực thi cả tập tin. Ta thường gọi tập tin này là **tập tin mã nguồn Python** (Python source code file), đây các lệnh trong nó là **mã nguồn Python** (Python source code) (hay gọi tắt là mã) và công việc nó mô tả là **chương trình Python** (Python program).¹

Với IDLE, ta dùng chức năng File → New File (Ctrl+N) để tạo tập tin mã nguồn. Trong cửa sổ tập tin, ta gõ mã nguồn, tức là dãy lệnh, mỗi lệnh trên một dòng. Sau khi gõ mã (chẳng hạn như minh họa dưới đây), ta lưu lại bằng File → Save (Ctrl+S). Ta cũng nên tổ chức thư mục và đặt tên tập tin cho phù hợp. Ở đây, tôi lưu lại với tên tập tin là `hello.py` (đuôi `py` là mặc định cho các file mã Python) đặt ở thư mục `D:\Python\lesson04`.



```
hello.py - D:\Python\lesson04\hello.py (3.8.2)
File Edit Format Run Options Window Help
1 # The first Python program
2
3 name = input("What's your name? ")
4 print("Hello", name)
5
Ln: 5 Col: 0
```

¹Thuật ngữ dãy lệnh nhấn mạnh đến thứ tự các lệnh. Ta đã thấy tầm quan trọng của thứ tự trong Phần 2.2.

Để yêu cầu Python **thực thi** (execute) (hay gọi là **chạy** (run)) file mã, ta dùng chức năng Run → Run Module (F5) trong IDLE. Khi đó, Python khởi động lại và thực thi từng lệnh trong file mã. Với mã trên, Python sẽ đợi ta gõ một cái tên (chẳng hạn, gõ Guido van Rossum rồi nhấn phím Enter) và xuất ra lời chào đến tên đó như hình dưới.

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29)
[MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
===== RESTART: D:\Python\lesson04\hello.py
=====
What's your name? Guido van Rossum
Hello Guido van Rossum
>>> |
```

Để IDLE đánh số dòng lệnh cho dễ theo dõi, ta dùng Options → Configure IDLE và ở thẻ General trong hộp thoại Setting, ta bật lựa chọn “Show line numbers in new windows”.

Ta cũng có thể dùng Python trực tuyến để tạo, lưu và thực thi các chương trình Python. Search Google “online Python”, ta sẽ có nhiều công cụ như vậy, chẳng hạn, trang <https://repl.it/languages/python3>.² Bạn gõ chương trình trong main.py (bạn cũng có thể đặt lại tên) và nhấn nút Run để Python chạy chương trình. Bên Python Shell bạn gõ một cái tên và Python xuất ra lời chào đến tên đó như hình dưới.

Các mã nguồn Python được mô tả như sau trong tài liệu này. Tiêu đề của khung là link mã nguồn ở GitHub của tôi (<https://github.com/vqhBook/>). Bạn có

²Đây, đơn giản, là một trong những kết quả tìm được đầu tiên thôi.

thể dùng link này để mở file mã tương ứng. Tuy nhiên, trừ khi mã quá dài, bạn nên tự gõ để “động thủ”, giúp trải nghiệm và nắm bài tốt hơn. Cũng lưu ý, để tiện trình bày, mã viết ở đây có thể ngắn gọn, không đầy đủ và cập nhật như mã trong link.

<https://github.com/vqhBook/python/tree/master/lesson04/hello.py>

```
1 # The first Python program
2
3 name = input("What's your name? ")
4 print("Hello", name)
```

Kết quả khi thực thi chương trình (cùng với chuỗi nhập) được mô tả như sau:

```
What's your name? Guido van Rossum
Hello Guido van Rossum
```

Trong đó, chuỗi gạch dưới là chuỗi nhập (kết thúc với phím Enter).

Phần sau đây “mổ xẻ” chương trình Python đầu tiên này. Dòng 1 là dòng **ghi chú** (comment) được đánh dấu bởi kí hiệu # đầu dòng. Python không đọc dòng này cũng như tất cả các ghi chú khác. *Ghi chú chỉ có ý nghĩa với người viết và thường được dùng để mô tả, giải thích hoặc lưu ý thêm mục đích, ý nghĩa, ... của chương trình, lệnh, biến, ... cho chính người viết hoặc người khác đọc chứ không phải cho Python.*³ Python sẽ bỏ qua tất cả nội dung từ dấu # đến hết dòng nên ghi chú có thể để sau một lệnh và được viết bằng bất kì cách nào (Tiếng Anh, Tiếng Việt, kí hiệu Toán, ...). Chẳng hạn:

```
1 import math          # Nạp thư viện math
2 R = 10               # R là bán kính đường tròn (đơn vị cm)
3 S = math.pi * R * R # Diện tích hình tròn
```

Ta cũng không nên lạm dụng ghi chú, như ghi chú ở lệnh import trên nên được bỏ đi vì bản thân lệnh import đã có ý nghĩa rõ ràng là nạp thư viện tương ứng. Nói chung, *việc ghi chú (hay rộng hơn, sưu liệu) là một nghệ thuật!*

Dòng thứ 2 là dòng trống, mà mục đích là để trình bày đẹp: một chương trình Python thường có ghi chú ở đầu, sau đó là dòng trống rồi mới đến các lệnh thực sự. Cách trình bày này cũng nằm trong phong cách viết PEP 8.

Dòng thứ 3 dùng một hàm dựng sẵn là `input` giúp nhận thông tin từ người dùng chương trình. Hàm này nhận một đối số (tùy chọn) làm **thông báo nhập** (input prompt). Khi thực thi `input`, Python xuất ra thông báo nhập và đợi người dùng nhập vào một chuỗi, sau khi người dùng nhập chuỗi (gõ chuỗi và nhấn Enter) thì **chuỗi nhập** (input string) đó được dùng làm giá trị trả về của hàm. Lưu ý, *ta nên đưa ra thông báo nhập rõ ràng để người dùng biết họ cần nhập dữ liệu gì.*

Chuỗi nhập sau khi được gán cho biến `name` ở Dòng 3 thì được xuất ra trong lời gọi hàm `print` ở Dòng 4 sau chuỗi Hello. Kết quả, ta có thông báo Hello được xuất ra cho tên người dùng tương ứng.

³Ghi chú cũng hay được dùng để “bật/tắt” hoặc “viết nháp” các lệnh.

Chúc mừng!!! Bạn vừa viết và chạy thành công chương trình Python đầu tiên. Sau đây là những khái niệm và thuật ngữ quan trọng nhất mà bạn cần nắm. Một **chương trình máy tính** (computer program) là một dãy các **lệnh** (statement, instruction) yêu cầu máy tính thực thi một **công việc/tác vụ** (task) để giải quyết một **vấn đề/bài toán** (problem) nào đó. **Lập trình** (Python) là viết chương trình (Python) và **lập trình viên** (Python) là người viết chương trình (Python).

Ngoài Python, ta có thể viết chương trình bằng các **ngôn ngữ lập trình** (programming language) khác như C, C++, C#, Java, ... Tuy nhiên, như đã nói trong phần mở đầu, Python được xem là ngôn ngữ đơn giản, thông dụng, có tính quốc tế, tương tự như tiếng Anh trong **ngôn ngữ tự nhiên** (natural language) là những ngôn ngữ con người dùng để giao tiếp.

4.2 Chế độ chương trình và người dùng

Trong các bài trước, ta đã dùng Python theo chế độ tương tác với vòng lặp REP: ta nhập lệnh (bao gồm biểu thức), Python đọc (và phân tích) rồi thực thi (bao gồm lượng giá), sau đó xuất ra kết quả (nếu có). Chế độ này có tính tương tác cao (như tên gọi của nó) nhưng không phù hợp khi chương trình lớn. Với tập tin mã nguồn, Python thực thi liên tiếp các lệnh, từ đầu đến cuối và không xuất ra kết quả (trừ khi được yêu cầu tường minh bằng lệnh `print`). Cách thức thực thi này được gọi là **chế độ hàng loạt** (batch mode) hay **chế độ chương trình** (program mode).⁴

Với IDLE, khi thực thi một file mã, nó khởi động lại Python để tạo một phiên làm việc mới, thực thi file mã theo chế độ chương trình và vẫn giữ phiên làm việc sau khi kết thúc.⁵ Do đó, ta vẫn có thể tiếp tục tương tác với Python trong phiên làm việc đó (theo chế độ tương tác). Chẳng hạn, chạy file mã `hello.py` trên, gõ tên Python, chương trình xuất ra lời chào Python và kết thúc. Sau đó ta vẫn có thể dùng biến `name` (được định nghĩa bởi chương trình) như kết quả sau cho thấy:

```
===== RESTART: D:\Python\lesson04\hello.py =====
What's your name? Python
Hello Python
1 >>> print(name)
Python
```

Lưu ý, khi thực thi file mã, Python kiểm tra lỗi cú pháp trước; nếu mã có lỗi cú pháp thì Python sẽ báo lỗi và từ chối thực thi;⁶ ngược lại, Python thực thi tuần tự các lệnh từ đầu đến cuối. Nếu một lệnh nào đó bị lỗi thực thi thì Python sẽ dừng

⁴Thuật ngữ khác là **chế độ kịch bản** (script mode). Do đó mã nguồn (dãy lệnh) còn được gọi là **kịch bản** (script) và tập tin mã nguồn còn được gọi là **tập tin kịch bản** (script file).

⁵Chức năng Run → Run... Customized (Shift+F5) cho phép chọn có khởi động lại Python Shell hay không.

⁶Trong IDLE, ở cửa sổ soạn thảo mã nguồn, ta có thể yêu cầu Python kiểm tra lỗi cú pháp (mà không chạy) bằng Run → Check Module (Alt+X).

(và thông báo lỗi) mà không thực thi các lệnh còn lại bên dưới. Nếu không, chương trình kết thúc bình thường (dù có thể có lỗi logic).

Khi Python thực thi một chương trình, tùy theo các lệnh của chương trình mà nó đòi “ai đó” phải nhập liệu (hoặc tương tác như nhấn phím, nhấp chuột, ...), như “ai đó” phải nhập tên (từ bàn phím và nhấn Enter) trong chương trình “Hello” trên. Người làm công việc này (nhập liệu hay tương tác với chương trình) được gọi là **người dùng chương trình** (user). Rõ ràng, người dùng chương trình có vai trò khác với lập trình viên là người viết chương trình. Tuy nhiên, đặc biệt trong Python, *bạn cần phân biệt rõ ràng vì bạn thường đóng vai cả 2: bạn viết chương trình, Python chạy chương trình và bạn dùng luôn chương trình đó (thường là để kiểm tra).*

4.3 Dữ liệu

Dữ liệu (data) ám chỉ tất cả những thứ mà chương trình xử lý. Các dữ liệu cùng dạng, mục đích, cách xử lý, ... được xếp chung vào một nhóm, gọi là **kiểu dữ liệu** (data type). Các **kiểu dữ liệu cơ bản** (basic data type), như tên gọi, là các kiểu dữ liệu đơn giản, quan trọng, hay dùng nhất mà ta đã biết là: số nguyên, số thực, luận lý và chuỗi. Chúng được Python hỗ trợ sẵn nên được gọi là các **kiểu dữ liệu dựng sẵn** (built-in data type). Ngoài 4 kiểu cơ bản trên, Python còn hỗ trợ sẵn nhiều kiểu dữ liệu khác (nâng cao và phức tạp hơn).

Để biết kiểu dữ liệu của một dữ liệu (giá trị) nào đó, ta có thể dùng hàm dựng sẵn `type` như minh họa sau:

```
1 >>> a = 10
2 >>> print(a // 2, type(a // 2))
5 <class 'int'>
3 >>> print(a / 2, type(a / 2))
5.0 <class 'float'>
4 >>> print(a > 0, type(a > 0))
True <class 'bool'>
5 >>> print("10", type("10"))
10 <class 'str'>
```

Như kết quả trên cho thấy, Python “gọi” các kiểu số nguyên, số thực, luận lý và chuỗi lần lượt là `int`, `float`, `bool` và `str`.⁷ Lưu ý, bạn cần phân biệt giá trị 2 (là số nguyên, kiểu `int`) với 2.0 (là số thực, kiểu `float`) cũng như 10 là số với “10” là chuỗi số (kiểu `str`).

Học lập trình Python, ngoài việc học ngôn ngữ Python và các hàm, module, thư viện Python là việc học các kiểu dữ liệu. Ta đã biết khá kĩ về số nguyên, số thực. Chuỗi sẽ được tìm hiểu thêm ở phần dưới. Luận lý sẽ được tìm hiểu kĩ hơn ở bài khác. Các kiểu dữ liệu phức tạp hơn (gồm dựng sẵn và không) sẽ được lần lượt tìm hiểu sau nữa.

⁷Chúng lần lượt là viết tắt của integer, floating point number, boolean và string.

Dưới góc nhìn của dữ liệu, là nguyên vật liệu của chương trình, một chương trình Python điển hình gồm các phần:

- (1) yêu cầu nạp các module cần thiết cung cấp các hàm xử lý dữ liệu,
- (2) nhập dữ liệu vào từ người dùng và chuyển đổi dữ liệu nếu cần,
- (3) thao tác/xử lý/tính toán/chế biến dữ liệu,
- (4) xuất dữ liệu kết quả cho người dùng.

Trong quá trình nhập, xử lý, xuất, các biến được dùng để chứa dữ liệu (dữ liệu nhập, dữ liệu trung gian, dữ liệu phát sinh, dữ liệu xuất) và các hàm (cùng với các toán tử) được dùng để xử lý dữ liệu. Dữ liệu được thao tác/xử lý tuần tự qua nhiều bước nhiều công đoạn. *Chương trình là một dây chuyền xử lý dữ liệu!*

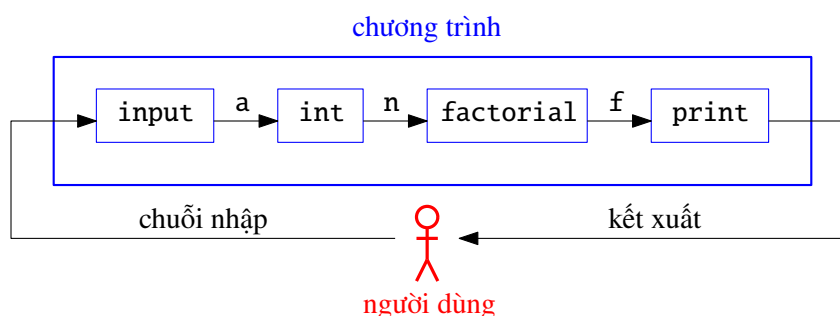
Ứng dụng khuôn chương trình trên, ta viết chương trình Python cho người dùng nhập một số nguyên (không âm), tính và xuất ra giai thừa của số nguyên đó như sau:

<https://github.com/vqhBook/python/tree/master/lesson04/factorial.py>

```

1 import math
2
3 a = input("Nhập n: ")
4 n = int(a)
5 f = math.factorial(n)
6 print("Giai thừa là:", f)
```

Tôi đã cố ý tách bạch từng bước xử lý dữ liệu với kết quả đưa vào từng biến rõ ràng. Lưu ý, vì dữ liệu người dùng nhập là chuỗi (kết quả trả về của `input` luôn là chuỗi) nên ta cần chuyển nó thành số nguyên trước khi dùng hàm `factorial` (của `math`) để tính giai thừa (vì `factorial` nhận đối số là số nguyên). Để chuyển chuỗi số thành số nguyên, ta dùng hàm dựng sẵn `int`. Tương tự, ta có thể dùng hàm `float` để chuyển chuỗi số thành số thực và hàm `str` chuyển số thành chuỗi số. Dây chuyền chế biến dữ liệu trong mã `factorial.py` có thể được hình dung như hình sau:



Ta có thể viết một chương trình cũng thực thi y chang nhưng cô đọng hơn như sau:⁸

⁸Bạn có thể hình dung ta đã “đóng gói” các bước của dây chuyền trên vào một chỗ để khỏi tốn công “vận chuyển” dữ liệu.

```

1 import math
2 print("Giải thừa là:", math.factorial(int(input("Nhập n:
   ↪ "))))

```

Mặc dù 2 chương trình này là tương đương (tức là như nhau) về mặt thực thi nhưng chương trình đầu viết rất rõ ràng (bù lại thì hơi đông dài), chương trình sau viết rất cô đọng (bù lại thì hơi khó đọc). Bạn thích viết kiểu gì? Tôi không thích cả 2. Tôi thích viết như sau:

<https://github.com/vqhBook/python/tree/master/lesson04/factorial2.py>

```

1 import math
2
3 n = int(input("Nhập n: "))
4 print("Giải thừa là:", math.factorial(n))

```

Cân bằng và hài hòa; rõ ràng nhưng không quá đông dài. Ta thường phải lựa chọn đánh đổi tính rõ ràng với tính ngắn gọn; chừng nào là vừa, là tốt? *Kinh nghiệm (sau khi đã lập trình nhiều) và tính cách sẽ cho bạn một lựa chọn.*

Ta đã biết thuật ngữ dữ liệu có nghĩa rất rộng, ám chỉ những thứ mà chương trình xử lý. Có một thuật ngữ còn rộng hơn nữa, **đối tượng** (object), được Python dùng để chỉ tất cả mọi thứ, từ dữ liệu cho đến thao tác, hàm, module, chương trình. *Trong Python, mọi thứ đều là đối tượng.*⁹ Ta sẽ tìm hiểu kĩ hơn vấn đề này trong Bài ??.

4.4 None

Kiểu luận lý (bool) chỉ gồm 2 giá trị là True và False nhưng chưa phải là kiểu đơn giản nhất của Python. Python có **kiểu None** (None type) chỉ gồm một giá trị là None.¹⁰ Thật kì lạ (và khó hiểu) khi có kiểu chỉ có một giá trị vì ta không cần (và không thể) làm gì trên kiểu đó, nó luôn luôn và chỉ có thể là giá trị độc nhất đó. Điều kì lạ (và khó hiểu) hơn là giá trị None được dùng để mô tả “không có gì” hay “không có giá trị”. Đơn giản nhưng khó hiểu như chữ “Không” trong Đạo Phật.¹¹

Một hàm (hay thao tác) không phải lúc nào cũng trả về giá trị.¹² Chẳng hạn, khác với hàm abs cần trả về trị tuyệt đối của một số, hàm print không cần trả về giá trị. Công việc của nó là xuất ra các đối số. Việc nó trả về gì không có ý nghĩa; nó nên không trả về giá trị. Tuy nhiên, để thống nhất (thuận tiện cho động cơ Python hoạt động), mọi hàm (và thao tác) trong Python đều trả về giá trị. Để giải quyết mâu thuẫn này, Python dùng giá trị None để báo là “giá trị không có” (ý

⁹Nó tương tự thuật ngữ “cái” (“thing”) của ngôn ngữ tự nhiên.

¹⁰None cũng là từ khóa như True và False.

¹¹Vì tài liệu này là lập trình Python chứ không phải “triết học Python” nên tôi sẽ không bàn thêm chỗ này.

¹²Những hàm (hay thao tác) như vậy hay được gọi với nghĩa hẹp là **thủ tục** (procedure).

nghĩa) hay “không phải giá trị” (như thông thường).¹³ Thử minh họa sau để hiểu rõ hơn:

```

1 >>> a = print("hi")
hi
2 >>> a
3 >>> print(a, type(a), a == None)
None <class 'NoneType'> True
4 >>> print(print("hi"))
hi
None

```

Lệnh gán thành công ở Dòng 1 cho thấy rằng `print` có trả về giá trị. Tuy nhiên, giá trị này là `None` với ý nghĩa không có gì nên việc truy cập giá trị trong `a` qua biểu thức ở Dòng 2 không được Python xuất ra. Dù vậy, ta vẫn có thể xuất ra “chuỗi biểu diễn” của `None` và xác định kiểu của nó qua hàm `print` và `type` như kết quả của Dòng 3 cho thấy. Lệnh ở Dòng 4 trông khá bất thường về ý nghĩa nhưng vẫn hợp lệ và có thể hiểu được. `None`, như vậy, mang lại sự thống nhất trong mô hình xử lý và đây chính là điều mà Python muốn hướng đến.

4.5 Chuỗi

Có thể nói, *các con số là tất cả những gì mà máy tính thật sự làm*. Tuy nhiên, ta (con người) lại quen thuộc với chuỗi hơn. Chuỗi quan trọng không kém gì số nếu không muốn nói là quan trọng hơn. Phần này sẽ tìm hiểu thêm các thao tác cơ bản trên chuỗi.

Chương trình Python sau giúp “vẽ” một hình chữ nhật có “kích thước” do người dùng chọn:

```

https://github.com/vqhBook/python/tree/master/lesson04/string\_rectangle.py
1 char = input("Nhập kí hiệu vẽ: ")
2 ncol = int(input("Nhập số cột: "))
3 nrow = int(input("Nhập số dòng: "))
4 line = char * ncol + "\n"
5 rect = line * nrow
6 print("\n" + rect)

```

Chẳng hạn, để vẽ hình chữ nhật bằng kí hiệu `*` rộng 10 kí hiệu, cao 3 dòng thì bạn nhập như sau:

```

Nhập kí hiệu vẽ: *
Nhập số cột: 10
Nhập số dòng: 3

```

¹³Ngoài việc dùng để báo hàm không trả về giá trị thì `None` được dùng cho nhiều mục đích khác (rất hay) mà ta sẽ biết sau.


```
*****
*****
*****
```

Ta đã dùng 2 toán tử thao tác trên chuỗi là toán tử + giúp **nối chuỗi** (concatenate) và toán tử * giúp **sao chép lặp lại** (replicate) nhiều lần một chuỗi. Chẳng hạn, "ab" + "cd" được "abcd" còn "cd" + "ab" được "cdab" và "ab" * 3 được "ababab". Lưu ý, không có các phép toán khác trên chuỗi như -, / vì chúng không có ý nghĩa.

Ta cũng có thể dùng các hàm để thao tác trên chuỗi như hàm dựng sẵn len giúp tính **chiều dài** (length) của chuỗi (tức là số lượng kí tự của chuỗi) hay hàm str chuyển số (và các dữ liệu) khác thành chuỗi. Ví dụ, len("cdab") là 4 còn len("") là 0 và len(str(2**1000)) là số lượng chữ số của con số “khủng” 2^{1000} .

Chương trình sau đây minh họa một thao tác quan trọng nữa trên chuỗi là thao tác định dạng:¹⁴

```
1 https://github.com/vqhBook/python/tree/master/lesson04/string\_format.py
2 import math
3
4 r = float(input("Nhập bán kính: "))
5 c = 2 * math.pi * r
6 s = math.pi * r**2
7 print("Chu vi là: %.2f" % c)
8 print("Diện tích là: %.2f" % s)
```

Chương trình cho thấy cách dùng hàm float để chuyển chuỗi số thành số thực. Quan trọng hơn, chương trình cho thấy cách dùng **toán tử định dạng chuỗi** (string formatting operator) %.¹⁵ Toán hạng thứ nhất (bên trái %) là chuỗi gồm các kí tự cố định và kí hiệu %<format> cho biết **chỗ cần thay thế** (replacement field); format là chuỗi đặc tả định dạng xác định cách định dạng toán hạng thứ hai (bên phải %) mà kết quả định dạng sẽ được thay vào chỗ cần thay thế để cùng với các kí tự cố định khác trong toán hạng thứ nhất tạo nên chuỗi kết quả.

Thay vì dùng toán tử định dạng chuỗi, ta cũng có thể dùng **hàng chuỗi được định dạng** (formatted string literal, f-string) trực quan hơn. Chẳng hạn, 2 lệnh xuất ở trên có thể được thay bằng:

```
6 print(f"Chu vi là: {c:.2f}")
7 print(f"Diện tích là: {s:.2f}")
```

Ta thông báo f-string bằng tiền tố f ngay trước hàng chuỗi và dùng cặp ngoặc nhọn thông báo chỗ cần thay thế với cú pháp {<expr>:<format>}, trong đó, expr là biểu thức cho giá trị thay thế và format xác định cách định dạng.

¹⁴Bạn đã biết một cách định dạng kết xuất là dùng hàm format trong Bài tập 3.7.

¹⁵Ta lại gặp hiện tượng lạm dụng kí hiệu: +, *, % cũng là các toán tử trên số.

4.6 Tiếng Việt và Unicode

Một điều quan trọng với chuỗi hay với văn bản nói chung là vấn đề Tiếng Việt (hay vấn đề địa phương hóa). Tin cực kì vui là Python hỗ trợ rất tốt Tiếng Việt (và các thứ tiếng khác). Chẳng hạn, sau đây là chương trình “Hello Việt hóa”:

_____ <https://github.com/vqhBook/python/tree/master/lesson04/hello2.py> _____

```

1 # Chương trình Python đầu tiên
2
3 tên = input("Tên của bạn là gì? ")
4 print("Chào", tên)
```

Và kết quả khi chạy (cũng Việt hóa luôn, nghĩa là gõ tên tiếng Việt):

Tên của bạn là gì? Vũ Quốc Hoàng
Chào Vũ Quốc Hoàng

Dĩ nhiên, tên hàm dựng sẵn `input` và `print` thì không phải Tiếng Việt. Đơn giản vì Python đã chọn tên đó (mà Tiếng Anh là tiếng quốc tế nên hợp lý khi dùng từ Tiếng Anh để đặt tên). Thế bạn vẫn muốn Việt hóa thì sao? Chơi luôn:

_____ <https://github.com/vqhBook/python/tree/master/lesson04/hello3.py> _____

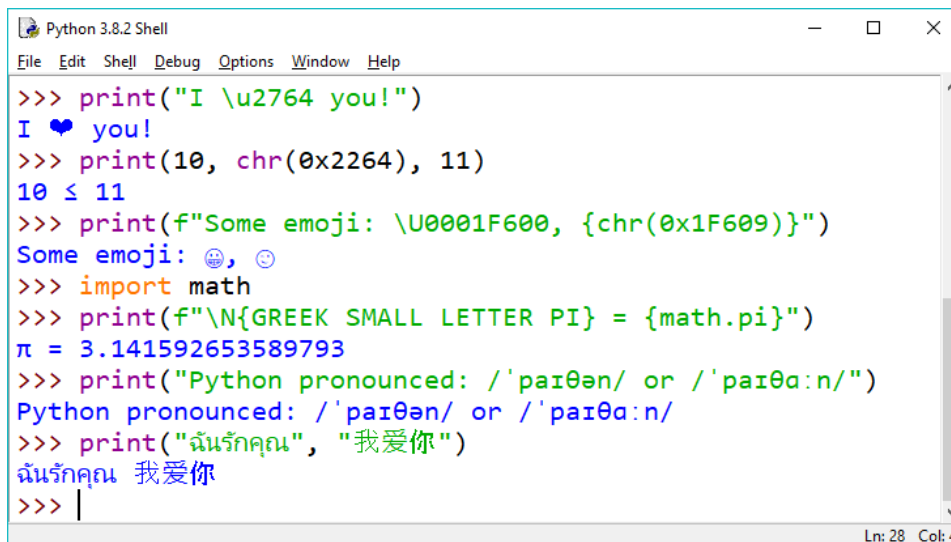
```

1 nhập = input; xuất = print
2
3 tên = nhập("Tên của bạn là gì? ")
4 xuất("Chào", tên)
```

Chương trình chạy tốt nhé! Mẹo cũng đơn giản thôi: ta đặt thêm tên cho các hàm `input` và `print` bằng 2 lệnh gán ở Dòng 1.¹⁶ Điều này cũng tương tự như việc ta đặt nhiều tên biến cho cùng một giá trị. Tưởng tượng, có một hàm dùng để xuất mà tên tiếng Anh là `print` còn tên tiếng Việt là `xuất`, cả 2 tên đều tham chiếu đến cùng hàm đó.

Dĩ nhiên, để dùng Tiếng Việt (đúng hơn là các kí tự Tiếng Việt), ta cần bộ gõ Tiếng Việt (như UniKey). Trường hợp không gõ được thì ta có thể làm như minh họa sau. Nếu biết **mã Unicode** (Unicode code point) của kí hiệu, là con số nguyên xác định kí hiệu, ta có thể gõ mã trực tiếp trong hàng chuỗi bằng cách viết `\uxxxx` như ở Lệnh 1 hoặc ta có thể dùng hàm dựng sẵn `chr` để chuyển mã thành kí tự như ở Lệnh 2. Mã Unicode thường được viết theo cơ số 16 (hexadecimal, hex) mà trong Python ta có thể dùng cách viết hàng số nguyên với tiền tố `0x` (xem Bài tập 4.9). Dĩ nhiên, ta cũng có thể dùng cơ số 10 như cách viết thông thường. Nếu mã Unicode của kí hiệu lớn hơn 4 chữ số hex, ta có thể dùng cách viết `\Uxxxxxxxx` như ở Lệnh 3. Trường hợp biết tên kí hiệu, ta có thể dùng tên (theo chuẩn Unicode) như kí hiệu π trong Lệnh 5.

¹⁶Bạn đã biết rằng có thể viết nhiều lệnh trên 1 dòng bằng cách dùng dấu `;` để phân cách lệnh. Tuy nhiên, bạn không nên lạm dụng điều này trong chương trình Python vì ta có thể viết nhiều lệnh trong một chương trình. Nói chung nên viết mỗi lệnh trên 1 dòng cho rõ ràng.



```

Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
>>> print("I \u2764 you!")
I ♥ you!
>>> print(10, chr(0x2264), 11)
10 ≤ 11
>>> print(f"Some emoji: \U0001F600, {chr(0x1F609)}")
Some emoji: 😄, 😊
>>> import math
>>> print(f"\N{GREEK SMALL LETTER PI} = {math.pi}")
π = 3.141592653589793
>>> print("Python pronounced: /'paɪθən/ or /'paɪθɑ:n/")
Python pronounced: /'paɪθən/ or /'paɪθɑ:n/
>>> print("ฉันรักคุณ", "我爱你")
ฉันรักคุณ 我爱你
>>> |
Ln: 28 Col: 4

```

Một cách khác để làm việc với các kí tự Unicode “lạ” là **chép-dán** (copy-paste) trực tiếp kí tự vào IDLE (Shell hoặc tập tin mã nguồn). Chẳng hạn, ở Lệnh 6, tôi chép phiên âm của chữ python từ trang Wiktionary. Ở Lệnh 7, tôi dùng “Google Translate” để dịch câu tiếng Anh “I love you” sang Tiếng Thái, Tiếng Trung rồi chép-dán trực tiếp vào hằng chuỗi. Bạn có thể tìm hiểu thêm cũng như tra cứu các kí tự Unicode tại trang chủ Unicode Consortium (<https://home.unicode.org/>). Bạn cũng có thể search Google với các cụm từ như “Unicode heart symbol”, “Unicode pi symbol”, “unicode emoji code”, ...

Tóm tắt

- Dãy các lệnh Python, về mặt nội dung, khi được chạy sẽ thực hiện một công việc nào đó, được gọi là chương trình Python; về mặt hình thức, khi viết ra, được gọi là mã nguồn và thường được lưu trữ trong các tập tin mã nguồn có phần mở rộng là `.py`.
- Ghi chú là chuỗi bắt đầu từ dấu `#` đến hết dòng. Python bỏ qua các ghi chú nên chúng có thể được viết theo bất kì cách nào nhưng lập trình viên nên dùng hợp lý để chú thích thêm trên mã nguồn.
- Hàm `input` giúp nhập thông tin, là một chuỗi, từ người dùng.
- Python thực thi chương trình theo chế độ chương trình và người dùng chương trình là người tương tác với chương trình.
- Dữ liệu là tất cả những thứ mà chương trình xử lý như số nguyên, số thực, giá trị luận lý, chuỗi, ... Hàm `type` cho biết kiểu dữ liệu, là thông tin cho biết nhóm và các thao tác có thể thực hiện trên dữ liệu. Chương trình là một dãy chuyển xử lý dữ liệu.

- Các hàm `int`, `float`, `bool`, `str` giúp “chuyển” một dữ liệu về kiểu tương ứng là số nguyên, số thực, luận lý và chuỗi.
- Kiểu đặc biệt `None` chỉ có một giá trị là `None`, mô tả “không có gì”. `None` thường được các thủ tục, về ý nghĩa là các hàm không trả về giá trị, trả về để có sự thống nhất chung là mọi hàm đều phải trả về giá trị.
- Cũng như số, chuỗi là kiểu dữ liệu rất quan trọng. Các thao tác hay dùng trên chuỗi là nối chuỗi (toán tử `+`), sao chép lặp lại (toán tử `*`), định dạng (toán tử `%`), lấy chiều dài chuỗi (hàm `len`) và chuyển dữ liệu thành chuỗi (hàm `str`).
- Python hỗ trợ Tiếng Việt và các thứ tiếng khác rất tốt bằng bộ mã kí tự Unicode.

Bài tập

- 4.1** Trong Bài tập 2.2, ta đã tính giá trị của biểu thức M, N tại các giá trị cụ thể của x, y . Viết chương trình Python cho phép người dùng nhập giá trị của x, y rồi tính và xuất ra giá trị của M, N .

Tính giá trị biểu thức nhập từ người dùng. Ta cũng có thể cho người dùng nhập biểu thức và lượng giá nó bằng hàm dựng sẵn `eval` của Python như minh họa sau:¹⁷

```
1 >>> x = int(input("Nhập giá trị cho x: "))
    Nhập giá trị cho x: 2
2 >>> M = input("Nhập biểu thức cần tính: ")
    Nhập biểu thức cần tính: x**4 - (x - 2)**2 + 2
3 >>> print("Giá trị của biểu thức %s tính tại %d là %d" %
    ↪ (M, x, eval(M)))
    Giá trị của biểu thức x**4 - (x - 2)**2 + 2 tính tại 2 là 18
```

Lệnh xuất trên cũng dùng **bộ 3** (triple, 3-tuple) giá trị và các chuỗi đặc tả định dạng với toán tử định dạng chuỗi. Bạn đã gặp **cặp** (pair hay couple hay 2-tuple) giá trị trong Phần 2.4 và sẽ học kĩ về bộ sau. Bạn cũng có thể dùng f-string để lệnh xuất trên gọn hơn: `print(f"Giá trị của biểu thức {M} tính tại {x} là {eval(M)}")`. Bạn lưu ý cách viết này vì ta sẽ dùng nó thường xuyên từ giờ.

Viết lại chương trình trên để cho người dùng nhập không chỉ giá trị của x, y mà cả biểu thức M .¹⁸

- 4.2** Dùng “phương pháp lặp” ở Bài tập 2.7 để viết chương trình tính căn bậc 2 của một số (thực dương) do người dùng nhập.

¹⁷`eval` và các đặc trưng cao cấp tương tự sẽ được tìm hiểu sau.

¹⁸Người dùng phải nhập một biểu thức Python hợp lệ với chỉ 2 biến x, y . Việc kiểm tra chuỗi nhập có **hợp lệ** (valid) hay không và xử lý khi chuỗi nhập không hợp lệ là một việc khó, quan trọng mà ta sẽ bàn sau.

Lưu ý: Trong chế độ chương trình, Python không hỗ trợ biến đặc biệt Ans (`_`), hơn nữa, Python cũng không tự động xuất ra giá trị các biểu thức (mà ta phải yêu cầu bằng hàm `print`).

4.3 Nghệ thuật ASCII. Viết chương trình xuất ra chữ Python “cách điệu” như mẫu.¹⁹

4.8 Viết chương trình đổi độ F (Fahrenheit) qua độ C (Celsius) theo công thức:

$$C = \frac{5(F - 32)}{9}$$

và đổi ngược lại.

Lưu ý: kết quả sau cùng cần làm tròn thành số nguyên và phải xuất ra được kí hiệu độ ($^{\circ}$).²²

4.9 Số viết theo hàng. Bạn chắc chắn đã biết cách viết số đếm (số tự nhiên): 0, 1, ..., 9, đến 10, 11, ..., 19, đến 20, 21, ..., 99, đến 100, 101, ... Quy tắc: các số được viết gồm nhiều hàng, mỗi hàng là một kí số từ 0 đến 9, khi đếm (tăng 1) ta tăng thêm 1 cho hàng đơn vị mà nếu thành 10 thì ta đặt lại hàng đơn vị là 0 và tăng thêm 1 cho hàng chục mà nếu thành 10 thì ta đặt lại hàng chục và tăng thêm 1 cho hàng trăm, ... Cách viết số này được gọi là **hệ thống số thập phân** (decimal numeral system, decimal) hay **hệ thống số cơ số 10** (base-ten positional numeral system) và được ta dùng thường xuyên.

Một cách tổng quát, ta có thể dùng hệ thống số với **cơ số** (base) $b > 1$. Quy tắc chung: các số được viết gồm nhiều **hàng** (position), mỗi hàng là một **kí số** (digit) từ 0 đến $b - 1$, khi đếm (tăng 1) ta tăng thêm 1 cho hàng thấp nhất (hàng tận cùng bên phải) mà nếu thành b thì ta đặt lại hàng đó là 0 và tăng thêm 1 cho hàng liền trước (hàng ngay trái) mà nếu thành b thì ta đặt lại hàng đó và tăng thêm 1 cho hàng liền trước nữa, ... Ví dụ, với $b = 2$, ta có **hệ thống số nhị phân** (binary numeral system, binary) gồm các số lần lượt là: 0, 1, 10, 11, 100, 101, ... tương ứng lần lượt với số thập phân là 0, 1, 2, 3, 4, 5, ...

Ngoài số nhị phân nói trên thì trong lập trình, người ta cũng hay dùng số **thập lục phân** (hexadecimal, hex) với cơ số $b = 16$. Vì cần tới 16 kí số nên trong hệ thống này người ta qui ước dùng các chữ cái a, b, ..., f (hoặc các chữ cái viết hoa tương ứng) kí hiệu cho các kí số 10, 11, ..., 15. Chẳng hạn số thập lục phân 1a kí hiệu cho số thập phân 26 (là $16 + 10$). Python hỗ trợ các hệ thống số này rất tốt như minh họa sau:²³

```
1 >>> print(0b101, 0x2020, 0x2020 == 2*16**3 + 2*16)
5 8224 True
2 >>> n = 5; print(bin(n), format(n, "b"), f"{n:b}")
0b101 101 101
3 >>> n = 8224; print(hex(n), format(n, "x"), f"{n:X}")
0x2020 2020 2020
4 >>> n = 2**20; print(f"{n:_b}", {n:_x})
1_0000_0000_0000_0000_0000, 10_0000
5 >>> print(int("2020", base=16), int(str(2020), 16))
8224 8224
```

²²Bạn có thể search Google “Unicode degree symbol” để biết mã Unicode của kí hiệu độ.

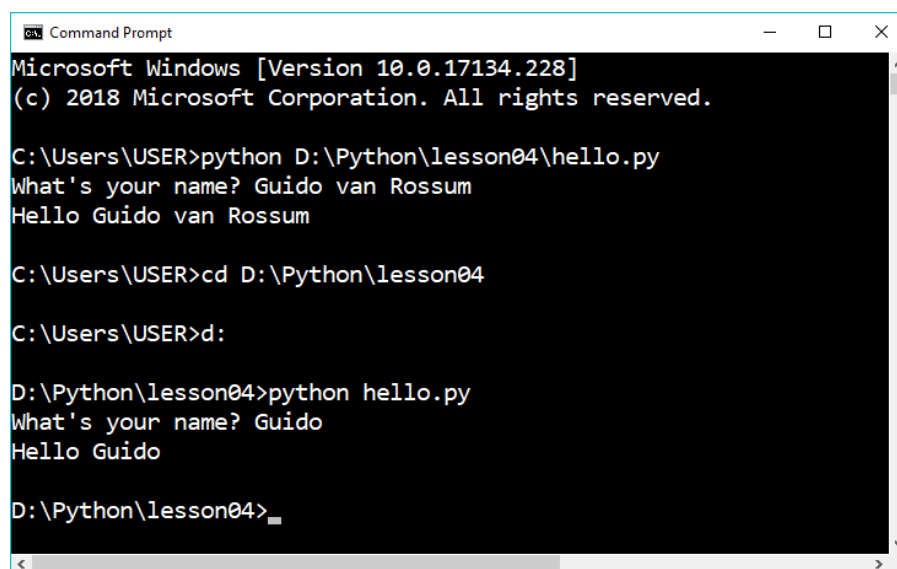
²³Python cũng hỗ trợ tốt số bát phân (cơ số 8) nhưng ít được dùng.

Viết chương trình “đổi cơ số”: nhập số viết theo cơ số này, xuất ra số viết theo cơ số khác.

- 4.10** Dùng f-string, viết chương trình xuất bảng ở Bài tập 1.4 đơn giản hơn bằng các chuỗi đặc tả định dạng phù hợp.

Gợi ý: xem thử kết xuất của cặp lệnh `ch = r"\\"; print(f"|{ch:^20}|")`. Xem thêm chuỗi đặc tả định dạng tại <https://docs.python.org/3/library/string.html#formatspec>.

- 4.11** Chạy chương trình Python từ cửa sổ lệnh. Ở Bài tập 1.5 ta đã dùng Command Prompt để tương tác với Python. Ta cũng có thể dùng nó để yêu cầu Python chạy các chương trình để trong file mã Python. Chẳng hạn, ta cần chạy chương trình trong file mã `hello.py` (ở Phần 4.1) để trong thư mục `D:\Python\lesson04`. Điều quan trọng là ta phải cho Python biết vị trí (tức thư mục) chứa file mã. Cách đơn giản nhất là ta chạy `python` với đường dẫn đầy đủ của file mã (bao gồm thư mục và tên file). Cách thứ 2 là thay đổi thư mục hiện hành của Command Prompt bằng lệnh `cd` thành thư mục chứa file mã và chạy `python` với chỉ tên file mã như hình sau:



```
Microsoft Windows [Version 10.0.17134.228]
(c) 2018 Microsoft Corporation. All rights reserved.

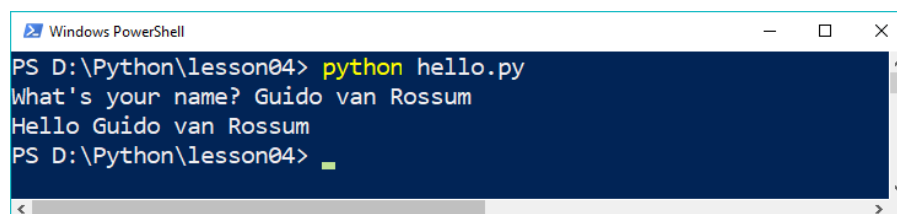
C:\Users\USER>python D:\Python\lesson04\hello.py
What's your name? Guido van Rossum
Hello Guido van Rossum

C:\Users\USER>cd D:\Python\lesson04

C:\Users\USER>d:

D:\Python\lesson04>python hello.py
What's your name? Guido
Hello Guido

D:\Python\lesson04>
```



```
Windows PowerShell

PS D:\Python\lesson04> python hello.py
What's your name? Guido van Rossum
Hello Guido van Rossum
PS D:\Python\lesson04>
```

Ta cũng có thể mở Command Prompt từ thư mục mong muốn bằng cách mở thư mục rồi gõ `cmd` trên ô chứa đường dẫn thư mục hoặc giữ phím Shift rồi nhấp

chuột phải, trong Menu hiện ra, nhấp “Open PowerShell/Command Prompt window here”.²⁴ Sau đó ta chỉ cần chạy python với tên file mã như hình trên.

Chạy lại các chương trình ví dụ trên bằng Command Prompt (hoặc PowerShell).

4.12 Chạy chương trình Python bằng cách nhấp đúp. Trên Windows, ta cũng có thể chạy chương trình Python bằng cách nhấp đúp file mã của nó. Chẳng hạn, nhấp đúp file mã `hello.py` sẽ chạy chương trình trong đó. Tuy nhiên, sau khi chương trình chạy xong, cửa sổ chương trình sẽ đóng nên ta không thấy được kết quả. Để quan sát kết quả ta có thể thêm lệnh `input` ở cuối chương trình Python để đợi người dùng gõ phím và do đó “tạm dừng” cửa sổ. Chẳng hạn, sửa file mã `hello.py` như sau:

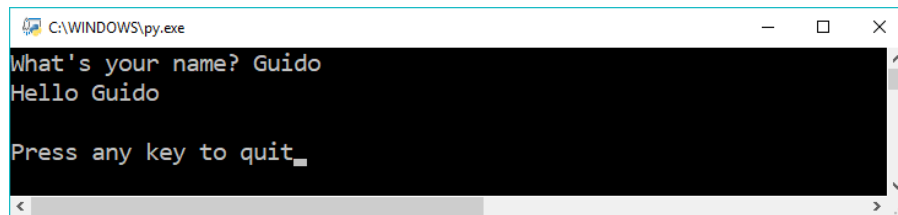
— <https://github.com/vqhBook/python/tree/master/lesson04/hello4.py> —

```

1 # Python program with "pausing"
2
3 name = input("What's your name? ")
4 print("Hello", name)
5
6 input("\nPress any key to quit")

```

rồi nhấp đúp file mã để chạy chương trình như hình sau:



Sửa lại các chương trình minh họa trong bài học để có thể chạy bằng cách nhấp đúp và chạy thử.

4.13 Đọc IDLE, thử các chức năng tiện lợi sau:

- Trong Python Shell, bạn có thể mở các tập tin mã nguồn (đã tạo và lưu trên máy) bằng File → Open (Ctrl+O) hay File → Recent Files.
- Trong cửa sổ soạn thảo tập tin mã nguồn, bạn có thể dùng các chức năng hỗ trợ cho việc soạn mã nguồn trong các menu Edit và Format. Thử nghiệm và tra cứu thêm (Help → IDLE Help hoặc search Google) để biết các chức năng tiện lợi.

²⁴PowerShell là phiên bản cửa sổ lệnh cao cấp hơn Command Prompt.

Bài 5

Case study 1: Vẽ rùa

Đến đây, bạn đã biết lập trình! Các **bài nghiên cứu** (case study) là các bài học tìm hiểu một chủ đề thực tế ở mức độ rộng hơn, sâu hơn với khối lượng lập trình lớn hơn các bài khác. Đây cũng là dịp để bạn tăng cường và nâng cao việc luyện tập cũng như trải nghiệm lập trình. Bài nghiên cứu đầu tiên này không quá lớn và về chủ đề rất thú vị: vẽ, vẽ nhờ một con rùa, **vẽ rùa** (turtle drawing).

5.1 Khởi động con rùa

Thư viện chuẩn Python có module `turtle` hỗ trợ việc học Python và vẽ các hình đơn giản rất tốt. Ta bắt đầu với các lệnh Python sau trong chế độ tương tác:

```
1 >>> import turtle as t
2 >>> t.showturtle()
3 >>> t.shape("turtle")
```

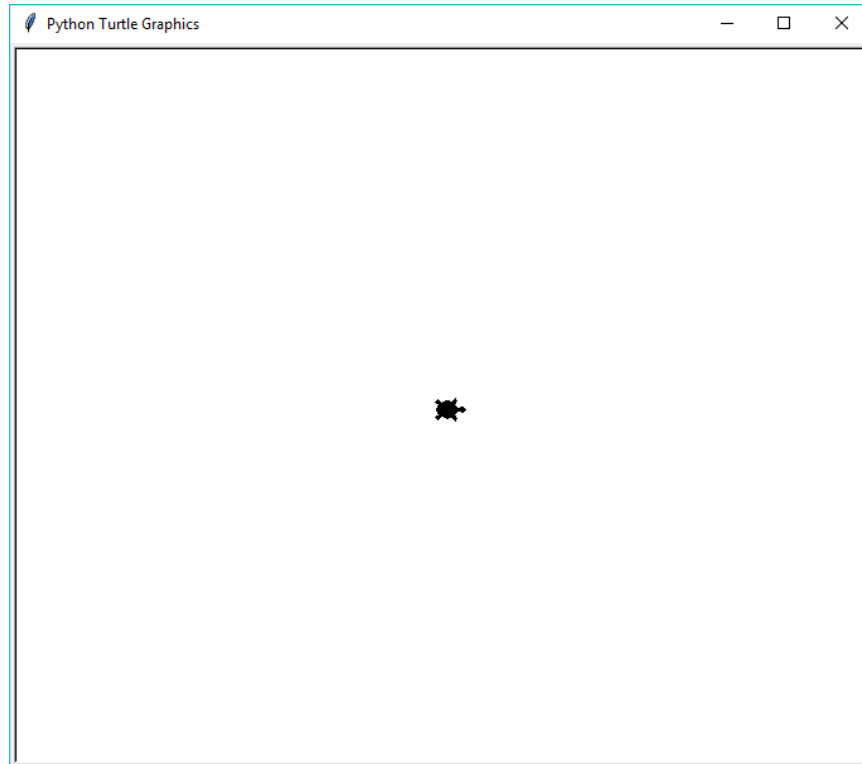
Lệnh đầu tiên nạp module `turtle` nhưng đặt lại tên module là `t` bằng từ khóa `as`, mà mục đích là để gõ ít phím hơn. Ta cũng đã dùng dạng `import` này ở Bài tập 3.10 nhưng không nên lạm dụng.¹ Lệnh thứ hai gọi một hàm của module `turtle` (mà bây giờ được viết gọn là `t`) là `showturtle`.² Như tên gọi, hàm này hiển thị một cửa sổ trong đó có một ... “mũi tên” ở giữa. Đó chính là con rùa với hình dạng ... một mũi tên!:) Để hiển thị hình dạng con rùa, ta có thể dùng hàm `shape`,³ với

¹Đặt lại tên `t` cho `turtle` được cộng đồng Python dùng nhiều. Những module có tên ngắn (như `math`) thì không nên dùng cách này (dùng tên `math` luôn).

²Theo phong cách PEP 8 thì nên đặt tên là `show_turtle`. Tuy nhiên, không phải ai cũng theo PEP 8 hết, kể cả những người làm nên Python, vì nhiều lí do trong đó có lí do “lịch sử” (nói cách khác là lỡ đặt tên vậy rồi!). Bạn là người mới thì nên theo phong cách PEP 8 nhưng nếu gặp mã nguồn không theo PEP 8 thì cũng đừng bức xúc lắm nhé vì bạn sẽ gặp khá thường xuyên.

³Đúng ra phải gọi là hàm `shape` của gói `turtle` nhưng gọi như vậy dài quá mà ngữ cảnh (tên gói và dấu chấm đằng trước) đã cho thấy rõ điều này. Tôi sẽ thường xuyên dùng cách gọi tắt như vậy từ giờ.

đối số là chuỗi "turtle".⁴ Bây giờ con rùa đã có hình dạng nguyên bản, nằm ngay chính giữa cửa sổ và hướng sang phải như hình dưới.



Tiếp theo, ta ra lệnh cho con rùa tiến tới 100 “bước” rồi quay trái 90° bằng 2 lệnh sau:

```
1 >>> t.forward(100)
2 >>> t.left(90)
```

Trong chế độ tương tác, ta sẽ thấy con rùa bò (đúng ra là chạy:)) tới và quay trái rất thú vị. Bạn cũng có thể kéo cửa sổ con rùa (cửa sổ có tiêu đề “Python Turtles Graphics”) dóng ngang với cửa sổ IDLE để khỏi bị che, sẽ thấy rõ hơn.

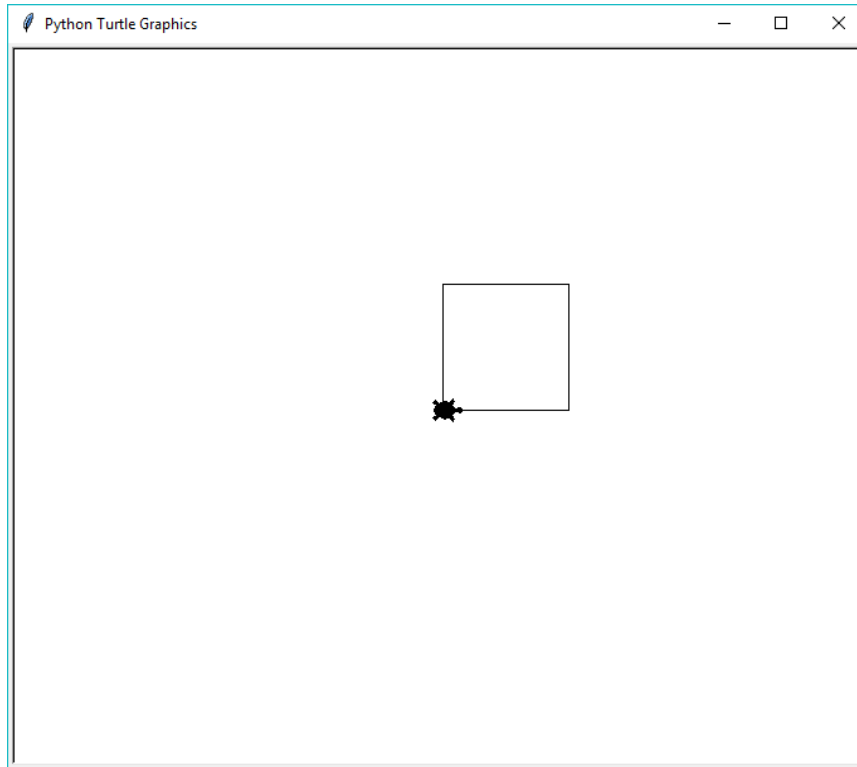
Tiếp theo, ta ra lệnh cho con rùa tiến tới 100 “bước” rồi quay trái 90° thêm 3 lần như vậy nữa để hoàn thành hình vuông cạnh 100 bằng các lệnh sau (mà thực chất là lặp lại 3 lần cặp lệnh trên):⁵

```
1 >>> t.forward(100); t.left(90)
2 >>> t.forward(100); t.left(90)
3 >>> t.forward(100); t.left(90)
```

⁴Như bạn đoán, có thể dùng các hình dạng khác cho con rùa bằng đối số phù hợp cho hàm `shape`. Bạn tra cứu module `turtle` và hàm `shape` để biết thêm chi tiết (serach “Python docs turtle”).

⁵Bạn có thể dùng chức năng lấy lại các lệnh đã gõ của IDLE (phím tắt Alt+P) mà không cần gõ lệnh.

Hình sau minh họa kết quả:



Chúc mừng ... con rùa!!! Bạn (đúng hơn là con rùa) đã thực hiện thành công hình vẽ đầu tiên, hình vuông cạnh 100. Việc vẽ với con rùa trong chế độ tương tác rất thú vị. Tuy nhiên, với các hình lớn (nhiều lệnh) thì ta nên dùng chế độ chương trình. Thử chương trình minh họa sau:

```
1 import turtle as t
2 t.shape("turtle")
3 d = int(input("Kích thước hình vuông? "))
4 t.forward(d); t.left(90)
5 t.forward(d); t.left(90)
6 t.forward(d); t.left(90)
7 t.forward(d); t.left(90)
```

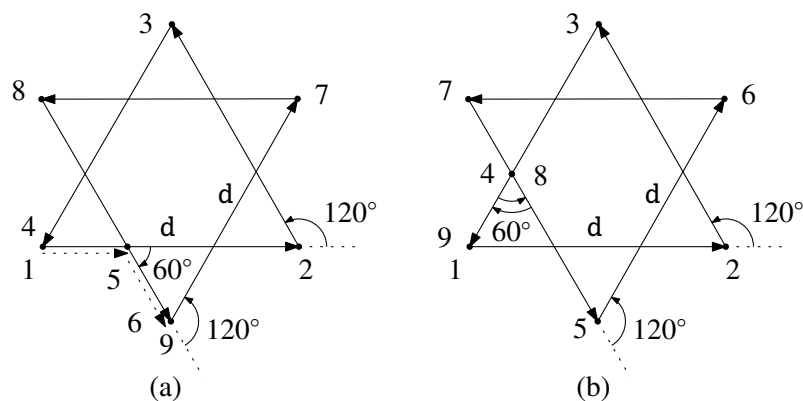
Ta cũng đã cho người dùng nhập kích thước hình vuông (độ dài cạnh). Thực thi chương trình này, ta có kết quả là hình vuông với kích thước như người dùng nhập. Cũng lưu ý, trong chế độ chương trình ta không nên viết nhiều lệnh trên một dòng, tuy nhiên với các lệnh ngắn và nhất là khi việc “gom lại” có ý nghĩa thì ta cũng có thể viết “vài” lệnh trên một dòng (như các Dòng 4-7).⁶

⁶Trong tài liệu này, việc viết nhiều lệnh trên một dòng (cũng như bỏ đi các ghi chú và các dòng trống) là một “thủ đoạn” nữa để tiết kiệm giấy.

Câu hỏi tự nhiên: 100 “bước” là bao xa? hay “bước” là gì? Điều này tùy thuộc vào cách ta thiết lập hệ trục tọa độ cho cửa sổ con rùa. Ta sẽ bàn kĩ vấn đề này trong Bài tập 5.5, trước mắt, ta cứ chạy và “cảm nhận”.

5.2 Bắt con rùa bò nhiều hơn

Ta đã khởi động con rùa ở phần trên với hình vuông đơn giản. Trong phần này, ta sẽ bắt con rùa làm việc nhiều hơn để vẽ các hình phức tạp hơn. Đầu tiên là hình ngôi sao 5 cánh với chiến lược vẽ (tức “kế hoạch bò” của con rùa) như Hình (a) sau:



Chiến lược này được “hiện thực” bằng chương trình sau:

```

1  import turtle as t
2  t.shape("turtle")
3
4  d = 200
5
6  t.forward(d); t.left(120)      # 1 --> 2
7  t.forward(d); t.left(120)      # 2 --> 3
8  t.forward(d); t.left(120)      # 3 --> 4 (1)
9
10 t.up()
11 t.forward(d/3); t.right(60)    # 1 --> 5 (up)
12 t.forward(d/3); t.left(120)   # 5 --> 6 (up)
13 t.down()
14
15 t.forward(d); t.left(120)      # 6 --> 7
16 t.forward(d); t.left(120)      # 7 --> 8
17 t.forward(d); t.left(120)      # 8 --> 9 (6)

```

Biến `d` xác định kích thước ngôi sao và ta cũng có thể cho người dùng nhập kích thước này. Chiến lược của ta là vẽ 2 hình tam giác đều đan nhau. Sau khi vẽ

hình tam giác đều thứ nhất (tam giác gồm các cạnh 1-2, 2-3, 3-4), ta “nhắc bút” bằng hàm `up`, nghĩa là ta cho con rùa bò không mà không vẽ (bình thường khi con rùa bò sẽ để lại “vết”, đó chính là đường vẽ). Sau khi nhắc bút và cho con rùa bò đến đỉnh của hình tam giác đều thứ 2 (bò không vẽ từ điểm 4 đến điểm 5, quay phải 60° , bò không vẽ từ điểm 5 đến điểm 6, rồi quay trái 120°), ta “hạ bút” bằng hàm `down` và bắt con rùa vẽ hình tam giác đều thứ 2 (tam giác gồm các cạnh 6-7, 7-8, 8-9) để hoàn thành hình ngôi sao.

Bạn có thấy tội con rùa không? Ta sẽ giảm thiểu việc “hành hạ” con rùa bằng việc chỉ “vẽ một nét mà không nhắc bút”. Đây cũng là một trò chơi (và là một thách thức) mà ta hay gặp. Ta có thể vẽ như vậy với hình ngôi sao không? Được. Với chiến lược vẽ như Hình (b) trên, con rùa sẽ đỡ cực nhất với mã sau đây:

<https://github.com/vqhBook/python/tree/master/lesson05/star2.py>

```
1 import turtle as t
2 t.shape("turtle")
3 t.speed("slowest")
4
5 d = 200
6
7 t.forward(d); t.left(120)
8 t.forward(d); t.left(120)
9 t.forward(2*d/3); t.left(60)
10 t.forward(2*d/3); t.left(120)
11 t.forward(d); t.left(120)
12 t.forward(d); t.left(120)
13 t.forward(d/3); t.right(60)
14 t.forward(d/3); t.left(120)
```

Tôi đã dùng hàm `speed` với đối số `"slowest"` để đặt tốc độ chậm nhất cho con rùa (đúng nghĩa là bò chứ không chạy như trước nữa) để ta có thể dễ theo dõi quá trình con rùa “vẽ một nét”. Đúng là “vẽ một nét (mà không nhắc bút)” đấy nhé.⁷ Như dự đoán, bạn có thể đặt các tốc độ khác cho con rùa, chẳng hạn, cho con rùa cõ tên lửa với đối số `"fastest"`.⁸

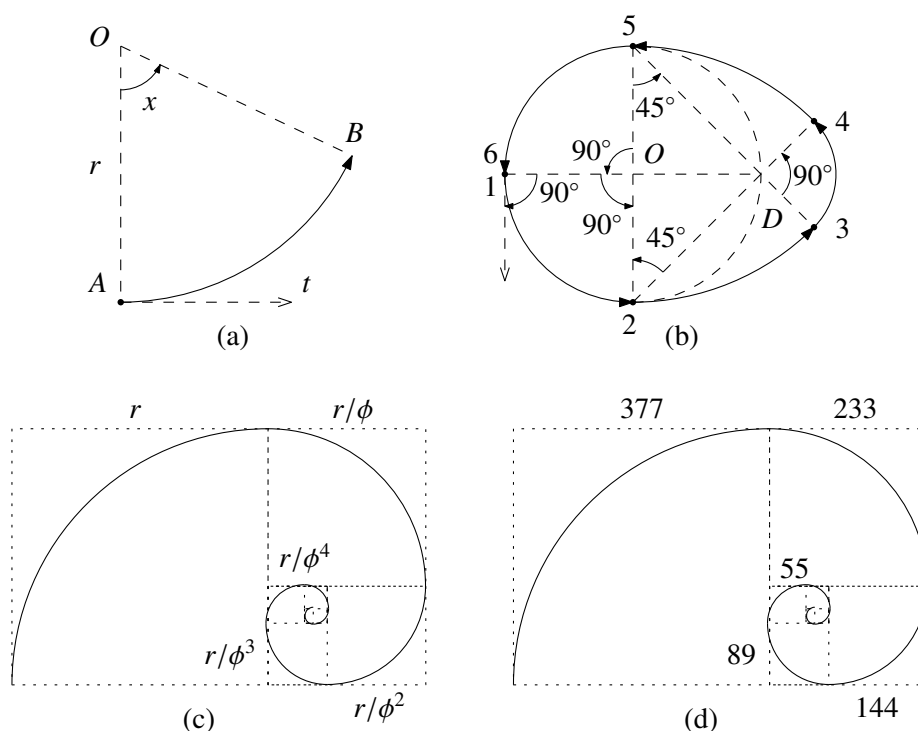
5.3 Bắt con rùa bò nhiều hơn nữa

Ta sẽ vẽ những hình phức tạp hơn trong phần này. Điều này đòi hỏi cách ta bắt con rùa “bò” và “quay”. Cũng như Python, con rùa rất ngoan ngoãn và chịu khó. Nó có thể vẽ bất kì hình nào nếu ta biết cách bảo nó (tức là dùng các hàm phù hợp của gói `turtle`). Trước hết, với hàm `forward` ta đã bảo con rùa bò theo **đường thẳng** (line), ta cũng có thể bảo con rùa bò theo một **cung tròn** (arc) với hàm

⁷ Có phải hình nào cũng vẽ một nét được không? Nếu có thì cách vẽ thế nào? Thử ví thay, Toán học lại cho câu trả lời đơn giản và rõ ràng. (Euler đã trả lời câu hỏi này với lý thuyết về **đường đi Euler** (Eulerian path) mà bạn sẽ có dịp biết nếu học khoa học máy tính.)

⁸ Cũng vậy, đây là dịp để bạn tra cứu và thử nghiệm.

circle. Xem minh họa trong Hình (a) dưới đây, giả sử con rùa đang ở điểm (vị trí) A và hướng theo tia At , ta có thể bảo con rùa bò (và vẽ) theo cung tròn \widehat{AB} có tâm ở O với bán kính r ($OA = r$, $OA \perp At$ và O nằm bên trái tia At) và góc ở tâm là x° theo hướng “**ngược chiều kim đồng hồ**” (counterclockwise), bằng cách gọi `circle(r, x)`. Nếu đổi số thứ nhất (bán kính) âm thì con rùa sẽ bò theo hướng “cùng chiều kim đồng hồ” (tâm ở bên phải hướng). Nếu không có đối số thứ 2 (góc ở tâm) thì con rùa sẽ bò đủ 360° , tức là vẽ đủ một đường tròn.⁹



Dùng hàm `circle`, theo chiến lược ở Hình (b) trên, chương trình sau đây vẽ hình một “quả trứng”:¹⁰

<https://github.com/vqhBook/python/tree/master/lesson05/egg.py>

```

1 import turtle as t
2 t.shape("turtle")
3 t.width(2)
4 r = 100
5
6 t.right(90)
7 t.circle(r, 90)
8 t.circle(2*r, 45)
9 t.circle(2*r - 2**0.5*r, 90)

```

⁹Rõ ràng, bạn nên tra cứu để biết chi tiết hơn về hàm `circle` và các đối số của nó.

¹⁰SGK Toán 9 Tập 1, trang 125.

```

10 t.circle(2*r, 45)
11 t.circle(r, 90)

```

Xuất phát từ điểm 1, hướng sang phải, ta quay phải con rùa 90° để con rùa hướng xuống dưới (hướng Nam), sau đó ta bảo con rùa bò theo cung $\widehat{12}$ có bán kính r (tâm O , góc ở tâm 90°), sau đó ta bảo con rùa bò theo cung $\widehat{23}$ có bán kính $2r$ (tâm là điểm 5, góc ở tâm 45°), ...¹¹ Ta cũng đã tăng kích thước nét vẽ lên gấp đôi bằng hàm `width` để nét vẽ đậm hơn.

Chương trình sau đây vẽ “đường xoắn ốc vàng” (golden spiral) như Hình (c) ở trên.¹²

```

https://github.com/vqhBook/python/tree/master/lesson05/golden_spiral.py
1 import turtle as t
2 t.speed("slowest")
3
4 PHI = (1 + 5**0.5)/2
5 r = 377
6
7 t.left(90)
8 t.circle(-r, 90); t.circle(-r/PHI, 90)
9 t.circle(-r/PHI**2, 90); t.circle(-r/PHI**3, 90)
10 t.circle(-r/PHI**4, 90); t.circle(-r/PHI**5, 90)
11 t.circle(-r/PHI**6, 90); t.circle(-r/PHI**7, 90)
12 t.circle(-r/PHI**8, 90); t.circle(-r/PHI**9, 90)
13 t.circle(-r/PHI**10, 90); t.circle(-r/PHI**11, 90)
14
15 t.hideturtle()

```

Chương trình có dùng một hằng số, gọi là **tỉ lệ vàng** (golden ratio), $\phi = \frac{1+\sqrt{5}}{2}$. Ở cuối chương trình, ta ẩn con rùa đi (để dễ nhìn hình) bằng hàm `hideturtle`. Nhân tiện, các biến chứa giá trị cố định như PHI thường được gọi là **hằng** (constant) và được qui ước viết HOA trong Python.¹³

Điều hay ho là ta có thể vẽ “từ trong ra” (thay vì “từ ngoài vào”) bằng dãy Fibonacci như sau (còn gọi là **Fibonacci spiral**) (Hình (d) trên):

```

https://github.com/vqhBook/python/tree/master/lesson05/Fibonacci_spiral.py
1 import turtle as t
2 t.speed("slowest")
3
4 t.circle(0, 90); t.circle(1, 90); t.circle(1, 90)
5 t.circle(2, 90); t.circle(3, 90); t.circle(5, 90)

```

¹¹Bạn tiếp tục phân tích, đối chiếu từng lệnh của mã nguồn trên với hình minh họa để hiểu rõ. Bạn cũng phải làm tương tự như vậy cho các chương trình “lớn” từ giờ.

¹²Đúng hơn là xấp xỉ đường xoắn ốc vàng. Cũng vậy, bạn tự phân tích, đối chiếu các lệnh trong mã nguồn với hình vẽ minh họa để hiểu rõ.

¹³Phân biệt thuật ngữ hằng (constant) này với hằng số, hằng chuỗi, ... (literal).

```

6 t.circle(8, 90); t.circle(13, 90); t.circle(21, 90)
7 t.circle(34, 90); t.circle(55, 90); t.circle(89, 90)
8 t.circle(144, 90); t.circle(233, 90); t.circle(377, 90)
9
10 t.hideturtle()

```

Ta đã dùng **dãy số Fibonacci** (Fibonacci sequence) trong các bán kính hình tròn ở trên: 0, 1, 1, 2, 3, 5, 8, ...¹⁴ Điều thú vị của những hình này thật ra là những vấn đề Toán học đằng sau đó. Còn con rùa, nó chẳng biết gì đâu, ta bảo gì nó làm nấy thôi.

5.4 Tô màu

Ta đã vẽ các hình (đường nét) ở các phần trên, với turtle, ta cũng có thể tô màu cho các hình. Chương trình sau đây vẽ và tô màu **“biểu tượng Âm-Dương”** (Yin-Yang symbol):

https://github.com/vqhBook/python/tree/master/lesson05/yin_yang.py

```

1 import turtle as t
2 r = 120
3
4 t.color("red")
5 t.begin_fill(); t.circle(r); t.end_fill()
6
7 t.color("black")
8 t.begin_fill()
9 t.circle(r, 180); t.circle(r/2, 180); t.circle(-r/2, 180)
10 t.end_fill()
11
12 t.right(90); t.up(); t.forward(r/2 - r/10); t.right(90)
13 t.color("black")
14 t.begin_fill(); t.circle(r/10); t.end_fill()
15
16 t.left(90); t.up(); t.forward(r); t.right(90)
17 t.color("red")
18 t.begin_fill(); t.circle(r/10); t.end_fill()
19
20 t.hideturtle()

```

Ta tô màu bằng cách đặt các lệnh vẽ giữa lời gọi hàm `begin_fill()` và `end_fill()`. Trước đó, ta đặt màu tô bằng hàm `color`. Ta sẽ học cách mô tả một màu “bất kì” sau (xem Bài tập 5.3), trước mắt, ta dùng chuỗi **tên màu** (color name) như red (đỏ), green (xanh lục), blue (xanh lam), black (đen), ... để mô tả các

¹⁴Bạn để ý là cứ lấy 2 số đằng trước cộng lại sẽ được số đằng sau.

màu hay gặp.¹⁵ Lưu ý là ta có thể gọi hàm `color` với 2 đối số để đặt riêng tương ứng màu đường viền và màu tô bên trong. Chẳng hạn, lời gọi `t.color("black", "yellow")` giúp tô màu hình sau đó với màu đường viền là đen và màu tô bên trong là vàng.

5.5 Chế độ trình diễn

Thật ra con rùa của chúng ta là một “siêu rùa”. Nó đã cố ý bò chậm để trình diễn cho ta thấy. Trường hợp muốn vẽ các hình thật nhanh, nhất là khi vẽ các hình phức tạp, ta có thể bảo con rùa bay với tốc độ ... tên lửa bằng cách tắt chế độ **trình diễn** (animation). Để làm việc này, ta đặt các yêu cầu cho con rùa (vẽ, quay, tô, ...) giữa lời gọi hàm `tracer(False)` và `update()`. Để bật lại chế độ trình diễn, ta gọi `tracer(True)`. Chương trình sau yêu cầu người dùng nhập “bán kính” rồi vẽ và tô một hình vuông với tốc độ ánh sáng:¹⁶ Ta sẽ tìm hiểu kĩ hàm `numinput` (của `turtle`) trong Bài tập 5.7.

```

1 https://github.com/vqhBook/python/tree/master/lesson05/animation\_off.py
2 import turtle as t
3
4 x = t.numinput("Hello", "Enter radius: ")
5 y = 2**0.5 * x
6 t.hideturtle()
7 t.color("black", "yellow")
8
9 t.tracer(False)
10 t.up(); t.forward(x); t.left(135); t.down()
11 t.begin_fill()
12 t.forward(y); t.left(90); t.forward(y); t.left(90)
13 t.forward(y); t.left(90); t.forward(y); t.left(90)
14 t.end_fill()
15 t.update()

```

Vẫn còn nhiều điều thú vị nữa về con rùa mà ta sẽ tìm hiểu thêm và tôn vinh nó trong một bài khác. Trước mắt, con rùa là một “công cụ” sinh động giúp ta học lập trình vui và dễ dàng hơn. Nếu bạn là giáo viên, hãy bắt tội nhỏ (học sinh, sinh viên) đọc nhiều với con rùa. Nếu bạn là tội nhỏ, hãy tự đọc nhiều với con rùa. Nếu bạn không là giáo viên hay tội nhỏ, cũng hãy đọc nhiều với con rùa. Tóm lại, tôi rất thích con rùa và sẽ dùng nó làm minh họa cho nhiều kĩ thuật Python trong các bài sau. *Ôi, tội nghiệp con rùa!*¹⁷

¹⁵Bạn có thể tham khảo các “tên màu turtle” này ở một số trang, chẳng hạn <https://trinket.io/docs/colors>.

¹⁶Bạn cũng thử làm tương tự cho các chương trình minh họa trước.

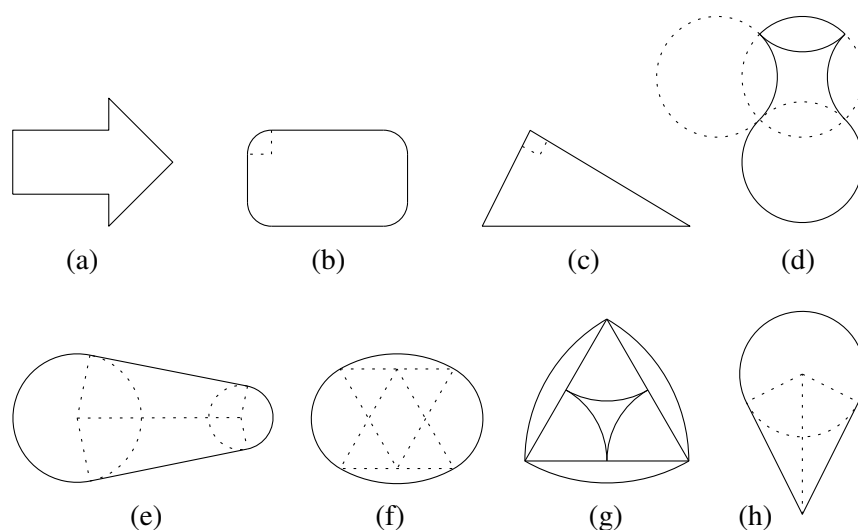
¹⁷Tra cứu <https://docs.python.org/3/library/turtle.html#module-turtle>.

Tóm tắt

- *Vẽ rùa là một công cụ học lập trình rất bổ ích. Nó được Python hỗ trợ trong module turtle.*
- *Ta có thể bảo con rùa vẽ đoạn thẳng với hàm forward, vẽ cung tròn với hàm circle, và quay trái/phải với hàm left/right.*
- *Ta có thể nhấc/đặt bút vẽ với hàm up/down. Việc này giúp di chuyển linh hoạt con rùa khi cần vẽ các hình “rời nhau”.*
- *Ta có thể tô màu với các hàm color, begin_fill, và end_fill.*
- *Ta có thể ẩn/hiện con rùa với hàm hideturtle/showturtle, điều chỉnh kích thước nét vẽ, hình dạng và tốc độ của con rùa với hàm width, shape, và speed.*
- *Ta cũng có thể tắt/bật chế độ trình diễn với các hàm tracer và update.*
- *Để vẽ (và tô màu) các hình phức tạp, ta cần “rã” hình ra thành các phần đơn giản và có chiến lược yêu cầu con rùa thực hiện lần lượt các bước. Đây chính là ý niệm cơ bản nhất của lập trình.*

Bài tập

5.1 Vẽ các hình dưới.¹⁸



Lưu ý: không vẽ các nét đứt; các nét đứt được dùng để gợi ý chiến lược vẽ.

Gợi ý: bạn có thể phải tính toán một chút với độ dài cạnh và số đo góc bằng các hàm lượng giác (như trong Phần 3.2).

¹⁸Dĩ nhiên là dùng con rùa của Python để vẽ. Các hình được hiệu chỉnh từ SGK Toán 8, 9.

5.2 Vẽ bằng một nét các Hình (d) và (g) ở Bài tập 5.1.

Gợi ý: bạn phải xuất phát ở vị trí phù hợp của Hình (d). Cả 2 hình đều có thể vẽ một nét được.

5.3 Mô hình màu RGB. Các hệ thống đồ họa trên máy tính thường dùng **mô hình màu RGB** (RGB color model) để mô tả màu sắc. Theo đó, các màu được “tổng hợp” từ 3 thành phần cơ bản là **đỏ** (Red), **xanh lục** (Green) và **xanh dương** (Blue) với **cường độ** (intensity) khác nhau. Trên máy, cường độ màu thường được mô tả bằng con số thực trong khoảng $[0, 1]$ hoặc con số nguyên trong khoảng $[0, 255]$ mà số càng lớn thì cường độ càng lớn. Chẳng hạn, màu đỏ có cường độ 3 thành phần R, G, B lần lượt là 255, 0, 0; màu trắng là 255, 255, 255; màu đen là 0, 0, 0; ...¹⁹

Ngoài tên màu, Turtle cho phép mô tả màu RGB bằng chuỗi **mã màu** (color code) có dạng #RRGGBB trong đó RR, GG, BB là 2 kí số cơ số 16 mô tả cường độ thành phần đỏ, xanh lục, xanh dương tương ứng (giá trị nguyên trong phạm vi 0-255). Chẳng hạn, mã màu đỏ là #FF0000, màu trắng là #FFFFFF, màu đen là #000000, ... Bạn có thể dùng các trang “tra màu” trên Web như https://www.rapidtables.com/web/color/RGB_Color.html để tra các mã màu này. Ngoài ra, Turtle cho phép mô tả màu bằng bộ 3 số nguyên (0-255) hoặc bộ 3 số thực (0-1) tùy theo chế độ màu mà ta có thể đặt bằng hàm colormode. Thử minh họa sau để rõ hơn:

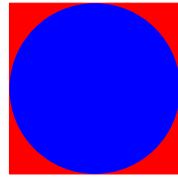
<https://github.com/vqhBook/python/tree/master/lesson05/color.py>

```

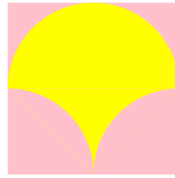
1 import turtle as t
2
3 r = 120
4 t.up()
5 t.left(90); t.forward(r); t.left(90)
6 t.down()
7
8 t.color("#FF0000")          # Red
9 t.begin_fill(); t.circle(r, 120); t.end_fill()
10
11 t.colormode(255)
12 t.color((0, 255, 0))       # Green
13 t.begin_fill(); t.circle(r, 120); t.end_fill()
14
15 t.colormode(1.0)
16 t.color((0.0, 0.0, 1.0))   # Blue
17 t.begin_fill(); t.circle(r, 120); t.end_fill()
18
19 t.hideturtle()
```

¹⁹Bạn có thể “nghĩa qua” https://en.wikipedia.org/wiki/RGB_color_model để biết sơ lược về màu RGB.

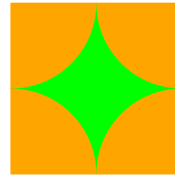
Tô màu các hình sau với các cách mô tả màu khác nhau:



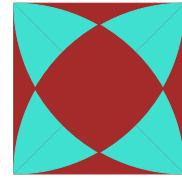
(a)



(b)

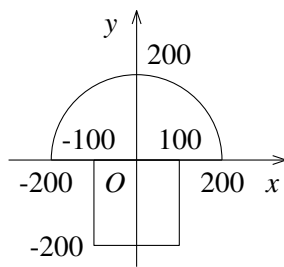


(c)

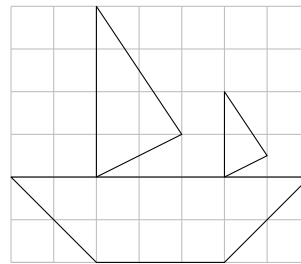


(d)

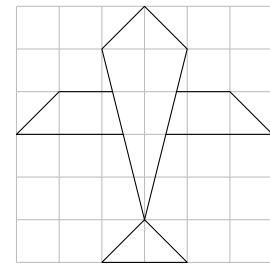
5.4 “Chế độ vẽ tuyệt đối”. Với các hàm `forward`, `circle`, `left/right`, ta đã vẽ theo “chế độ tương đối” vì các thao tác được thực hiện theo vị trí và hướng hiện tại của con rùa. Ta có thể di chuyển con rùa đến một điểm mà không cần để ý đến vị trí hiện tại và quay con rùa về một hướng mà không cần để ý đến hướng hiện tại. Chương trình sau minh họa cách vẽ Hình (a) ở dưới theo “chế độ tuyệt đối”.



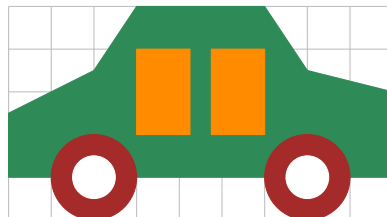
(a)



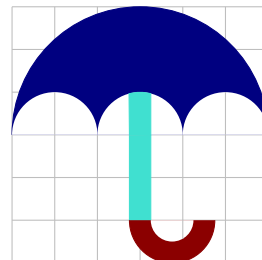
(b)



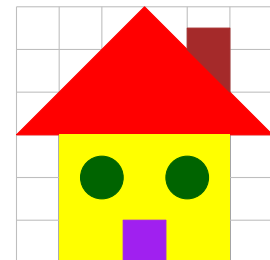
(c)



(d)



(e)



(f)

```

1 https://github.com/vqhBook/python/tree/master/lesson05/mushroom.py
2 import turtle as t
3 t.hideturtle()
4 t.color("saddle brown")
5 t.begin_fill()
6 t.goto(100, 0); t.goto(100, -200);
7 t.goto(-100, -200); t.goto(-100, 0)
8 t.end_fill()

```

```

9
10 t.color("orange red")
11 t.begin_fill()
12 t.goto(200, 0); t.setheading(90); t.circle(200, 180);
   ↳ t.goto(0, 0)
13 t.end_fill()

```

Vị trí các điểm cũng được con rùa xác định bằng hệ trục tọa độ Oxy , có gốc $(0, 0)$ ở giữa cửa sổ và hướng như thông thường (xem Bài tập 5.5). Để di chuyển con rùa đến điểm xác định nào đó (và vẽ nếu đang đặt bút), ta gọi hàm `goto` với đối số là hoành độ, tung độ của điểm đến (hướng của con rùa giữ không đổi). Để quay con rùa về hướng xác định nào đó, ta gọi hàm `setheading` với đối số là góc xác định hướng (0° – Đông, 90° – Bắc, 180° – Tây, 270° – Nam). Việc kết hợp chế độ vẽ tương đối với tuyệt đối thường giúp vẽ dễ dàng hơn các hình phức tạp.

Vẽ và tô màu các hình trên. Các ô lưới giúp xác định tọa độ các điểm dễ dàng hơn. Nên suy nghĩ để đưa ra chiến lược vẽ tốt (nhanh, đơn giản) trước khi gõ chương trình.²⁰

5.5 Hệ trục tọa độ Turtle. Cũng tương tự như cách ta xác định vị trí trên mặt phẳng, Turtle dùng **hệ trục tọa độ** (coordinate system) để xác định vị trí trên cửa sổ. Mặc định, gốc tọa độ của vị trí được đặt ở chính giữa cửa sổ với trục Ox hướng sang trái, trục Oy hướng lên trên và đơn vị trên cả 2 trục là 1 **điểm ảnh** (pixel). Kích thước thực sự của 1 điểm ảnh phụ thuộc **độ phân giải màn hình** (display resolution), tuy nhiên, về logic nó là kích thước nhỏ nhất có thể “nhìn được” trên màn hình.

Turtle cho phép ta thiết lập hệ trục tọa độ này bằng hàm `setworldcoordinates` với 4 đối số lần lượt xác định vị trí biên trái, biên dưới, biên phải và biên trên của cửa sổ sau khi đặt hệ trục. Lưu ý, kích thước đơn vị không nhất thiết như nhau cho 2 trục. Chương trình sau đây vẽ hình “con tàu” (Hình (b)) ở Bài tập 5.4 đơn giản hơn nhờ việc đặt lại hệ trục tọa độ. Cụ thể, ta đặt lại hệ trục để hệ thống lưới của hình “phủ đầy” cửa sổ mà khi đó vị trí biên trái là 0, biên dưới là 0, biên phải là 7, biên trên là 6 (lưu ý, các ô có thể không còn vuông nữa). Sau đó ta vẽ bằng cách định vị tọa độ các đỉnh trong hệ trục mới.

<https://github.com/vqhBook/python/tree/master/lesson05/coordinates.py>

```

1 import turtle as t
2
3 t.setworldcoordinates(0, 0, 7, 6)
4 t.hideturtle()
5
6 t.up(); t.goto(0, 2); t.down()

```

²⁰Đối với các chương trình lớn, phức tạp, bạn nên tập thói quen phân tích, suy nghĩ, ra chiến lược, ... trước khi bắt tay viết chương trình.

```

7 t.goto(7, 2); t.goto(5, 0); t.goto(2, 0); t.goto(0, 2)
8 t.up(); t.goto(2, 2); t.down()
9 t.goto(2, 6); t.goto(4, 3); t.goto(2, 2)
10 t.up(); t.goto(5, 2); t.down()
11 t.goto(5, 4); t.goto(6, 2.5); t.goto(5, 2)

```

Tương tự, vẽ lại các hình ở Bài tập 5.4 bằng cách đặt lại hệ trục tọa độ.

5.6 Vẽ chuỗi. Con rùa cũng có thể giúp ta xuất ra các chuỗi trên cửa sổ với các lựa chọn vị trí, màu sắc, cách đóng hàng, kích thước và font chữ bằng hàm `write` như chương trình minh họa sau:

https://github.com/vqhBook/python/tree/master/lesson05/draw_string.py

```

1 import turtle as t
2 t.hideturtle()
3
4 t.up(); t.dot(5); t.write("Python 1")
5 t.goto(100, 0); t.color("red"); t.dot(5)
6 t.write("Python 2", align="center")
7 t.goto(50, 100); t.color("blue"); t.dot(15)
8 t.write("Python 3", align="right", font=("Arial", 30,
   ↪ "italic"))

```

Sau khi di chuyển con rùa đến nơi cần xuất và chọn màu phù hợp, ta có thể xuất chuỗi bằng hàm `write`. Đối số quan trọng dĩ nhiên là chuỗi cần xuất, ngoài ra, đối số có tên `align` xác định cách đóng hàng (`left` – trái (mặc định), `right` – phải, `center` – giữa). Ta cũng có thể xác định font chữ (là bộ 3 gồm tên font, kích thước và kiểu dáng) với đối số có tên `font`. Con rùa cũng hỗ trợ vẽ “dấu chấm” với hàm `dot` (đối số xác định kích thước dấu chấm).

Vẽ các hình tam giác trong Phần 3.2. Lưu ý: cần xuất ra đầy đủ thông tin (tên đỉnh, độ dài cạnh, số đo góc, ...) như trong hình.

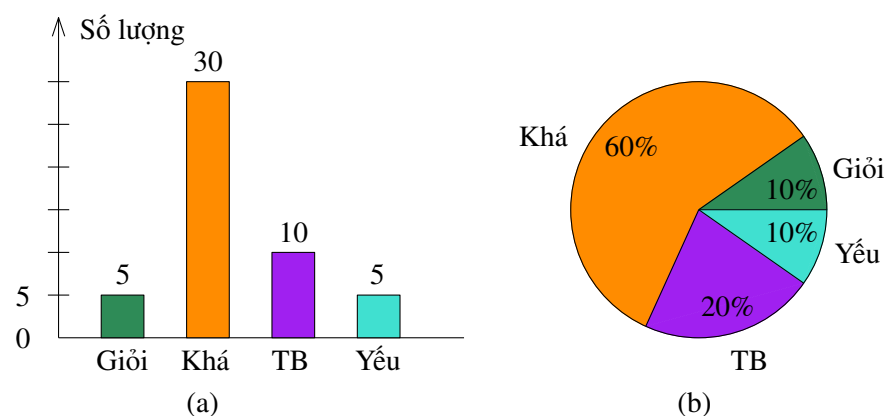
5.7 Nhập liệu từ turtle. Con rùa cho phép người dùng nhập chuỗi và số trực tiếp từ cửa sổ vẽ bằng hàm `textinput` và `numinput` (như minh họa ở Phần 5.5).

- (a) Tra cứu 2 hàm trên để biết rõ hơn.
- (b) Sửa lại mã nguồn các chương trình minh họa trong bài học cho người dùng nhập giá trị các tham số (thay vì gán “cứng” giá trị).
- (c) Viết chương trình cho người dùng nhập tên và xuất ra tên trang trọng trong một cái khung (một cái khung “thực sự” so với cái khung “giả” trong Phần 1.2). Đây có thể xem là phiên bản “đồ họa” của chương trình `hello.py` trong Phần 4.1. Bạn muốn trang trí thêm cho đẹp không?!

5.8 Vẽ biểu đồ. Mô tả số liệu một cách trực quan bằng các loại **đồ thị/biểu đồ** (graph/chart) là công việc quan trọng để tóm tắt, cảm nhận và hiểu số liệu. Chẳng hạn, với số liệu học lực của một lớp được cho như bảng sau:

Học lực	Giỏi	Khá	Trung bình	Yếu	Tổng
Số lượng (học sinh)	5	30	10	5	50
Tỉ lệ (%)	10%	60%	20%	10%	100%

Ta có thể mô tả số lượng bằng chiều cao của các thanh như **biểu đồ thanh** (bar chart) ở Hình (a) và mô tả tỉ lệ bằng diện tích (cũng là góc ở tâm) của các hình quạt như **biểu đồ quạt** (pie char) ở Hình (b) dưới.



Dùng Python vẽ 2 biểu đồ trên.

5.9 Viết chương trình cho người dùng nhập số liệu học lực,²¹ tính toán và vẽ biểu đồ thanh, biểu đồ quạt tương tự Hình (a), (b) ở Bài tập 5.8.²²

5.10 Vẽ logo Python như hình sau.²³



5.11 Bạn thử chạy các chương trình minh họa trong bài này bằng cách nhấp đúp file mã (chẳng hạn, file mã `star.py` ở Phần 5.2) sẽ thấy con rùa vẽ hình nhưng

²¹Dĩ nhiên là chỉ nhập số lượng. Chương trình tự tính tỉ lệ từ số lượng.

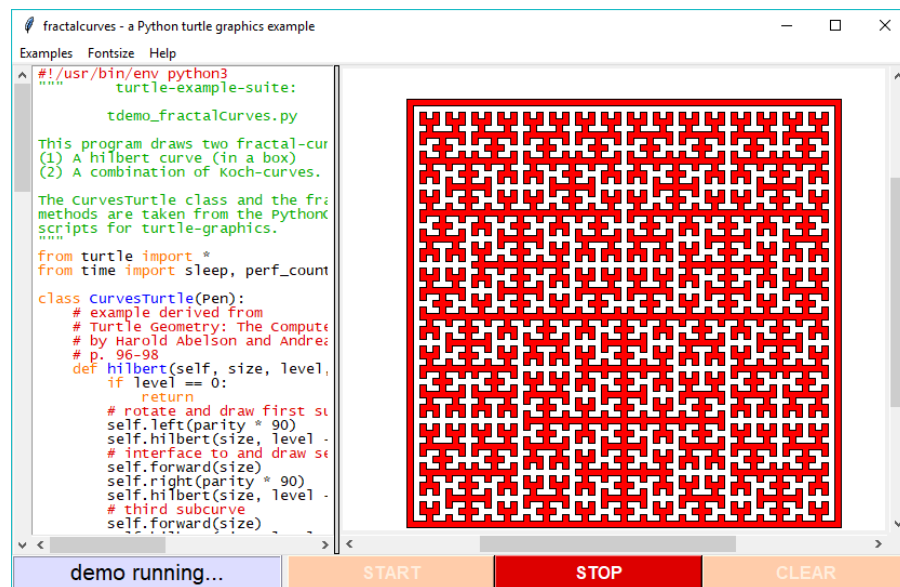
²²Bạn rất nên làm bài tập này. Nó cho thấy khác biệt giữa làm “thủ công” như Bài tập 5.8 và làm “tự động” bằng chương trình. “Tự động” chính là điều mà mọi chương trình hướng đến.

²³Nguồn: https://www.python.org/static/community_logos/python-logo-master-v3-TM.png. Không nhất thiết vẽ giống y chang. Logo này hơi xấu, bạn có thể vẽ đẹp hơn:) Sáng tạo thêm các mẫu khác và tạo logo cho mình. Bạn cũng so lại với “logo” trong Bài tập 4.3.

cửa sổ con rùa bị đóng ngay khi kết thúc. Xem lại cách “giữ màn hình” ở Bài tập 4.12 và thêm chức năng này vào các chương trình minh họa và bài tập trên.

5.12 IDLE cung cấp nhiều minh họa vẽ rùa rất hay và đẹp trong Help → Turtle Demo. Chẳng hạn, một trong các minh họa đó, vẽ “tấm thảm Hilbert” như hình dưới đây.

Chiêm ngưỡng các minh họa này. Nhớ rằng, chỉ nghĩa sơ mã nguồn thôi, bạn chưa hiểu gì nhiều đâu, mới bắt đầu học mà! Mục đích chính là thưởng lãm các tác phẩm mà con rùa mang lại và tạo cảm hứng rằng mình cũng có thể làm được như vậy nếu ... tiếp tục học các bài sau.²⁴



²⁴Thật ra, bạn sẽ viết được hay hơn! Mã nguồn của các minh họa đó được viết không tốt lắm!

Bài 6

Phá vỡ đơn điệu với lệnh chọn

Số phận của mỗi người được quyết định bởi các lựa chọn. Các chương trình cũng vậy. Trong bài này ta sẽ học cách điều khiển “số phận” của chương trình bằng các lệnh chọn. Ta cũng sẽ tìm hiểu biểu thức điều kiện, là phiên bản biểu thức của lệnh chọn; các toán tử luận lý, giúp ta mô tả các điều kiện lựa chọn phức tạp; và khối lệnh, là sự mở rộng của một lệnh. Khái niệm cực kì quan trọng của lập trình (và điện toán nói chung) là thuật toán cũng sẽ được tìm hiểu.

6.1 Luồng thực thi và tiến trình

Thứ tự thực thi các lệnh của chương trình được gọi là **luồng thực thi** (execution flow, control flow). Một quá trình thực thi (diễn tiến, trạng thái và kết quả thực thi) cụ thể của chương trình được gọi là **tiến trình** (process). Cho đến giờ, các chương trình của ta đều có chung một dạng luồng thực thi rất căn bản và đơn giản, là thực thi **tuần tự** (sequential) mỗi lệnh đúng một lần từ đầu đến cuối.¹ Chẳng có gì bất ngờ với một chương trình như vậy: chỉ có duy nhất một **con đường** (path) cho tiến trình của chương trình dạng này diễn ra.

Cuộc đời là một chuỗi các sự kiện, cũng như chương trình là dãy các lệnh, nếu chỉ có tuần tự thì đơn điệu và buồn tẻ biết bao. Có vô vàn những ngã rẽ, tạo nên vô vàn các con đường, tức vô vàn các khả năng để tiến trình cuộc đời diễn ra. Có lẽ, một trong những ngã rẽ lớn của cuộc đời ta là việc thi đậu đại học hay không. Nếu đậu, cuộc đời ta sẽ sang trang, nếu không, cuộc đời ta cũng sẽ sang trang ... khác.² Hãy bắt đầu với một chương trình Python có 2 khả năng như vậy:

```
1 https://github.com/vqhBook/python/tree/master/lesson06/exam.py  
điểm_thi = float(input("Bạn thi được bao nhiêu điểm? "))  
2 if điểm_thi < 5:  
3     print("Chúc mừng! Bạn đã rớt.")  
4 else:
```

¹Trừ khi một lệnh nào đó có lỗi thực thi thì Python sẽ dừng mà không thực thi các lệnh sau đó.

²Tôi không nghĩ là tệ hơn: “Ai bảo chăn trâu là khổ? Tôi cho rằng đi học khổ hơn trâu!”)

```
5 print("Chia buồn! Bạn đã đậu.")
```

Chương trình trên phá vỡ sự “đơn điệu” của các chương trình Python trước mà ta đã viết. Ở đây, có 2 khả năng được quyết định bởi điểm thi: tùy thuộc việc nó nhỏ hơn 5 hay không, ta sẽ rớt hoặc đậu. Lưu ý, khi bạn chạy chương trình, tức là thực hiện tiến trình thì chỉ có 1 trong 2 khả năng này xảy ra. Chẳng hạn, nhập 5 bạn đậu, nhập 4.5 bạn rớt.

Điều quan trọng cần học ở mã trên là cách mô tả các lựa chọn trong Python bằng **lệnh chọn** (selection statement).³ Cách viết điển hình là:

```
if <Cond>:
    <Suite1>
else:
    <Suite2>
```

Trong đó, `if` và `else` là các từ khóa, còn `<Cond>` là biểu thức luận lý mô tả một **điều kiện** (condition), mà giá trị `True` nghĩa là điều kiện đúng hay được thỏa hay xảy ra còn `False` nghĩa là điều kiện sai hay không thỏa hay không xảy ra. Các dấu hai chấm (`:`) ở 2 vị trí trên là bắt buộc và việc thụt đầu dòng cũng vậy.⁴ Do cách viết này mà lệnh chọn còn được gọi là **lệnh if** (if statement) và cách Python thực thi nó cũng khá rõ ràng: Python lượng giá `<Cond>`, nếu được giá trị đúng, Python thực thi `<Suite1>`, ngược lại (được giá trị sai), Python thực thi `<Suite2>`. Lưu ý là chỉ 1 trong 2 trường hợp được thực thi (mà không phải cả 2 hay 0 cái nào).

Vì việc lựa chọn tạo ra các khả năng thực thi (tức các luồng thực thi) khác nhau nên lệnh `if` là một trong những **cấu trúc điều khiển** (control structure) luồng thực thi, cụ thể là **cấu trúc chọn**, mà Python cung cấp. Ta sẽ học nhiều cấu trúc điều khiển khác trong các bài sau.

Trước khi tiếp tục, bạn nên phân biệt rõ ràng tiến trình với chương trình. Chương trình mô tả tất cả các khả năng, tức là tất cả các con đường với các ngã rẽ. Tiến trình là một chương trình được thực thi, đi qua các lựa chọn, hiện thực chỉ một khả năng thực thi. Theo nghĩa này, *chương trình là kịch bản của các tiến trình và một tiến trình quyết định số phận của mình bằng cách đưa ra các lựa chọn mà chương trình cho phép*.

6.2 Lệnh chọn và khối lệnh

Một trường hợp đặc biệt của cấu trúc chọn ở trên là chọn làm hay không dãy lệnh nào đó như minh họa sau:

```
1 https://github.com/vqhBook/python/tree/master/lesson06/exam2.py
2 điểm_thi = float(input("Bạn thi được bao nhiêu điểm? "))
3 if điểm_thi == 10:
```

³Các thuật ngữ khác là **lệnh điều kiện** (conditional statement), **lệnh rẽ nhánh** (branch statement).

⁴IDLE tự động thụt đầu dòng cho bạn. Nếu không, bạn dùng phím Tab (hoặc nên dùng 4 khoảng trắng, Space) để thụt đầu dòng.

```

3     print("Bạn đỗ thủ khoa!")
4     print("Liên hoan thôi")
5     print("Ăn, uống, ngủ, nghỉ")

```

Có 2 thứ quan trọng trong mã này. Một là, lệnh `if` không có `else` mà ta thường gọi là **if khuyết** (còn `if` có `else` là **if đủ**). Nếu điều kiện xảy ra, các lệnh thụt đầu dòng của `if`, còn gọi là **thân if** (`if body`) sẽ được thực thi, nếu không thì thân `if` không được thực thi. Lưu ý là các lệnh ngoài thân `if` luôn được thực thi mà không phụ thuộc vào điều kiện, chẳng hạn như lệnh xuất ở Dòng 5. Bạn chạy thử để thấy các khả năng thực thi khác nhau (2 khả năng hay 2 luồng thực thi). Trong tất cả các trường hợp, bạn đều phải “ăn, uống, ngủ, nghỉ”.

Hai là, đây các **lệnh thụt đầu dòng** (`indent`) cùng một mức được gọi là **khối lệnh** (`block`) hay **bộ lệnh** (`suite`). Chương trình trên có 2 mức thụt đầu dòng, tương ứng là 2 khối lệnh. Khối lệnh ngoài cùng được gọi là **khối chương trình** (`program block`) gồm 3 lệnh (lệnh gán ở Dòng 1,⁵ lệnh `if` ở Dòng 2-4, và lệnh xuất ở Dòng 5). Khối lệnh thứ 2 là thân `if` gồm 2 lệnh (lệnh xuất ở Dòng 3 và 4). *Các lệnh của một khối luôn cùng được thực thi tuần tự (từ đầu đến cuối, mỗi lệnh đúng 1 lần) hoặc cùng không được thực thi.*

Việc thụt đầu dòng “đúng”, như vậy, là rất quan trọng vì nó quyết định “cấu trúc” các khối lệnh. Lỗi cũng hay xảy ra với việc thụt đầu dòng (bạn thử thêm các khoảng trắng hoặc thay đổi việc thụt đầu dòng các lệnh ở mã trên và chạy thử) nhưng rất may là IDLE đã hỗ trợ ta nhiều để tránh các lỗi này: IDLE tự động thụt đầu dòng “hợp lý” và tự thay phím Tab bằng 4 ký tự Space như khuyến cáo của PEP 8.⁶

Vì có chứa khối lệnh bên trong (mà khối lệnh là dãy gồm 1 hoặc nhiều lệnh) nên lệnh `if` (cùng với các lệnh khác nữa mà ta sẽ học) được gọi là **lệnh phức** hay **lệnh ghép** (`compound statement`). Để phân biệt, các lệnh không có chứa lệnh khác bên trong được gọi là **lệnh đơn** (`simple statement`), như lệnh nhập/xuất (thật ra là lời gọi hàm `print/input`), lệnh gán (bao gồm lệnh gán tăng cường) mà ta đã biết và vài lệnh khác nữa.

Còn một kiểu lệnh `if` nữa, gọi là **if tầng** (`cascading if`) hay **if nối** (`chained if`), cho phép ta kiểm tra và thực thi một dãy nhiều trường hợp thay vì chỉ 1 trường hợp (có thì làm như `if` khuyết) hay 2 trường hợp (như `if` đủ). Cách viết của kiểu `if` này là:

```

if <C1>:
    <S1>
elif <C2>:
    <S2>
elif <C3>:
    <S3>

```

⁵Ta đã làm nhiều việc ở Dòng 1, nhập chuỗi, chuyển chuỗi sang số thực rồi gán, tuy nhiên ta chỉ gọi tên cái sau cùng, tức là gán.

⁶Trong IDLE ta có thể chọn Indentation Width trong thẻ Fonts/Tabs của hộp thoại Settings.

else:
 <Se>

Python lần lượt kiểm tra các điều kiện và thực thi lệnh tương ứng: nếu điều kiện <C1> đúng, thực thi các lệnh <S1> và xong lệnh `if`, nếu không thì kiểm tra điều kiện <C2>, nếu đúng thì thực thi các lệnh <S2> và xong, nếu không thì ..., sau cùng nếu điều kiện cuối cùng sai thì các lệnh của `else` (tức <Se>) sẽ được làm. Ta có thể nối thêm nhiều cặp điều kiện/các lệnh thực thi với khối `elif` và có thể không dùng khối `else` như trong lệnh `if` khuyết.

Thử chương trình “chỉ số BMI” sau để rõ hơn:

<https://github.com/vqhBook/python/tree/master/lesson06/BMI.py>

```

1 print("Hoan nghênh đo chiều cao, cân nặng, đo huyết áp, thử
   ↳ sức kéo!")
2
3 cân_nặng = float(input("Nhập cân nặng của bạn (kg): "))
4 chiều_cao = float(input("Nhập chiều cao của bạn (m): "))
5
6 BMI = cân_nặng / chiều_cao**2
7 print(f"Chỉ số BMI của bạn là: {BMI:.1f}")
8
9 if BMI < 18.5:
10     print("Thân hình hơi gầy một tí!")
11     print("Đề nghị ăn uống bồi dưỡng thêm.")
12 elif BMI < 25:
13     print("Thân hình hoàn toàn bình thường!")
14 elif BMI < 30:
15     print("Thân hình hơi béo một tí!")
16 else: # BMI >= 30
17     print("Thân hình không được bình thường!")
18     print("Đề nghị tập thể dục thường xuyên.")

```

Chương trình trên tính chỉ số khối cơ thể (body mass index, viết tắt BMI) theo công thức:

$$\text{BMI} = \frac{(\text{Cân nặng tính theo kg})}{(\text{Chiều cao tính theo mét})^2}$$

Sau đó đưa ra chẩn đoán dinh dưỡng dựa trên bảng phân loại sau:⁷

BMI	Phân loại dinh dưỡng
Dưới 18.5	Gầy
Từ 18.5 đến dưới 25	Bình thường
Từ 25 đến dưới 30	Béo
Từ 30 trở lên	Béo phì

⁷Theo khuyến cáo của WHO cho người lớn.

6.3 if lồng

Ta đã biết thân `if` (cũng như `elif`, `else`) là một khối lệnh. Đó là một dãy các lệnh bất kì. Trường hợp một trong các lệnh trong khối này lại là lệnh `if` thì ta có cấu trúc các lệnh `if` “lồng nhau” được gọi là **if lồng** (nested if). Chẳng hạn, cấu trúc `if` tầng ở trên có thể được viết lại bằng `if` lồng như sau:

```
if <C1>:
    <S1>
else:
    if <C2>:
        <S2>
    else:
        if <C3>:
            <S3>
        else:
            <Se>
```

Rõ ràng, trong trường hợp có nhiều “tầng” hay “mức lồng” thì ta nên dùng `if` tầng hơn `if` lồng như trong chương trình BMI ở trên.

Chương trình Python sau minh họa `if` lồng qua một trò chơi cờ bạc đơn giản.⁸ Python tung một đồng xu. Bạn không biết kết quả nhưng phải đưa ra lựa chọn mặt ngửa hay sấp. Nếu bạn chọn đúng mặt đồng xu ra, “bạn lên voi”; nếu chọn sai, “bạn xuống chó”.

— https://github.com/vqhBook/python/tree/master/lesson06/coin_toss.py —

```
1 import random
2
3 coin = random.randint(0, 1)
4 choice = input("Bạn chọn ngửa hay sấp(N/S)? ")
5 if coin == 1:
6     print("Đồng xu ra ngửa")
7     if choice == "N":
8         print("Bạn chọn ngửa")
9         print("Bạn lên voi!")
10    else:
11        print("Bạn chọn sấp")
12        print("Bạn xuống chó!")
13 else: # coin là 0
14     print("Đồng xu ra sấp")
15     if choice == "N":
16         print("Bạn chọn ngửa")
```

⁸Tài liệu này không khuyến khích cờ bạc. Tuy nhiên, nhiều quyết định trong cuộc đời ta là những canh bạc, chúng phụ thuộc nhiều vào may mắn (hay xui xẻo). Không gì rõ ràng và điển hình về các lựa chọn (và kết quả của chúng) hơn là cờ bạc.

```

17         print("Bạn xuống chó!")
18     else:
19         print("Bạn chọn sập")
20         print("Bạn lên voi!")

```

Module `random` cung cấp các dịch vụ liên quan đến ngẫu nhiên như tạo một số nguyên ngẫu nhiên bằng hàm `randint`. Hàm này nhận 2 đối số và tạo một số nguyên ngẫu nhiên trong phạm vi 2 đối số đó. Chẳng hạn, Dòng 3 trên tạo số ngẫu nhiên trong phạm vi `[0, 1]` tức là số ngẫu nhiên 0 hoặc 1. Số này được dùng để chỉ mặt ra của đồng xu với qui ước 1 là ngựa và 0 là sập.

Sau khi cho người dùng chọn mặt với chuỗi "N" là ngựa và "S" là sập, ta kiểm tra 4 khả năng bằng cấu trúc `if` lồng ở Dòng 5-20. Trước hết, ở khối chương trình, ta xét mặt đồng xu ra là 1 (ngựa) hay 0 (sập). Sau đó, bên trong khối của từng trường hợp (khối ứng với trường hợp 1 là Dòng 6-12 và khối ứng với trường hợp 0 là Dòng 14-20), ta xét lựa chọn của người dùng là "N" hay "S".

Cấu trúc `if` lồng có thể làm bạn hơi ngợp ban đầu (vì trong `if` có `if`), nhưng với cách nhìn tốt (trong `if` có thể là bất kì lệnh nào, mà `if` (khác) chỉ là một trường hợp thôi) và sự quen thuộc (sau khi đã đọc nhiều), bạn sẽ thấy đơn giản thôi. Chương trình trên cũng có thể viết ngược lại bằng cách xét lựa chọn của người dùng trước và mặt ra của đồng xu sau. Bạn thử xem.

6.4 Toán tử điều kiện và toán tử luận lý

Để giảm bớt sự phức tạp (và rườm rà) của việc lồng ghép, ta có thể dùng cách viết sau:

— https://github.com/vqhBook/python/tree/master/lesson06/coin_toss2.py —

```

1  import random
2
3  coin = random.randint(0, 1)
4  choice = input("Bạn chọn ngựa hay sập(N/S)? ")
5  print("Đồng xu ra " + ("ngựa" if coin == 1 else "sập"))
6  print("Bạn chọn " + ("ngựa" if choice == "N" else "sập"))
7  if (coin == 1 and choice == "N"
8      or coin == 0 and choice == "S"):
9      print("Bạn lên voi!")
10 else:
11     print("Bạn xuống chó!")

```

Trước hết là cách dùng **toán tử điều kiện** (conditional operator) ở Dòng 5 và 6. Đây là toán tử 3 ngôi, viết theo cú pháp:

`<E1> if <C> else <E2>`

và được Python lượng giá như sau: lượng giá biểu thức `<C>`, nếu được kết quả đúng, lượng giá biểu thức `<E1>` (mà không lượng giá `<E2>`) và kết quả của `<E1>` là kết

quả của toàn bộ biểu thức; ngược lại, lượng giá biểu thức <E2> (mà không lượng giá <E1>) và kết quả của <E2> là kết quả của toàn bộ biểu thức. Nói nôm na, ngữ nghĩa của **biểu thức điều kiện** (conditional expression) trên là: “nếu <C> đúng thì được <E1> còn không thì được <E2>”. Lưu ý, ở đây, `if` và `else` được dùng như là các toán tử.⁹ Đặc biệt lưu ý, toán tử điều kiện có độ ưu tiên thấp nhất trong Python nên ta phải dùng cặp ngoặc tròn như cách viết trong Dòng 5, 6 trên.

Biểu thức điều kiện thường giúp ta viết mã cô đọng hơn trong một số trường hợp, chẳng hạn, nếu không dùng biểu thức điều kiện thì Dòng 5 ở trên phải viết dài hơn bằng lệnh `if` như sau:

```
5 if coin == 1: print("Đồng xu ra ngựa")
6 else: print("Đồng xu ra sấp")
```

Ở đây, nhân tiện, tôi cũng giới thiệu cách viết “dồn” thân `if` lên cùng dòng với phần **tiêu đề** (header) (tức là phần chứa từ khóa `if`, biểu thức điều kiện và dấu `:`). Cách viết này có thể dùng khi thân `if` (hoặc các lệnh ghép sẽ học khác) ngắn. Dù giúp “tiết kiệm giấy” nhưng ta không nên dùng vì nó khó đọc hơn cách viết thông thường.

Điều quan trọng nữa trong chương trình trên là cách mô tả điều kiện phức tạp của lệnh `if` ở Dòng 7-8. Ta đã dùng các **toán tử luận lý** (boolean operator) hay **từ nối logic** (logical connective) `or` và `and`. Đây là các toán tử 2 ngôi, mô tả lần lượt các phép toán logic là **phép hoặc** (disjunction) và **phép và** (conjunction). Như mong đợi, Python cũng có toán tử 1 ngôi `not`, mô tả **phép phủ định** (negation). Các toán tử này được dùng với ý nghĩa thông thường như mô tả trong bảng sau:

<i>x</i>	<i>y</i>	<i>x and y</i>	<i>x or y</i>	<i>not x</i>
Sai	Sai	Sai	Sai	Đúng
Sai	Đúng	Sai	Đúng	Đúng
Đúng	Sai	Sai	Đúng	Sai
Đúng	Đúng	Đúng	Đúng	Sai

Để dễ nhớ: phủ định cho kết quả ngược lại, `x and y` chỉ đúng khi cả `x` và `y` đều đúng, `x or y` chỉ sai khi cả `x` và `y` đều sai. Theo nghĩa đó, toán tử `!=` là “phủ định” của `==` vì `x != y` nghĩa là `not x == y`. Các toán tử luận lý có độ ưu tiên cao hơn toán tử điều kiện nhưng thấp hơn các toán tử so sánh, trong đó, toán tử `not` có độ ưu tiên cao nhất, sau đó là `and` rồi `or`. Ta đã dùng qui tắc ưu tiên này trong biểu thức điều kiện của `if` ở Dòng 7-8 mà nếu không có nó ta phải dùng cặp ngoặc tròn để “bọc” 2 biểu thức con `and` lại.

Mã trên cũng cho thấy cách viết biểu thức điều kiện (và biểu thức nói chung) khi nó quá dài.¹⁰ Ta cũng có thể dùng dấu `\` để “xuống dòng” như cách viết sau:

⁹Dĩ nhiên, vì <E1>, <C>, <E2> là các biểu thức nên chúng cũng có thể là các biểu thức điều kiện, khi đó, ta sẽ có các “biểu thức điều kiện lồng”. Tuy nhiên, ta không nên lạm dụng cách viết này vì chương trình sẽ rất khó đọc.

¹⁰PEP 8 khuyên ta không nên viết mã có quá 72 kí tự trên một dòng vì dòng quá dài sẽ gây khó khăn cho việc đọc (nhất là khi nó dài quá khung cửa sổ).

```

7 if coin == 1 and choice == "N" \
8   or coin == 0 and choice == "S":

```

Lưu ý là ta không cần cặp ngoặc tròn. Tuy nhiên cách viết này không được ưa chuộng như cách viết đầu: dùng cặp ngoặc tròn để ép buộc Python cho phép viết biểu thức trên nhiều dòng (xem lại Phần 2.2).

Ta cần lưu ý là Python rất linh hoạt khi “hiểu” các giá trị luận lý. Khi cần giá trị luận lý (như trong điều kiện của `if` hay `elif`), Python chấp nhận mọi giá trị (của mọi kiểu) với qui ước: `False`, `None`, số nguyên 0, số thực 0.0, chuỗi rỗng `""` (và một số giá trị đặc biệt khác) được hiểu là sai; các giá trị khác được hiểu là đúng.

Ta cũng cần biết rằng, Python **lượng giá tắt** (short-circuit evaluation) các toán tử `and` và `or` chứ không “lượng giá đầy đủ”. Biểu thức `x and y` được Python lượng giá tắt như sau: lượng giá `x`, nếu được giá trị sai (theo nghĩa rộng đã nói trên) thì trả về giá trị của `x` (mà không lượng giá `y`); ngược lại, lượng giá và trả về giá trị của `y`. Biểu thức `x or y` được Python lượng giá tắt như sau: lượng giá `x`, nếu được giá trị đúng (theo nghĩa rộng) thì trả về giá trị của `x` (mà không lượng giá `y`), ngược lại, lượng giá và trả về giá trị của `y`. Nói nôm na, với cách lượng giá tắt, ngữ nghĩa của `x and y` là “`x`, nếu được thì `y`” và `x or y` là “`x`, nếu không thì `y`”. Khác với `and` và `or`, toán tử `not` thì thuần luận lý, theo nghĩa, `not x` trả về `True` nếu kết quả lượng giá `x` là sai (theo nghĩa rộng), ngược lại thì là `False`. Để hiểu rõ hơn thì lệnh xuất ở Dòng 5 trên có thể được viết lại là:

```

5 print("Đồng xu ra " + (coin == 1 and "ngửa" or "sấp"))

```

Tuy nhiên, cách viết này khá tối nghĩa và không được khuyến khích, mặc dù rất đáng giá để hiểu nó.

Cách lượng giá tắt thường được vận dụng có chủ ý trong khuôn mẫu “kiểm tra an toàn” như minh họa sau:

```

if mẫu_số != 0 and tử_số/mẫu_số < 1:
    print("Phân số nhỏ hơn 1")

```

Với cách viết này, chỉ khi `mẫu_số` khác 0 thì phép chia (trong biểu thức luận lý `tử_số/mẫu_số < 1`) mới được thực hiện, do đó tránh được lỗi chia cho 0 (`ZeroDivisionError`). Đây là cách viết tốt mà nếu không dùng nó, ta phải dùng `if` lồng dài hơn như sau:

```

if mẫu_số != 0:
    if tử_số/mẫu_số < 1:
        print("Phân số nhỏ hơn 1")

```

Cũng lưu ý là Python cho phép dùng cách viết **nôi** (chained) cho các toán tử so sánh như trong Toán. Chẳng hạn, thay vì viết `10 < x and x < 20` ta có thể viết

gọn hơn là $10 < x < 20$, hay biểu thức $a \leq b \leq c$ trong Toán có thể được viết trong Python là `a <= b <= c`. Tiện thể, toán tử gán cũng có thể được “nối” như vậy, chẳng hạn, thay vì viết `a = 10; b = 10` ta có thể viết gọn là `a = b = 10`.

6.5 Cuộc bỏ trốn khỏi cửa sổ của con rùa

Con rùa sau khi bị hành hạ quá nhiều ở Bài 5 đã quyết định bỏ trốn khỏi ... cửa sổ! Tuy nhiên, với tầm nhìn hạn chế, hắn không biết nên bò theo hướng nào: qua trái, qua phải, lên trên, hay xuống dưới. Thế là hắn ta quyết định ... chọn đại. Hơn nữa, sau mỗi lần lựa chọn và bò, hắn lại phân vân và lại quyết định chọn đại hướng bò trong bước tiếp theo. Chương trình sau tái hiện cuộc bỏ trốn khỏi cửa sổ của con rùa. Liệu rùa ta có bỏ trốn thành công hay không? Ta chạy chương trình và hồi hộp chờ xem.

```

1  https://github.com/vqhBook/python/tree/master/lesson06/turtle\_escape.py
2  import turtle as t
3
4  t.shape("turtle")
5  d = 20
6  while (abs(t.xcor()) < t.window_width()/2
7         and abs(t.ycor()) < t.window_height()/2):
8      direction = random.choice("LRUD")
9      if direction == "L":
10         t.setheading(180)
11     elif direction == "R":
12         t.setheading(0)
13     elif direction == "U":
14         t.setheading(90)
15     else: # direction == "D"
16         t.setheading(270)
17     t.forward(d)
18 print("Congratulations!")

```

Mã trên có dùng một cấu trúc lệnh lạ là `while`. Cách viết lệnh này giống như lệnh `if` khuyết nhưng cách thực thi có khác. Với lệnh `if` khuyết, khi điều kiện của `if` đúng, Python thực thi chỉ 1 lần thân `if` là xong; với lệnh `while`, khi điều kiện của `while` đúng, Python thực thi thân `while`, rồi lại kiểm tra điều kiện, nếu điều kiện vẫn đúng, Python lại thực thi thân `while`, rồi lại kiểm tra điều kiện, ... Nói cách khác, Python thực thi lặp lại thân `while` khi điều kiện của `while` vẫn còn đúng. Lệnh `while` này, do đó, được gọi là “lệnh lặp” và sẽ được tìm hiểu kĩ trong bài sau. Ở đây, với hàm `xcor/ycor` trả về hoành độ/tung độ của con rùa và hàm `window_width/window_height` trả về chiều rộng/chiều cao của cửa sổ, qua điều kiện của `while` (Dòng 6-7), con rùa sẽ phải kiên trì bò khi chưa bò ra khỏi cửa sổ.

Chương trình trên dùng hàm `choice` của module `random` để chọn ngẫu nhiên

một **kí tự** (character) trong một chuỗi. Ta chọn dùng các kí tự L, R, U, D cho các hướng trái, phải, lên, xuống.¹¹ Lưu ý, Python không có kiểu và cách viết riêng cho kí tự, kí tự được xem là chuỗi chỉ gồm ... 1 kí tự! Sau khi con rùa chọn ngẫu nhiên hướng bò ở Dòng 8, lệnh `if` ở Dòng 9-16 giúp con rùa đặt hướng đúng và Dòng 17 thực hiện việc bò (theo hướng đã chọn). Khi lệnh `while` kết thúc, tức là khi điều kiện của `while` sai (cũng là lúc con rùa đã bò ra khỏi cửa sổ), lệnh xuất ở Dòng 18 gửi lời chúc mừng đến con rùa.

Nếu thương con rùa, bạn hãy đặt tốc độ bò nhanh nhất (`t.speed("fastest")`) cho nó và/hoặc tăng bước bò (`d`) và/hoặc co cửa sổ nhỏ lại. Còn như ghét nó, muốn hành hạ nó, bạn biết phải làm gì rồi đấy?! Đừng để việc hành hạ con rùa trở thành thói quen tao nhả của bạn:) Nhưng nếu việc này giúp bạn lập trình khá hơn thì tốt thôi. *Hi sinh thân mình để giúp bạn học lập trình chính là sứ mệnh của con rùa vậy!*

6.6 Bài toán, thuật toán và mã giả

Hành hạ con rùa đủ rồi:) Giờ ta chuyển qua làm Toán chút. **Phương trình bậc hai** (quadratic equation) là phương trình có dạng:

$$ax^2 + bx + c = 0$$

Trong đó, kí hiệu hình thức x được gọi là ẩn; các số thực a, b, c được gọi là hệ số và $a \neq 0$. Giải phương trình bậc 2 là tìm nghiệm của phương trình trên khi cho các hệ số cụ thể, nghĩa là tìm số thực mà khi thay vào x ta được giá trị của vế trái là 0.

Ta đã biết cách giải một phương trình bậc 2 bất kì trong Toán Lớp 9 bằng công thức nghiệm (lập biệt thức Δ , ...). Ta nói rằng “giải phương trình bậc 2” là một bài toán và “công thức nghiệm” là một thuật toán để giải bài toán đó. Như vậy, **bài toán** (problem) là vấn đề cần giải quyết và **thuật toán** (algorithm) là cách giải quyết vấn đề. Điểm khác biệt quan trọng của các thuật ngữ này với các thuật ngữ bình dân (vấn đề/cách giải) là ở sự rõ ràng: bài toán là vấn đề, thuật toán là cách giải quyết **được xác định rõ** hay **được định nghĩa tốt** (well-defined).¹²

Bài toán được xác định rõ ràng qua **đầu vào** (input) và **đầu ra** (output). Đầu vào cho biết các dữ kiện được cho và đầu ra mô tả kết quả, lời giải cần tính/tìm/dựng của bài toán tương ứng với đầu vào. Lưu ý, cặp đầu vào/đầu ra chỉ mô tả yêu cầu (tức là “what to do”) của bài toán. Thuật toán được xác định rõ ràng qua các bước thực hiện để từ input cho ra output thỏa yêu cầu của bài toán. Thuật toán, như vậy, cho biết cách thực hiện (tức là “how to do”) để giải bài toán.

Ta có thể dùng ngôn ngữ tự nhiên như thông thường để mô tả thuật toán. Tuy nhiên, cách mô tả này thường rối rắm, dài dòng và mơ hồ. Cách mô tả tốt hơn là dùng ngôn ngữ tự nhiên ngắn gọn, rõ ràng, hướng thủ tục như trong các **công thức nấu ăn** (cooking recipe). Cách mô tả cô đọng, rõ ràng nhất là dùng ngôn ngữ Toán

¹¹Đây là các kí tự đầu của các chữ Left, Right, Up, Down.

¹²Điều nữa là các khái niệm bài toán và thuật toán đều hướng đến việc **tính toán** (computation), là thuật ngữ có nghĩa rất rộng như ta đã biết.

như trong các **công thức Toán** (mathematical formula). Kết hợp cả 2 cách, ngôn ngữ tự nhiên cô đọng với các kí hiệu và công thức Toán, ta có cách mô tả hay được dùng là **mã giả** (pseudocode). Chẳng hạn, “thuật toán giải phương trình bậc 2” có thể được mô tả bằng mã giả như trong bảng dưới đây.¹³

Input: phương trình bậc hai $ax^2 + bx + c = 0$ (a, b, c là các hệ số thực và $a \neq 0$)

Output: tập nghiệm của phương trình

Algorithm:

- Tính biệt thức $\Delta = b^2 - 4ac$
- Xét dấu Δ :
 - Nếu $\Delta > 0$ thì phương trình có hai nghiệm phân biệt:

$$x_1 = \frac{-b + \Delta}{2a}, x_2 = \frac{-b - \Delta}{2a}$$

- Nếu $\Delta = 0$ thì phương trình có nghiệm kép:

$$x_1 = x_2 = \frac{-b}{2a}$$

- Nếu $\Delta < 0$ thì phương trình vô nghiệm

Như tên gọi, không có tiêu chuẩn hay qui tắc cụ thể để viết mã giả. Tuy nhiên, ta cần nhớ là mã phải rõ ràng, cô đọng và dễ hiểu. Mã giả gần với con người hơn là máy và không có qui chuẩn nên máy không làm theo được (do đó được gọi là mã giả). Muốn máy làm theo được, ta phải viết cụ thể hơn nữa bằng ngôn ngữ qui chuẩn như các ngôn ngữ lập trình, khi đó, ta có mã thật (như vậy, “mã thật” chẳng qua là cách nói ví von cho mã nguồn của các chương trình viết theo ngôn ngữ lập trình nào đó). Chẳng hạn, mã giả trên được viết “thật” bằng mã Python như sau:

```

1 https://github.com/vqhBook/python/tree/master/lesson06/quad\_equation.py
2 print("Chương trình giải phương trình bậc 2: ax^2 + bx + c =
   ↳ 0 (a \u2260 0).")
3 a = float(input("Nhập hệ số a (a \u2260 0): "))
4 b = float(input("Nhập hệ số b: "))
5 c = float(input("Nhập hệ số c: "))
6
7 if (delta := b**2 - 4*a*c) > 0:
8     x1 = (-b + delta**0.5)/(2*a)
9     x2 = (-b - delta**0.5)/(2*a)
10    print("Phương trình có 2 nghiệm phân biệt:")
11    print(f"x1 = {x1:.2f}, x2 = {x2:.2f}.")
12 elif delta < 0:
13    print("Phương trình vô nghiệm.")

```

¹³Chỉnh lý từ “bảng công thức nghiệm của phương trình bậc hai”, SGK Toán 9 Tập 2 trang 44.

```

13 else:    # delta == 0
14     x = -b/(2*a)
15     print("Phương trình có nghiệm kép:")
16     print(f"x1 = x2 = {x:.2f}.")

```

Ở trên, tôi đã dùng một kĩ thuật viết gọn hay gặp trong lệnh `if` (và các trường hợp sẽ học khác) là “**toán tử hà mã**” (“walrus operator”) `:=`.¹⁴ Khác với dấu gán (`=`), toán tử `:=` không chỉ gán mà còn trả về giá trị được gán, và do đó nó có thể tham gia vào biểu thức lớn hơn. Nếu không dùng toán tử này thì ta phải “tồn thêm” 1 lệnh gán. Cụ thể, ta phải viết lại Dòng 6 như sau:

```

5 delta = b**2 - 4*a*c
6 if delta > 0:
7     # ...

```

Tóm lại, toán tử `:=` cho phép ta “gán và dùng luôn”. Cũng lưu ý, mặc dù toán tử này giúp ta “tiết kiệm” 1 dòng lệnh nhưng ta chỉ nên dùng khi phù hợp, tránh các trường hợp gây khó hiểu. Thậm chí, trong trường hợp mã trên, ta cũng không nên dùng vì nó “phá vỡ” sơ đồ thuật toán (có 1 bước tính Δ “hẳn hoi” trong thuật toán).¹⁵ Toán tử `:=` có độ ưu tiên thấp nhất trong các toán tử (và do đó ta cần cặp ngoặc tròn trong điều kiện của lệnh `if` ở Dòng 6.)

Khi đã có chương trình trên, ta có thể giải “tự động” mọi phương trình bậc 2 mà không cần giải “thủ công” nữa. Chẳng hạn, dùng chương trình trên (chạy và nhập liệu) giải phương trình $3x^2 + 5x - 1 = 0$ như sau:

```

Chương trình giải phương trình bậc 2: ax2 + bx + c = 0 (a ≠ 0).
Nhập hệ số a (a ≠ 0): 3
Nhập hệ số b: 5
Nhập hệ số c: -1
Phương trình có 2 nghiệm phân biệt:
x1 = 0.18, x2 = -1.85.

```

Việc xây dựng thuật toán (tìm ra cách làm và mô tả rõ ràng bằng mã giả) được gọi là **thiết kế thuật toán** (algorithm design); việc viết mã nguồn cho thuật toán được gọi là **hiện thực** hay **cài đặt thuật toán/chương trình** (algorithm/program implementation); việc chạy thử, kiểm tra thuật toán (sau khi cài đặt) được gọi là **kiểm thử thuật toán/chương trình** (algorithm/program/software testing). Các bước viết chương trình, như vậy, nên là: *xác định bài toán (input/output)*, *thiết kế thuật toán*, *cài đặt thuật toán (bằng mã Python)*, và *kiểm thử chương trình*; tức là *nghĩ trong đầu, viết ra giấy, làm trên máy và kiểm tra thử*.

¹⁴Là cách nói vui của đội ngũ phát triển Python vì kí hiệu `:=` trông giống cặp mắt và răng nanh của hà mã. Đây cũng là toán tử khá mới trong Python (có từ phiên bản Python 3.8).

¹⁵Tôi giới thiệu sớm ở đây vì ta sẽ dùng nó trong nhiều tình huống “hay” như sẽ thấy sau.

6.7 Lệnh pass và chiến lược thiết kế chương trình từ trên xuống

Ta đã biết Python có một giá trị đặc biệt là None mô tả giá trị “không giá trị”. Kì lạ tương tự, Python có lệnh đặc biệt pass mô tả việc “không làm gì cả”. Thật vậy, pass là một lệnh đơn hợp lệ và khi thực thi lệnh này, Python không làm gì cả. Cũng vậy, có nhiều triết lý đằng sau đó (như việc “không làm gì cả” có thể xem là “làm thình” và như vậy cũng là làm?!),¹⁶ nhưng quan trọng hơn, Python dùng nó để có một mô hình thực thi thuận tiện và thống nhất. Chẳng hạn, thay vì viết:

```
if <C>:
    <S>
```

ta cũng có thể viết:

```
if <C>:
    <S>
else:
    pass # TODO: sẽ cài đặt sau
```

Dĩ nhiên cách viết đầu thì ngắn gọn hơn nhiều nhưng nếu ta cho rằng chương trình cũng sẽ làm gì đó khi điều kiện <C> sai thì ta có thể viết theo cách thứ 2 mà mục đích là tạo **khung sườn** hay **dàn ý** chương trình (framework hay outline). Trong trường hợp này, pass sẽ là **lệnh giữ chỗ** (placeholder) mà ta sẽ thêm lệnh “thật” vào sau (như ghi chú cho thấy).

Cũng như cách ta vẽ một bức tranh: vẽ bố cục, tổng thể trước rồi chi tiết sau; cách ta thiết kế, qui hoạch, xây dựng mọi thứ nói chung đều (nên) như vậy. Thiết kế thuật toán và chương trình cũng không là ngoại lệ: ta *xác định ý tưởng, bố cục, tổng thể chung trước, sau đó là các phần (hay bước) lớn, sau cùng mới đến các phần nhỏ, chi tiết, cụ thể*. Cách thiết kế này được gọi là **thiết kế từ trên xuống** (top-down design) và ta nên ý thức vận dụng nó thành thói quen. Cũng vậy, việc rèn luyện thường xuyên sẽ giúp ta nâng cao kĩ năng này.

Tóm tắt

- Tiến trình là một quá trình thực thi cụ thể của chương trình, mà theo đó, thứ tự thực thi các lệnh xác định luồng thực thi.
- Các lệnh điều khiển luồng thực thi có khả năng thay đổi dạng thực thi căn bản là tuần tự mỗi lệnh đúng một lần từ đầu đến cuối chương trình.
- Lệnh if với các dạng khuyết, đủ, tăng cho phép chọn lựa các khả năng thực thi dựa trên các điều kiện.

¹⁶Như câu nói này của Debasish Mridha “Sometimes the most important thing to do is to do nothing.”

- Thân các lệnh ghép là khối lệnh, đó là dãy các lệnh được viết thụt đầu dòng như nhau và cùng được thực thi.
- Khi cần thực hiện việc chọn lựa lồng, tức là trong các lựa chọn lại có các lựa chọn, ta dùng `if` lồng, là kỹ thuật dùng lệnh `if` bên trong thân của lệnh `if` lớn hơn.
- Biểu thức điều kiện là phiên bản biểu thức của lệnh chọn, nó giúp ta viết mã cô đọng hơn trong một số trường hợp.
- Toán tử luận lý (`not`, `and`, `or`) giúp ta mô tả các điều kiện phức tạp. Chúng được lượng giá tất và có ngữ nghĩa không hoàn toàn giống các từ nối logic tương ứng. Python cũng hiểu giá trị luận lý theo nghĩa rất linh hoạt.
- Rất nhiều trường hợp lựa chọn là ngẫu nhiên. Module `random` hỗ trợ các dịch vụ liên quan đến ngẫu nhiên như: chọn một ký tự ngẫu nhiên trong chuỗi bằng hàm `choice` hay tạo một số nguyên ngẫu nhiên bằng hàm `randint`.
- Bài toán là vấn đề cần giải quyết được xác định rõ qua input/output. Thuật toán là cách giải quyết vấn đề được mô tả rõ ràng bằng mã giả và cài đặt cụ thể thành chương trình.
- Các bước viết chương trình là: xác định bài toán (input/output), thiết kế thuật toán, cài đặt thuật toán (bằng chương trình Python) và kiểm thử chương trình.
- Lệnh đơn `pass` là lệnh hợp lệ nhưng khi gặp nó Python không làm gì cả. Nó thường được dùng để tạo khung chương trình trong cách thiết kế chương trình từ trên xuống.

Bài tập

- 6.1 Trong Bài tập 4.4, bạn đã viết chương trình xuất bài hát “Happy birthday”. Sửa chương trình để nếu người dùng không nhập `<name>` (chỉ nhấn Enter, tức là nhập chuỗi rỗng) thì dùng chữ `you` thay cho `<name>`.
- 6.2 Trong Bài tập 4.5, bạn đã viết chương trình xuất đoạn trích “Roméo & Juliet”. Sửa chương trình để nếu người dùng không nhập tên `<Roméo>` và/hoặc `<Juliet>` thì dùng chữ `Roméo` và `Juliet` tương ứng.
- 6.3 Trong Bài tập 4.8, bạn đã viết chương trình đổi độ F sang độ C rồi đổi độ C sang độ F. Sửa chương trình, cho phép người dùng chọn “chế độ đổi” (F sang C hay C sang F) rồi thực hiện việc chuyển đổi tương ứng.
- 6.4 Viết chương trình xếp loại học lực của sinh viên từ điểm trung bình theo bảng sau đây:¹⁷

¹⁷Theo “Qui chế học vụ” năm 2016 của Trường ĐH KHTN, Tp. HCM.

Điểm trung bình	Đạt/Không đạt	Học lực
9 đến 10	Đạt	Xuất sắc
8 đến cận 9		Giỏi
7 đến cận 8		Khá
6 đến cận 7		Trung bình khá
5 đến cận 6		Trung bình
4 đến cận 5	Không đạt	Yếu
Dưới 4		Kém

6.5 Viết chương trình tính tiền điện sinh hoạt từ lượng điện tiêu thụ (kWh) theo cách tính lũy tiến với 6 bậc giá như sau:¹⁸

Bậc	kWh áp dụng	Giá (ngàn đồng/kWh)
1	0 – 50	1.678
2	51 – 100	1.734
3	101 – 200	2.014
4	201 – 300	2.536
5	301 – 400	2.834
6	401 trở lên	2.927

6.6 Viết chương trình cho nhập 3 số, xuất ra theo thứ tự tăng dần.

Gợi ý: Nên dùng cách viết nối các toán tử so sánh khi cần để mã “đẹp” hơn.

6.7 Viết chương trình cho nhập một số nguyên và cho biết số đó là số âm, số không, hay số dương.

Yêu cầu: không dùng lệnh `if` (mà dùng biểu thức điều kiện).

6.8 Thời điểm trong ngày được xác định bởi giờ, phút, giây theo hệ 24 giờ, nghĩa là từ 0 giờ : 0 phút : 0 giây đến 23 giờ : 59 phút : 59 giây. Viết chương trình cho nhập 2 thời điểm, kiểm tra tính hợp lệ, cho biết thời điểm nào trước thời điểm nào (hay trùng nhau) và cho biết tổng số giây trôi qua giữa hai thời điểm đó.

6.9 Thiết kế thuật toán và cài đặt chương trình giải phương trình bậc nhất “tổng quát”:

$$ax + b = 0 \quad (a \text{ có thể bằng } 0)$$

Gợi ý: xét 2 trường hợp $a = 0$ và $a \neq 0$. Trường hợp $a = 0$ có thể phải xét tiếp 2 trường hợp $b = 0$ và $b \neq 0$. Trường hợp $a \neq 0$ thường được gọi là phương trình bậc nhất.¹⁹

¹⁸Không tính thuế và các khoản “kì bí” khác. Bảng giá của tập đoàn điện lực Việt Nam, áp dụng từ ngày 20/3/2019.

¹⁹Xem chi tiết hơn trong SGK Toán 8 Tập 2.

Lưu ý: sau khi viết chương trình, bạn nên kiểm thử chương trình bằng cách chạy chương trình và thử giải các phương trình cụ thể nào đó (bằng cách nhập hệ số tương ứng) như:²⁰

$$\begin{array}{lll} 2x - 1 = 0; & 3 - 5y = 0; & -\frac{2}{5}x = 10; \\ x + 2 = x; & x - x = 0; & 2x + 4(36 - x) = 100. \end{array}$$

6.10 Thiết kế thuật toán và cài đặt chương trình giải phương trình trùng phương:

$$ax^4 + bx^2 + c = 0 \quad (a \neq 0).$$

Kiểm thử các phương trình sau:

$$\begin{array}{lll} x^4 - 13x^2 + 36 = 0; & 4x^4 + x^2 - 5 = 0; & 3x^4 + 4x^2 + 1 = 0; \\ x^4 - 1 = 0; & x^4 - 4x^2 = 0. & \end{array}$$

Gợi ý: ý tưởng chính là đưa (quy) về phương trình bậc 2.²¹

6.11 Thiết kế thuật toán và cài đặt chương trình giải hệ 2 phương trình bậc nhất 2 ẩn:

$$\begin{cases} ax + by = c \\ a'x + b'y = c' \end{cases}$$

Kiểm thử với các hệ phương trình sau:

$$\begin{array}{lllll} \begin{cases} x + y = 36 \\ 2x + 4y = 100 \end{cases} & \begin{cases} 3x - 2y = 0 \\ -3x + 2y = 0 \end{cases} & \begin{cases} x + 2y = 1 \\ 2x + 4y = 2 \end{cases} & \begin{cases} x - 3y = 2 \\ -2x + 5y = 1 \end{cases} & \begin{cases} 2x + y = 3 \\ x - y = 6 \end{cases} \end{array}$$

Gợi ý: có thể dùng các phương pháp như “phương pháp thế”, “phương pháp cộng đại số”.²²

6.12 Tiếp tục đọc IDLE: dùng các chức năng hỗ trợ việc viết lệnh hay soạn mã nguồn trong các menu Edit và Format (trong cửa sổ soạn thảo file mã). Thử nghiệm và tra cứu thêm (Help → IDLE Help hoặc search Google) để biết các chức năng tiện lợi.

²⁰Bạn tự “đưa” phương trình về phương trình bậc nhất rồi nhập hệ số phù hợp. Mặc dù ta có thể viết chương trình để “tự động” làm việc này luôn nhưng hiện giờ chương trình của ta chưa đủ “thông minh”.

²¹Xem chi tiết hơn trong SGK Toán 9 Tập 2 trang 54-55.

²²Xem chi tiết hơn trong SGK Toán 9 Tập 2.

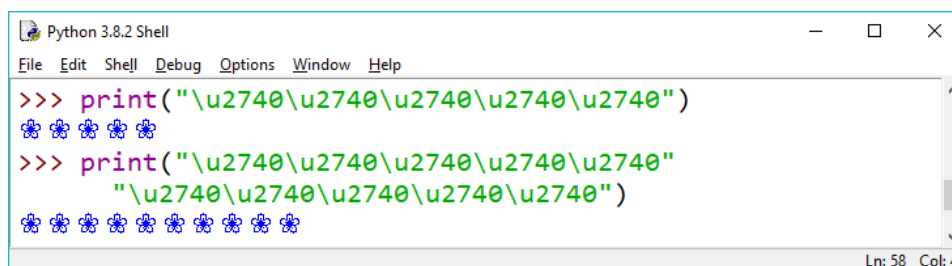
Bài 7

Vượt qua hữu hạn với lệnh lặp

Với con người, không gì chán hơn việc làm đi làm lại một việc quá bình thường nào đó; ta thích làm ít và làm những việc thú vị. Máy móc (hay Python) thì khác, chúng “thích” làm những việc đơn giản với số lượng lớn mà không hề phiền hà hay mệt mỏi. Như ta sẽ thấy, “lặp lại nhiều lần một việc đơn giản” không chỉ là sở trường mà còn là sức mạnh của máy móc (và Python). Bài học này chỉ cho bạn cách yêu cầu Python thực thi lặp lại một công việc nào đó. Bạn cũng học cách nhận dạng khuôn mẫu và tổng quát hóa công việc bằng kỹ thuật tham số hóa. Các lệnh điều khiển luồng thực thi là `break` và `continue` cũng được tìm hiểu.

7.1 Lặp số lần xác định với lệnh `for`

Thử minh họa sau:



```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
>>> print("\u2740\u2740\u2740\u2740\u2740")
* * * * *
>>> print("\u2740\u2740\u2740\u2740\u2740"
          "\u2740\u2740\u2740\u2740\u2740")
* * * * * * * * * *
Ln: 58 Col: 4
```

Lệnh đầu tiên xuất ra 5 đóa hồng;¹ không có gì bí hiểm ngoại trừ mã Unicode của “bông cúc” là `0x2740`. Ta cũng dễ dàng xuất ra “bó” 10 đóa cúc như lệnh thứ 2 bằng cách “chép-dán” (Ctrl+C, Ctrl+V), nhưng nếu ta muốn xuất ra “giỏ” 100 đóa cúc thì sao? Chẳng lẽ cũng chép-dán này, 1000 đóa thì sao? Tôi không nghĩ là bạn (hay ai đi nữa) đủ kiên nhẫn đâu. Bảo Python như sau:

```
1 >>> for i in range(100):
2     print("\u2740", end="")
```

¹Trông giống bông cúc hơn.

Tôi không chép kết xuất trong khung trình bày trên, nhưng như bạn chạy và thấy, 100 đóa cúc được xuất ra không thiếu đóa nào. Để xuất ra một “vườn” 1000 đóa hay thậm chí một “rừng” triệu đóa thì bạn biết cách viết rồi đó. (Mà nhớ phím tắt Ctrl+C để ngắt thực thi chứ một triệu đóa thì đếm mệt xui!) Tôi thực sự nể phục Python (và IDLE) về khoản cần cù và kiên trì này. *Khả năng làm lặp lại nhiều lần một việc đơn giản thực sự là một thế mạnh của Python (và máy móc).*

Ở trên, ta đã dùng **lệnh lặp** (iteration statement, looping statement) giúp thực thi lặp lại các việc nào đó với số lần lặp theo yêu cầu. Cụ thể, ta đã dùng **lệnh for** (for statement) của Python với cách viết:

```
for <Name> in range(<Expr>):
    <Suite>
```

Trong đó, <Expr> là một biểu thức nguyên với giá trị xác định số lần lặp lại khối lệnh <Suite>, được gọi là **thân lặp** (loop body). Hơn nữa, <Name> là một tên biến, được gọi là **biến lặp** (loop variable); dấu hai chấm (:) là bắt buộc; for, in là các từ khóa và range là hàm dựng sẵn mà ta sẽ biết kĩ hơn sau. Hiện giờ ta chưa dùng biến lặp nên tạm thời cứ dùng tên i, hoặc tốt hơn, ta có thể dùng tên _ để đỡ bận tâm như cách mọi người hay dùng.²

Xem lại cách vẽ hình vuông trong mã nguồn square.py ở Phần 5.1, ta đã phải lặp lại 4 việc giống nhau để vẽ một hình vuông. Dùng lệnh for ta có thể viết gọn hơn nhiều như sau:

— <https://github.com/vqhBook/python/tree/master/lesson07/square.py> —

```
1 import turtle as t
2
3 for _ in range(4):
4     t.forward(200)
5     t.left(90)
```

Như đã nói, lệnh for giúp ta lặp lại số lần cụ thể các lệnh nào đó và điều hay ho là ta lặp lại bao nhiêu lần cũng được như minh họa sau:

— <https://github.com/vqhBook/python/tree/master/lesson07/square2.py> —

```
1 import turtle as t
2 t.speed("fastest")
3
4 for _ in range(91):
5     t.forward(200)
6     t.left(91)
```

Trong chương trình trên, ta đã lặp lại 91 lần việc bắt con rùa bò tới và quay trái. Tôi đã cố ý quay góc 91° để các đường không trùng nhau (nếu không con rùa sẽ vẽ lặp lại nhiều lần một hình vuông).

²Nhớ rằng _ là một tên hợp lệ và trong chế độ tương tác được Python dùng để tham chiếu đến giá trị vừa tính (xem Phần 2.2), tuy nhiên, trong chế độ chương trình, nó “bình thường” như bao cái tên khác (mà ta lợi dụng “hình ảnh” của nó để kí hiệu cho thứ ta “không quan tâm”).

7.2 Công việc được tham số hóa và biến lập

Trong các chương trình trên, ta đã lặp lại nhiều lần cùng một công việc, nghĩa là lần lặp nào cũng làm công việc như nhau. Trường hợp phức tạp hơn, các công việc này thường khác nhau, tuy nhiên, chúng chỉ khác nhau chi tiết cụ thể còn giống nhau ở khuôn dạng tổng quát. Chẳng hạn, lệnh sau xuất ra các số nguyên từ 0 đến 99:

```
for i in range(100): print(i)
```

Nhân tiện, vì thân của lệnh `for` này rất ngắn nên ta cũng có thể viết trên cùng dòng với dấu `:`, tương tự như trường hợp ở Phần 6.4.

Công việc cần làm là: xuất số 0, xuất số 1, xuất số 2, ..., xuất số 99; là dãy 100 công việc “xuất một số” nhưng mỗi việc lại là một số khác nhau. Các công việc này có chung khuôn dạng là “xuất số nào đó” nhưng khác nhau chi tiết ở số được xuất. Để mô tả những việc này, ta **tham số hóa** (parameterization) chúng thành việc “xuất số i ” với i là số cần xuất, gọi là **tham số** (parameter) của công việc.

Ta cũng thường dùng kí hiệu $S(i)$ để chỉ công việc được tham số với S là dạng công việc (ở trên là “xuất”) và i là tham số (ở trên là “số cần xuất”). Bằng cách cho tham số nhận giá trị cụ thể, ta sẽ có công việc cụ thể. Chẳng hạn, ở trên, cho tham số i nhận giá trị 0, ta có công việc cụ thể là $S(0)$ tức là “xuất số 0” (công việc đầu tiên trong dãy 100 công việc trên) hay cho i nhận giá trị 99, ta có công việc cụ thể là $S(99)$ tức là “xuất số 99” (công việc cuối cùng trong dãy 100 công việc trên).

Quan trọng hơn, bằng cách cho tham số i nhận lần lượt các giá trị 0, 1, 2, ..., 99 ta có dãy 100 công việc ở trên. Đây chính xác là điều mà lệnh `for` ở trên thực hiện: lần lượt cho biến lập (<Name>) nhận các giá trị 0, 1, 2, ... đến trước số lần lặp (<Expr>), với từng lần, thực hiện thân lặp (<Suite>). Như vậy, bằng cách đặt số lần lặp phù hợp (ở đây là 100 để trước số lần lặp là 99), dùng biến lập làm tham số cho công việc là thân lặp (ở đây là biến lập i và công việc “xuất số i ” trong Python là `print(i)`), ta sẽ lặp lại được 100 lần công việc “vừa giống vừa khác” trên. Rõ ràng, ở đây, ta không nên dùng tên `_` cho biến lập như các minh họa ở phần trước vì nó đóng vai trò quan trọng.

Dĩ nhiên, công việc cần lặp lại (và do đó được tham số hóa và trở thành thân `for`) có thể phức tạp hơn (không chỉ 1 lệnh mà là khối gồm nhiều lệnh tùy ý). Quan trọng, ta cần thấy được sự giống nhau (khuôn dạng tổng quát) và khác nhau (chi tiết cụ thể) của thân lặp, nghĩa là, ta tham số hóa được chúng. Chẳng hạn, với thân lặp phức tạp hơn, ta có chương trình đếm khá thú vị như sau:

— <https://github.com/vqhBook/python/tree/master/lesson07/counter.py> —

```
1 import turtle as t
2 import time
3
4 t.hideturtle(); t.tracer(False)
5 for i in range(100):
6     t.clear()
```

```

7     t.write(i, align="center", font=("Arial", 150, "normal"))
8     t.update()
9     time.sleep(1)

```

Module `turtle` lại được dùng như mọi khi, ngoài ra, module `time` cung cấp hàm `sleep` giúp chương trình tạm nghỉ hay “ngủ” (tức là đợi) một khoảng thời gian (đối số là thời gian cần đợi, tính theo giây, như 0.5 là nửa giây).³ Bạn có thể cho chương trình đếm các số từ 1 đến 100 (thay vì từ 0 đến 99) không? Phải dừng lại và làm cho được rồi mới đi tiếp nhé.⁴ Dĩ nhiên, biến lặp cũng là biến nên như mọi biến trong Python, bạn có thể đặt tên tùy ý (không nhất thiết tên `i`) và nó có thể được dùng trong các biểu thức phức tạp hơn như `i + 1`, (thay vì chỉ là `i` như trong các lệnh xuất trên).

Các công việc cùng khuôn dạng nhưng khác nhau chi tiết còn được gọi là các **biến thể** (variant) và việc lặp lại nhiều lần các biến thể là việc rất hay gặp của Tự nhiên. Chẳng hạn, tham số hóa chiều dài cạnh hình vuông được vẽ, ta có chương trình vẽ hình khá đẹp sau:

<https://github.com/vqhBook/python/tree/master/lesson07/square3.py>

```

1  import turtle as t
2  t.speed("fastest")
3
4  for i in range(200):
5      t.forward(i)
6      t.left(90)

```

Hoặc kết hợp với xoay, sửa Dòng 6 trên thành `t.left(91)`, ta có hình khác cũng rất đẹp; hoặc sửa thành `t.left(i)`, ta được ... “hình chẳng biết là hình gì luôn”: Các hình kết xuất này đều có đặc điểm là trông chúng rất phức tạp (và “đẹp”) nghĩa là khó lòng vẽ tay nhưng lại có “kiến trúc” rất đơn giản: chúng chỉ là sự *lặp lại nhiều lần một vài bước cơ bản nào đó với các biến thể* (ở trên là vẽ đoạn thẳng tới và xoay góc). Ta không thể vẽ tay nếu số lần lặp lại nhiều, nhưng Python (và con rùa) không ngại làm điều này (và làm cực kì chính xác không có sai lầm do mỗi một gì nhé). Tương tự, bạn hãy viết lại chương trình vẽ “đường xoắn ốc vàng” trong mã nguồn `golden_spiral.py` (Phần 5.3) cho gọn hơn bằng lệnh `for`.⁵

Thật ra, ngoài giá trị “kết thúc” (stop), hàm `range` cũng cho phép ta chỉ định giá trị “bắt đầu” (start, thay vì 0) và giá trị “tăng thêm” (step, thay vì 1) như minh họa sau:

³ Bạn có thể đoán được `turtle.clear()` làm gì, còn không, tra cứu nhé.

⁴ Nếu sau khoảng một tiếng suy nghĩ, mày mò và thử nghiệm mà vẫn không được thì ... đốt sách, nghỉ luôn đi nhé:) Hãy ... quãng gánh lo đi và vui sống:) Giãn thôi, luyện lại từ đầu, một ngày không được thì ba bốn ngày!

⁵ Cũng vậy nhé, làm không được thì đốt sách đi:) Sau khi thử xong thì có thể xem mẫu ở GitHub https://github.com/vqhBook/python/tree/master/lesson07/golden_spiral.py.

```

1 >>> for i in range(10, 20, 2):
2     print(i, end=" ")
10 12 14 16 18

```

Bạn tra cứu thêm để “rõ sơ” nhé. Ta sẽ tìm hiểu kĩ hơn hàm `range` và lệnh `for` sau.

7.3 Lập linh hoạt với lệnh `while`

Ta có thể vẽ lại “đường xoắn ốc vàng” bằng lệnh `for` rất gọn. Tuy nhiên, nếu dùng `for` để vẽ lại “đường xoắn ốc Fibonacci” thì khó hơn. Lí do là vì bán kính đường tròn của mỗi lần lặp thay đổi khó mô tả theo biến lặp. Ta sẽ dùng cấu trúc lặp khác linh động hơn, **lệnh `while`** (while statement), để vẽ đường xoắn ốc Fibonacci như sau:

```

1 https://github.com/vqhBook/python/tree/master/lesson07/Fibonacci\_spiral.py
import turtle as t
2
3 x, y = 0, 1
4 while x <= 377:
5     t.circle(x, 90)
6     x, y = y, x + y
7
8 t.hideturtle()

```

Trước hết, ta đã dùng cặp số và cặp biến (nhớ lại dấu phẩy) trong chương trình trên. Lệnh gán ở Dòng 3 là gán cặp giá trị 0, 1 cho cặp biến `x, y` mà có nghĩa là lần lượt gán 0 cho `x`, 1 cho `y` (điều này có thể xem là cách viết gọn cho `x = 0; y = 1`). Kỹ thuật gán này lại được dùng ở Dòng 6, nhưng lần này nó không còn tương đương với cách viết `x = y; y = x + y` nữa. Lí do là trong lệnh “gán cặp” ở Dòng 6, Python lượng giá “cặp biểu thức” bên phải trước mà khi đó giá trị của `x`, `y` là các giá trị “cũ”; sau đó `x, y` mới được “đồng thời” cập nhật giá trị mới.

Quan trọng hơn, ta đã dùng lệnh lặp `while` của Python với cách viết:

```

while <Cond>:
    <Suite>

```

Cách viết tương tự như `if` khuyết (với từ khóa `while` thay cho `if`) nhưng cách thực thi thì khác: Python lượng giá điều kiện của `while` (biểu thức `<Cond>`), nếu đúng, Python thực thi thân `while` (khối `<Suite>`), rồi lại kiểm tra điều kiện, nếu vẫn đúng, Python lại thực thi thân `while`, rồi lại kiểm tra điều kiện, ..., cho đến khi điều kiện không còn đúng (tức là sai) thì Python kết thúc (thực thi xong) lệnh `while`. Nói cách khác, Python thực thi lặp lại thân `while` khi điều kiện của `while` vẫn còn đúng. Lưu ý là thân `while` có thể được thực thi 0 lần (tức là không thực thi), 1 lần, 2 lần, ... hay thậm chí vô hạn lần tùy theo “diễn biến” của điều kiện. (Để phân biệt thì nhớ là điều kiện của `if` chỉ được kiểm tra 1 lần và thân `if` chỉ có thể được thực

thi 0 hoặc 1 lần). Bạn xem lại lệnh `while` trong mã nguồn `turtle_escape.py` ở Phần 6.5 để hiểu rõ lại nhé.

Với cách thực thi như trên, *lệnh while mang lại cơ chế lặp với sự linh hoạt tối đa vì chính ta (lập trình viên) sẽ tự mình điều khiển việc lặp qua điều kiện lặp và các lệnh phù hợp trong thân while*. Chẳng hạn, lệnh `for` xuất ra các số chẵn trên có thể được viết lại “tương đương” bằng lệnh `while` như sau:

```

1 >>> i = 10
2 >>> while i < 20:
3     print(i, end=" ")
4     i += 2
10 12 14 16 18

```

Thật sự thì ta có thể viết lại mọi lệnh `for` bằng lệnh `while` nhưng trong các trường hợp lặp đơn giản (như lặp lại số lần xác định cùng một công việc hay lặp lại các việc được tham số hóa đơn giản theo biến lặp) thì dùng lệnh `for` sẽ tự nhiên và gọn gàng hơn.

Ngữ nghĩa của lệnh `while` thường được nói rõ là WHILE-DO vì điều kiện lặp được kiểm tra trước rồi mới làm thân lặp (nếu điều kiện lặp đúng). Một cấu trúc lặp hay gặp khác, DO-WHILE:

```

DO:
    <S>
WHILE <C>

```

lại làm thân lặp <S> trước rồi mới kiểm tra điều kiện lặp <C> (nếu điều kiện lặp đúng thì lại làm thân lặp, rồi kiểm tra điều kiện lặp, ...). Python không có lệnh trực tiếp cho cấu trúc này nhưng ta có thể “hiện thực” nó (dài hơn một chút) bằng lệnh `while` như sau:

```

<S>
while <C>:
    <S>

```

Còn một cấu trúc lặp hay gặp khác nữa mà Python không hỗ trợ là REPEAT-UNTIL:

```

REPEAT:
    <S>
UNTIL <C>

```

Như tên gọi, thân lặp sẽ được làm lặp lại cho đến khi điều kiện đúng thì dừng, nghĩa là, làm thân lặp <S> rồi kiểm tra điều kiện lặp <C>, nếu điều kiện đúng thì kết thúc (xong lệnh lặp), nếu sai thì lại làm thân lặp, rồi kiểm tra điều kiện lặp, ... Dùng lệnh `while` ta có thể hiện thực cấu trúc lặp này như sau:

```

<S>
while not(<C>):
    <S>

```

7.4 Vòng lặp lồng

Bạn chắc đã nghe qua bài toán cổ sau đây:

“Vừa gà vừa chó, bó lại cho tròn.
Ba mươi sáu con, một trăm chân chẵn.
Hỏi có bao nhiêu gà, bao nhiêu chó?”

Gọi x là số gà và y là số chó, ta có x, y là nghiệm của hệ phương trình:

$$\begin{cases} x + y = 36 \\ 2x + 4y = 100 \end{cases} \quad \text{với } x, y \text{ là các số nguyên và } 1 \leq x, y \leq 36$$

Bạn chắc cũng đã biết cách giải của Toán với các phép “biến đổi đại số” để đưa hệ phương trình trên về dạng “tương đương” nhưng đơn giản hơn, cụ thể:⁶

$$\begin{cases} x + y = 36 \\ 2x + 4y = 100 \end{cases} \Leftrightarrow \begin{cases} 2x + 2y = 72 \\ 2x + 4y = 100 \end{cases} \Leftrightarrow \begin{cases} x + y = 36 \\ 2y = 100 - 72 \end{cases} \Leftrightarrow \begin{cases} y = 14 \\ x = 36 - 14 = 22 \end{cases}$$

Vậy số gà là 22 con và số chó là 14 con. Ta cũng có thể dùng Python để giải bằng cách “mò nghiệm”. Cụ thể, ta lần lượt kiểm tra tất cả các trường hợp có thể của x, y (nhớ rằng x, y là các số nguyên từ 1 đến 36), xem trường hợp nào thỏa mãn yêu cầu của bài toán (tức thỏa hệ phương trình trên). Chương trình sau thực hiện công việc này:

```

1 https://github.com/vqhBook/python/tree/master/lesson07/chicken\_dog.py
2 for x in range(1, 37):
3     for y in range(1, 37):
4         if x + y == 36 and 2*x + 4*y == 100:
5             print(f"Số gà là: {x}, số chó là: {y}.")

```

Khi chạy chương trình, Python cũng giúp ta tìm ra số gà là 22 và số chó là 14. Bạn cho rằng cách làm này (“mò nghiệm”) không có gì hay ho (và “vi diệu”) như cách biến đổi đại số. Bạn ... đúng rồi! Nhưng, tin tôi đi, rất nhiều trường hợp ta không dễ dàng (thậm chí là không thể) biến đổi đại số được, nghĩa là không giải bằng Toán được. Nhiều trong số đó vẫn có thể mò nghiệm được.⁷

Điều quan trọng hơn ta học được từ chương trình trên là về cách dùng và sức mạnh của các **vòng lặp lồng** (nested loop). Không có gì mới về mặt ngôn ngữ, ta đã biết thân của các vòng lặp là một khối lệnh và nó có thể gồm số lượng và loại lệnh tùy ý, mà nếu có một lệnh nào đó lại là lệnh lặp thì ta có cái gọi là vòng lặp lồng. Tuy nhiên về mặt ứng dụng, *các vòng lặp lồng mang lại nhiều điều mới mẻ và có sức mạnh ghê gớm*. Cũng vậy, bạn có thể hơi ngợp lúc đầu, nhưng sẽ nhanh chóng quen thuộc nếu luyện tập nhiều.

⁶SGK Toán 9 Tập 2.

⁷Bạn sẽ thấy nhiều trường hợp Python “giải cứu” Toán trong các bài sau.

Thật ra, ta không nhất thiết dùng vòng lặp lồng để giải bài toán trên. Nhận xét rằng tổng của gà và chó là 36 nên y luôn bằng $36 - x$, do đó ta có thể “mò nghiệm” trên ẩn x thôi (thay vì trên cặp ẩn (x, y)) như sau:

https://github.com/vqhBook/python/tree/master/lesson07/chicken_dog2.py

```

1 for x in range(1, 37):
2     y = 36 - x
3     if 2*x + 4*y == 100:
4         print(f"Số gà là: {x}, số chó là: {y}.")

```

Cách làm này thật sự hiệu quả hơn (tức là chương trình chạy “nhanh hơn”) và cách viết cũng đơn giản, ngắn gọn hơn. Tuy nhiên bạn phải làm Toán chút đỉnh (thật ra ta đã dùng phương pháp “khử ẩn” trong cách cài đặt trên). Cách làm đầu (với vòng lặp lồng) thì kém hiệu quả hơn chút xíu và cách viết cũng rắc rối hơn nhưng tự nhiên và thoải mái hơn (ta cứ “sinh và thử” thôi).⁸ Việc chuyển các vòng lặp lồng về vòng lặp đơn (tức là không lồng), thường được gọi là “**khử lồng**”, như vậy, không nên được đặt nặng quá mức. Bạn *không nên sợ vòng lặp lồng* bởi “*vẽ bề ngoài*” của nó mà nên dùng nó trong nhiều trường hợp vì nó thường giúp việc viết chương trình đơn giản hơn.

Ta đã “vẽ” hình chữ nhật bằng các kí tự trong mã `string_rectangle.py` ở Phần 4.5. Tuy nhiên, ta không dễ dàng làm như vậy với các hình phức tạp hơn như hình tam giác vuông cân sau:

```

*
**
***
****

```

Đây là hình tam giác vuông cân có “chiều cao” là 4. Chương trình sau đây vẽ hình tam giác vuông cân như vậy với “chiều cao” do người dùng nhập:

<https://github.com/vqhBook/python/tree/master/lesson07/triangle.py>

```

1 char = input("Nhập kí hiệu vẽ: ")
2 n = int(input("Nhập chiều cao: "))
3 for i in range(1, n + 1):
4     for _ in range(i):
5         print(char, end=" ")
6     print() # xuống dòng

```

Ý tưởng khá đơn giản như sau:

Ta lặp qua từng dòng $i = 1, \dots, n$:

- Với dòng i , ta lặp i lần việc xuất ra kí hiệu vẽ (dấu `*` chẳng hạn)
- Xuống dòng

⁸Thật ra, với số lượng trường hợp phải kiểm tra ít như trường hợp này (chỉ $36 \times 36 = 1296$ trường hợp) thì thời gian Python làm là không đáng kể.

Từ ý tưởng trên, ta thấy rõ cấu trúc 2 vòng lặp lồng nhau. Vòng lặp ngoài lặp qua các dòng (i là dòng thứ) và vòng lặp trong dùng để xuất đủ i kí hiệu vẽ ứng với từng dòng i . Lưu ý, khi các vòng lặp lồng nhau, ta cần đặt tên khác nhau cho các biến lặp (nếu không thì lộn xộn lắm:)) Hơn nữa, như đã biết, hàm xuất `print` luôn tự động thêm xuống dòng, trừ khi ta đặt lại đối số `end`.

Ta cũng có thể khử lồng để được chương trình viết đơn giản hơn như sau:

<https://github.com/vqhBook/python/tree/master/lesson07/triangle2.py>

```

1 char = input("Nhập kí hiệu vẽ: ")
2 n = int(input("Nhập chiều cao: "))
3 for i in range(1, n + 1):
4     print(char * i)

```

7.5 Điều khiển chi tiết lệnh lặp

Thử chương trình sau:

https://github.com/vqhBook/python/tree/master/lesson07/perfect_square.py

```

1 a, b = 10, 100
2 for i in range(a, b + 1): # tìm số chính phương trong [a, b]
3     if i % 2 == 0:
4         pass # TODO: thay bằng continue để bỏ qua số chẵn
5
6     if int(i**0.5)**2 == i: # i là số chính phương
7         print(i)
8         pass # TODO: thay bằng break nếu chỉ muốn tìm một số
9
10 print("Done!")

```

Chương trình này tìm **số chính phương** (perfect square number), tức là bình phương của một số nguyên, trong phạm vi từ a đến b . Chẳng hạn, với a được gán 10 và b được gán 100 như Dòng 1 thì chương trình tìm ra các số 16, 25, 36, 49, 64, 81, 100. Để kiểm tra một số có là số chính phương ta đã dùng mẹo là lấy phần nguyên của căn và bình phương lên, tuy nhiên, ta có thể kiểm tra đơn giản hơn bằng cách dùng điều kiện sau, dễ hiểu hơn, cho lệnh `if` ở Dòng 6: `(i**0.5).is_integer()`.

Bây giờ nếu ta thay lệnh đơn `pass` ở Dòng 4 bằng lệnh đơn `continue` (lệnh này chỉ gồm một từ khóa tương tự như lệnh `pass`) thì như ghi chú cho thấy: chương trình bỏ qua các số chẵn. Thay và chạy thử ta có kết quả là các số 25, 49, 81 như mong đợi. Lệnh `continue` là một lệnh giúp điều khiển luồng thực thi của vòng lặp; cụ thể, khi gặp lệnh này trong vòng lặp, Python sẽ bỏ qua (không thực thi) các lệnh bên dưới của thân lặp và chuyển qua kiểm tra điều kiện cho lần lặp mới. Lệnh này, như vậy, thường được dùng trong cấu trúc lặp “tìm có bỏ qua”. Lưu ý, lệnh `continue` phải nằm trong vòng lặp (nếu không Python sẽ báo lỗi cú pháp) và có tác dụng trên vòng lặp “gần nhất” chứa nó.

Một lệnh đơn khác, lệnh `break`, rất giống lệnh `continue` chỉ trừ việc: khi gặp lệnh này, Python sẽ thoát ra khỏi (kết thúc) vòng lặp “gần nhất” chứa nó. Chẳng hạn, nếu ta thay lệnh `pass` ở Dòng 8 bằng lệnh `break` thì như ghi chú cho thấy: chương trình chỉ tìm số chính phương đầu tiên. Thay và chạy thử ta có kết quả là số 25 (nếu “bật” `continue` ở Dòng 4) hoặc số 16 (nếu “không bật” `continue`). Lệnh `break`, như vậy, thường được dùng trong cấu trúc lặp “tìm thấy là được (dừng)”. Minh họa này cũng cho thấy cách dùng lệnh `pass` với khung chương trình như ta đã biết ở Phần 6.7.

Thật lạ kì, các lệnh lặp (`for`, `while`) cũng có **phần else** (else part, else clause) tùy chọn. Mặc dù cú pháp như trong lệnh `if` nhưng ngữ nghĩa hoàn toàn khác: không phải “ngược lại (còn lại) thì” (otherwise) mà là “(lặp) xong rồi thì” (then). Cũng hơi “tào lao” nhưng nó cũng hữu ích trong một số tình huống như minh họa sau:

```

1  https://github.com/vqhBook/python/tree/master/lesson07/perfect\_square2.py
2  a, b = 10, 100
3  for i in range(a, b + 1):
4      if i % 2 == 0:
5          continue
6
7      if int(i**0.5)**2 == i: # i là số chính phương
8          print(i)
9          break
10
11 else:
12     print("Không tìm thấy số nào!")
13
14 print("Done!")

```

Chương trình này tìm số chính phương lẻ đầu tiên trong phạm vi $[a, b]$ như đã biết. Khác biệt là: chương trình sẽ có thông báo nếu không tìm thấy (không có số chính phương lẻ nào trong phạm vi). Điều này có được nhờ phần `else` (Dòng 10-11) của lệnh `for` (lưu ý, `else` này là của `for` chứ không phải của `if` ở Dòng 6 do cách thụt đầu dòng).

Khi các lệnh lặp kết thúc một cách “tự nhiên”, nếu lệnh lặp có phần `else`, Python sẽ thực thi khối lệnh của `else`. Trường hợp các lệnh lặp kết thúc một cách “khiên cưỡng”, như kết thúc bằng lệnh `break` trong thân lặp (và một số cách khác ta sẽ biết sau), Python sẽ không thực thi phần `else`. Trong chương trình trên, vòng lặp `for` sẽ kết thúc “khiên cưỡng” khi gặp lệnh `break` ở Dòng 8, mà điều này chỉ xảy ra khi tìm thấy số chính phương (đầu tiên) bởi lệnh `if` ở Dòng 6; khi đó, phần `else` sẽ không được thực thi. Ngược lại, nếu không có số chính phương (lẻ, trong phạm vi) thì điều kiện của `if` ở Dòng 6 không bao giờ thỏa nên lệnh `break` không được thực thi nên vòng lặp sẽ kết thúc “tự nhiên”; khi đó, phần `else` sẽ được thực thi và lệnh xuất ở Dòng 11 sẽ đưa ra thông báo “không tìm thấy số nào”. Lưu ý, dù kết thúc thế nào đi nữa (“tự nhiên” hay không), lệnh kế tiếp vòng lặp luôn được

thực thi nên thông báo “Done!” luôn được xuất ra khi chương trình chạy (lệnh xuất ở Dòng 13).

7.6 Lặp vô hạn

Bạn có cho rằng thời gian là vĩnh hằng? Chạy chương trình “đồng hồ” sau sẽ rõ:

<https://github.com/vqhBook/python/tree/master/lesson07/clock.py>

```

1 import turtle as t
2 import time
3
4 t.hideturtle(); t.tracer(False)
5 while True:
6     now = time.localtime()
7     time_str = "%02d:%02d:%02d" % (now.tm_hour, now.tm_min,
8     ↪ now.tm_sec)
9     t.clear()
10    t.write(time_str, align="center", font=("Arial", 100,
11    ↪ "normal"))
12    t.update()
13    time.sleep(0.1)

```

Cấu trúc chương trình này rất giống với chương trình “đếm” ở Phần 7.2. Tuy nhiên, thay vì đếm số, ta “đếm thời gian”.) Trước hết, ta dùng hàm `localtime` của module `time` để lấy về thời điểm hiện tại (“now”), sau đó truy cập các “thuộc tính” `tm_hour`, `tm_min`, `tm_sec` để lấy về giờ, phút, giây và tạo chuỗi thời gian có dạng “hh:mm:ss” bằng toán tử định dạng chuỗi.⁹ Ta cũng đã cho thời gian “ngủ” ít hơn để “làm tươi” (refresh) đồng hồ nhanh hơn.

Quan trọng hơn, ta không đếm tới 99 (rồi xong) mà đếm tới ... vô cùng. Như ta thấy, điều kiện lặp là hằng đúng `True` và thân lặp không có các cấu trúc điều khiển thoát khỏi vòng lặp (như lệnh `break` và một số lệnh khác ta sẽ học) nên vòng lặp sẽ được thực thi mãi mãi. Cấu trúc lặp này, do đó, được gọi là **lặp vô hạn** (infinite loop, endless loop). Nhưng ... đó là lý thuyết. Rõ ràng, nếu bạn đóng cửa sổ đồng hồ (cửa sổ `turtle`) thì vòng lặp trên sẽ kết thúc.¹⁰ Sâu xa hơn, nếu thời gian “dừng lại” thì lời gọi hàm `localtime` sẽ có lỗi thực thi vì “không còn” thời gian để mà lấy, khi đó, Python sẽ dừng thực thi chương trình (và hiển nhiên kết thúc vòng lặp). Câu hỏi thời gian có “dừng lại” hay không xin nhường cho các nhà Vật lý trả lời nhưng bạn cũng nên biết rằng: *không có vô hạn trong thực tế, vô hạn chỉ có trong tưởng tượng mà thôi!*

⁹Tương tự như cách ta lấy năm hiện tại trong Bài tập 4.6. Ta sẽ tìm hiểu về “thuộc tính” và bộ 3 kí hơn ở Phần ?? và ??.

¹⁰Khi bạn đóng cửa sổ `turtle` thì lệnh `update()` ở Dòng 11 sẽ có lỗi thực thi (do không còn cửa sổ để mà `update`). Lỗi này được gọi là `turtle.Terminator` như thông báo trong IDLE và làm cho Python kết thúc thực thi (và do đó kết thúc vòng lặp) như mọi lỗi thực thi khác.

Thường thì cấu trúc lặp vô hạn (“while True”) được dùng có chủ đích khi việc đưa ra điều kiện lặp khó khăn. Thay vì cố gắng xác định điều kiện từ đầu, ta dùng hằng đúng True và trong thân lặp ta dùng lệnh break để thoát khỏi vòng lặp khi cần. Cách xác định điều kiện như vậy (không xác định trước từ đầu mà xác định khi cần, khi phù hợp) là một trường hợp của chiến lược **đánh giá/lượng giá muộn** (lazy/deferred evaluation). Bạn không nên lạm dụng nhưng cũng không nên ghét nó. Thường thì việc xác định trước sẽ rõ ràng hơn nhưng đánh giá muộn cũng phù hợp trong nhiều tình huống.

Kĩ thuật đánh giá muộn cũng giúp ta hiện thực khuôn mẫu lặp DO-WHILE một cách tự nhiên như sau:

```
while True:
    <S>
    if not(<C>):
        break
```

Hay khuôn mẫu lặp REPEAT-UNTIL:

```
while True:
    <S>
    if <C>:
        break
```

Trong những trường hợp như thế này, ta gọi vòng lặp là “**giả vô hạn**” (pseudo-infinite loop).

Nếu câu hỏi về lặp vô hạn ở trên khá Triết và “tào lao” thì câu hỏi lặp vô hạn sau đây lại rất Toán và thực tế. Xét quá trình lặp đơn giản sau:

Cho n là một số nguyên dương, lặp lại quá trình sau đây cho đến khi n là 1 thì dừng:

- Nếu n chẵn: $n = n/2$,
- Nếu n lẻ: $n = 3n + 1$.

Người ta đã thử với rất nhiều giá trị n (nguyên dương) khác nhau và đều thấy rằng quá trình lặp trên là dừng, chẳng hạn, với $n = 12$ thì quá trình lặp sẽ cho n lần lượt là 12, 6, 3, 10, 5, 16, 8, 4, 2, 1 và dừng. Tuy nhiên, không ai chứng minh được là quá trình trên luôn dừng với mọi trường hợp của n (nguyên dương). Câu hỏi “có trường hợp nào của n để quá trình lặp trên là vô hạn hay không?” được gọi là **nghi vấn Collatz** (Collatz conjecture) hay **bài toán $3n + 1$** ($3n + 1$ problem) mà chưa có ai trả lời được.¹¹

Chương trình sau đây thực hiện quá trình $3n + 1$ với giá trị n do người dùng nhập. Bạn hãy chạy thử, biết đâu giải được nghi vấn này (nghĩa là tìm được giá trị n làm cho chương trình lặp vô hạn:;) Bạn cũng nên nghiên cứu để thành thạo cách viết các lệnh lặp trong Python.

¹¹Bạn có thể đọc thêm về bài toán này ở https://en.wikipedia.org/wiki/Collatz_conjecture.

https://github.com/vqhBook/python/tree/master/lesson07/Collatz_conj.py

```

1 while text := input("Nhập số nguyên dương n: "):
2     n = int(text)
3     num = 0
4     print(n, end=" ")
5     while n != 1:
6         if n % 2 == 0: # n chẵn
7             n = n // 2
8         else: # n lẻ
9             n = 3*n + 1
10        num += 1
11        print(n, end=" ")
12
13    print("\nSố lần lặp là", num)

```

Chương trình trên thực ra cho bạn thử nghiệm với nhiều số n khác nhau. Hơn nữa, chương trình cũng đếm và xuất số lần lặp cho mỗi thử nghiệm (biến `num`). Chương trình sẽ dừng khi bạn nhập chuỗi rỗng (chỉ ấn phím Enter). Lưu ý là chuỗi rỗng được Python xem là `False`. Đây cũng là ví dụ cho thấy cách dùng toán tử `:=` hay mà nếu không dùng nó thì ta có thể phải viết dài hơn như sau:

```

1 while True:
2     text = input("Nhập số nguyên dương n: ")
3     if not text:
4         break
5     #...

```

Vì lặp vô hạn là thủ phạm chính làm cho chương trình bị “treo” nên ta luôn muốn kiểm tra để đảm bảo các vòng lặp không bị lặp vô hạn.¹² Trong một số trường hợp, việc kiểm tra lặp vô hạn này là khá dễ nhưng trong một số trường hợp khác thì rất khó, thậm chí là bất khả thi. Theo bạn, vòng lặp `while` ngoài cùng (Dòng 1) có lặp vô hạn không?

7.7 Duyệt chuỗi

Ta đã thấy con rùa tự bò để bỏ trốn trong Phần 6.5. Bây giờ ta sẽ hướng dẫn con rùa bò để “vẽ”. Chương trình sau đây cũng minh họa nhiều kỹ thuật lặp.

https://github.com/vqhBook/python/tree/master/lesson07/turtle_draw.py

```

1 import turtle as t
2
3 t.shape("turtle")

```

¹²Thật ra người dùng luôn rất thiếu kiên nhẫn, thậm chí, chỉ cần chương trình hơi chậm phản hồi (nghĩa là hơi “đơ đơ”) là nó có nguy cơ bị tắt rồi.

```

4 t.speed("slowest")
5 d = 20
6 while ins := input("Nhập chuỗi lệnh (L, R, U, D): "):
7     for i in range(len(ins)):
8         if ins[i] == "L":
9             t.setheading(180)
10        elif ins[i] == "R":
11            t.setheading(0)
12        elif ins[i] == "U":
13            t.setheading(90)
14        elif ins[i] == "D":
15            t.setheading(270)
16        else:
17            continue
18        t.forward(d)
19 print("Done!")

```

Khi chạy chương trình, bạn nhớ đóng hàng cửa sổ IDLE với cửa sổ con rùa để quan sát tốt hơn vì bạn ra lệnh cho con rùa bằng cách nhập chuỗi lệnh trong IDLE còn con rùa thực thi (bò) trong cửa sổ của nó. Chuỗi lệnh là dãy các kí tự với ý nghĩa L là bò qua trái (Left), R là bò qua phải (Right), U là bò lên trên (Up), D là bò xuống dưới (Down), các kí tự khác sẽ được bỏ qua bằng lệnh `continue` ở Dòng 17 (nghĩa là con rùa vẫn đứng yên, lệnh `forward` ở dòng 18 không được thực thi). Khi nhập chuỗi lệnh rỗng "" (nghĩa là nhấn Enter mà không nhập gì), chương trình sẽ thoát khỏi vòng lặp `while`; xuất thông báo ở Dòng 19 và kết thúc.

Quan trọng, vòng lặp `for` (Dòng 7-18) minh họa kĩ thuật lặp qua các kí tự của một chuỗi. Ta đã biết rằng chuỗi gồm các kí tự (theo thứ tự từ kí tự đầu đến kí tự cuối cùng). Nay ta biết thêm rằng, ta có thể “truy cập” (tức là đọc hay lấy về) các kí tự này bằng **chỉ số** (index) (tức là vị trí) của kí tự theo cách viết:

`<String>[<Index>]`

Vị trí này tính từ 0 nên kí tự đầu tiên có chỉ số 0, kí tự thứ hai có chỉ số 1, ..., kí tự cuối cùng có chỉ số là $l - 1$ với l là **chiều dài** (length) của chuỗi (tức là số lượng kí tự trong chuỗi). Hàm dựng sẵn `len` giúp ta lấy về chiều dài của chuỗi. Bằng cách dùng biến lặp làm chỉ số và cho nó nhận giá trị từ 0 đến trước chiều dài chuỗi, ta có thể “duyet” qua các kí tự của chuỗi (nghĩa là “xử lý” lần lượt từng kí tự) như lệnh `for` trên cho thấy.¹³ Ta cũng có cách khác, tinh tế hơn, để duyệt qua các kí tự của chuỗi mà ta sẽ biết trong bài chuyên sâu về chuỗi.

¹³Đây cũng là lí do mà biến lặp hay được đặt tên là `i` (chữ cái đầu của index); lý do tổng quát hơn là vì nó thường là số nguyên (chữ cái đầu của integer). Dĩ nhiên, bạn có thể dùng tên khác nếu muốn.

Tóm tắt

- Ta có thể lặp lại một số lần xác định các công việc nào đó với lệnh `for`. Số lần lặp có thể lớn tùy ý và được xác định lúc thực thi. Hàm `range` giúp điều khiển chi tiết biến lặp.
- Các công việc cùng khuôn dạng nhưng khác nhau chi tiết có thể được tham số hóa thành lớp các công việc mà giá trị của tham số sẽ xác định việc cụ thể. Bằng cách đặt công việc được tham số hóa trong thân lặp, xác định tham số theo biến lặp, và điều khiển biến lặp, ta có thể lặp qua các công việc này.
- Module `time` cung cấp các dịch vụ liên quan đến thời gian như: tạm dừng chương trình trong một khoảng thời gian với hàm `sleep`, lấy thời gian trên máy với hàm `localtime`.
- Lệnh `while` rất linh hoạt, nó có thể được dùng để hiện thực nhiều cấu trúc lặp khác nhau.
- Vòng lặp lồng là kĩ thuật “lặp trong lặp” mà ta dùng nó khi công việc cần lặp lại cũng có dạng lặp (nghĩa là cần lặp lại các việc nhỏ hơn). Việc khử lồng thường có thể được thực hiện để đưa vòng lặp lồng về vòng lặp đơn nhưng cũng không nên đặt nặng quá việc này.
- Lệnh `break` giúp thoát ra khỏi vòng lặp. Lệnh `continue` giúp chuyển qua lần lặp mới mà không thực thi phần sau đó trong thân lệnh. Phần `else` (nếu có) của lệnh lặp sẽ được thực thi sau khi lệnh lặp kết thúc tự nhiên.
- Không có lặp vô hạn trong thực tế. Lặp vô hạn chỉ có trong tưởng tượng, khi đó, có nhiều trường hợp rất khó xác định một vòng lặp có lặp vô hạn hay không.
- Cấu trúc lặp giả vô hạn, nhất là cấu trúc “`while True` với `break`”, hay được dùng có chủ đích trong nhiều trường hợp. Toán tử `:=` cũng hay được dùng trong điều kiện của vòng lặp giúp viết vòng lặp gọn hơn.
- Ta có thể lặp qua các kí tự của một chuỗi bằng cách dùng lệnh `for` với biến lặp là chỉ số. Hàm `len` giúp lấy về chiều dài chuỗi.

Bài tập

7.1 Tính lặp. Không gì phù hợp hơn việc cài đặt các phương pháp “tính lặp” bằng cách dùng các cấu trúc lặp. Chẳng hạn, việc tính giai thừa:

$$S(n) = n! = 1 \times 2 \times 3 \times \dots \times (n-1) \times n$$

có thể được cài đặt bằng lệnh lặp `for` như sau:

```

1 https://github.com/vqhBook/python/tree/master/lesson07/factorial.py
2 n = int(input("Nhập số nguyên dương: "))
  f = 1

```

```

3 for i in range(1, n + 1):
4     f *= i
5 print(f"{n}! = {f}")

```

hoặc có thể viết bằng lệnh lặp while như sau:¹⁴

```

https://github.com/vqhBook/python/tree/master/lesson07/factorial2.py
1 n = int(input("Nhập số nguyên dương: "))
2 f, i = 1, 1
3 while i <= n:
4     f *= i
5     i += 1
6 print(f"{n}! = {f}")

```

Cài đặt chương trình thực hiện các tính toán sau bằng phương pháp tính lặp với các lệnh lặp:¹⁵

(a) $2^{32}, 2^{60}, 2^{1000}$.

(b) Tính căn bậc 2 của một số thực dương (được nhập) bằng phương pháp Newton trong Bài tập 2.7.

(c) $S(n) = 1 + 2 + \dots + (n-1) + n$.¹⁶

(d) $S(n) = 1^2 + 2^2 + \dots + (n-1)^2 + n^2$.¹⁷

(e) $S(x, n) = 1 + x + x^2 + \dots + x^{n-1} + x^n$ (với x là số thực, n là số nguyên không âm).¹⁸

(f) $S(x, n) = 1 + x + \frac{x^2}{2!} + \dots + \frac{x^{n-1}}{(n-1)!} + \frac{x^n}{n!}$.¹⁹

(g) $\frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{\dots}}}}$.²⁰

(h) $\frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{\dots}}}}$.²¹

Gợi ý: Câu (e), (f) có thể tính dễ dàng bằng vòng lặp lồng nhưng nếu tính được bằng vòng lặp đơn (khử lồng) thì sẽ hiệu quả hơn. Dấu 3 chấm (...) trong Câu

¹⁴Dĩ nhiên, bạn không cần “vắt vớ” như vậy để tính giai thừa. Bạn chỉ cần nạp module math và gọi `math.factorial(n)` hoặc `math.prod(range(1, n+1))` để tính $n!$.

¹⁵Nhớ là bạn không được dùng các hàm (dựng sẵn hay trong các module) để tính. Bạn tự “tính tay” bằng cách lặp. Bạn cũng nên kiểm tra lại kết quả với các hỗ trợ có sẵn, chẳng hạn, bạn nên dùng hàm `math.factorial` để kiểm tra kết quả tính giai thừa ở trên có đúng không.

¹⁶Bạn có thể đối chiếu kết quả với $\frac{n(n+1)}{2}$.

¹⁷Bạn có thể đối chiếu kết quả với $\frac{n(n+1)(2n+1)}{6}$.

¹⁸Kí hiệu $S(x, n)$ để chỉ kết quả tính phụ thuộc vào x và n , tức là việc tính giá trị trên là công việc được tham số hóa bởi 2 tham số x, n . Bạn có thể đối chiếu kết quả với $\frac{1-x^{n+1}}{1-x}$.

¹⁹Khi x nhỏ và n lớn thì giá trị này xấp xỉ e^x , mà trong Python ta có thể tính bằng cách gọi `math.exp(x)`.

²⁰Khi số lần lặp đủ lớn thì giá trị này xấp xỉ tỉ số vàng $\phi = \frac{1+\sqrt{5}}{2}$.

²¹Khi số lần lặp đủ lớn thì giá trị này xấp xỉ \sqrt{x} . Như vậy, đây là một phương pháp khai phương khác.

(g), (h) có nghĩa là lặp vô hạn. Ta cũng có thể viết vòng lặp vô hạn để tính, nhưng rõ ràng, ta không thể đợi nó chạy! Như vậy, ta chỉ lặp với số lần đủ lớn (và có được kết quả xấp xỉ). Ta có thể dùng giá trị “khởi động” (tức là giá trị cho dấu 3 chấm khi đã “khai triển” nó đủ nhiều) là 1.

7.2 Thiết kế thuật toán và cài đặt chương trình cho người dùng nhập các số và thực hiện các yêu cầu sau trên các số người dùng đã nhập:

- (a) Tìm số nhỏ nhất (min)
- (b) Tìm số lớn nhất (max)
- (c) Tìm đồng thời số lớn nhất và nhỏ nhất
- (d) Tìm số lớn thứ 2
- (e) Tìm số chẵn lớn nhất
- (f) Tính tổng (sum)
- (g) Tính trung bình cộng (average)
- (h) Đếm số lượng số dương
- (i) Đếm số lượng số chính phương
- (j) Tính tỉ lệ số dương

7.3 Trong Bài tập 4.7, ta đã viết chương trình tính tổng các chữ số của một số nguyên dương có không quá 3 chữ số. Mở rộng chương trình để tính tổng các chữ số của một số nguyên dương:

- (a) Có không quá 5 chữ số
- (b) Có không quá 10 chữ số
- (c) Có số chữ số bất kì

7.4 Tính mũ lũy thừa của 2 (x^{2^n}). Từ nhận xét:

$$(x^{2^k})^2 = x^{2 \times 2^k} = x^{2^{k+1}}$$

Với x là số thực bất kì, ta dễ dàng tính $x, x^2, x^4, x^8, \dots, x^{2^{n-1}}, x^{2^n}$ bằng cách bình phương số đang trước để có số đằng sau. Viết chương trình cho nhập số thực x , số nguyên không âm n , tính x^{2^n} .

7.5 Trong mã nguồn `star.py` ở Phần 5.2, ta đã vẽ hình “ngôi sao 5 cánh” bằng con rùa. Hãy thiết kế thuật toán (tức là cánh vẽ hay chiến lược vẽ) và cài đặt chương trình vẽ hình ngôi sao:

- (a) 6 cánh
- (b) 10 cánh
- (c) Có số lượng cánh do người dùng nhập (dĩ nhiên là từ 3 cánh trở lên)

7.6 “Vẽ” bằng kí tự. Viết chương trình “vẽ” bằng kí tự các hình sau:²²

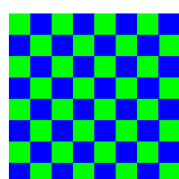
²²Chương trình cho người dùng nhập các tham số hợp lý của hình rồi vẽ theo đó (tương tự mã nguồn `triangle.py` ở Phần 7.4).

```

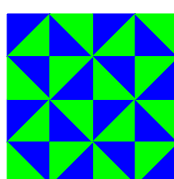
      *          *****          *          *****
     **         *****         ***         *****
    ***        *****        *****        *****
   ****       *****       *****       *****
  *****     *****     *****     *****
 *****     *****     *****     *****
*****      *****      *****      *****
*****      *****      *****      *****
*****      *****      *****      *****
*****      *****      *****      *****

```

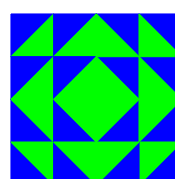
7.7 Viết chương trình dùng con rùa tô màu các hình sau:



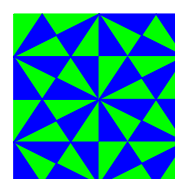
(a)



(b)



(c)



(d)

Gợi ý: Suy nghĩ chiến lược (thuật toán) trước rồi viết chương trình sau. Hình nào khó quá thì ... bỏ qua:)

7.8 Trong Bài tập 6.6, ta đã viết chương trình xuất ra theo thứ tự tăng dần của 3 số được nhập. Hãy thiết kế thuật toán và cài đặt chương trình mở rộng cho số lượng số bất kì.

Gợi ý: hơi khó à:)

7.9 **Tam giác Pascal.** Tam giác Pascal (Pascal's triangle)²³ là “tam giác các số” với dòng 1 gồm 1 số, dòng 2 gồm 2 số, ... Ví dụ sau là tam giác Pascal gồm 7 dòng:

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1

```

Như ví dụ trên cho thấy, các số của tam giác Pascal được xây dựng theo quy tắc:

- Các số ở cột đầu tiên và “đường chéo” đều là 1,
- Các số “ở giữa” là tổng của số ở “ngay trên” và “trên trước”.

²³Đặt theo tên nhà Toán học Pháp Blaise Pascal.

Cụ thể, ta nói P_n là tam giác Pascal n dòng nếu P_n gồm các số $P_n(i, j)$, số ở dòng i cột j , theo qui tắc:

- $P_n(i, j) = 1$ nếu $j = 1$ hoặc $i = j$,
- $P_n(i, j) = P_n(i - 1, j - 1) + P_n(i - 1, j)$ nếu $1 < i \leq n$ và $1 < j < i$.

Viết chương trình cho nhập số n (nguyên dương) và xuất tam giác Pascal n dòng.

Gợi ý: hơi khó à:)

7.10 Viết các chương trình sau:

“Tra” mã Unicode của kí tự được nhập như minh họa:

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Nhập kí tự muốn tra mã: ò
Kí tự ò có mã Unicode là 0xf2
Nhập kí tự muốn tra mã: ≤
Kí tự ≤ có mã Unicode là 0x2264
Nhập kí tự muốn tra mã: 😊
Kí tự 😊 có mã Unicode là 0x1f600
Nhập kí tự muốn tra mã:
Done!
Ln: 13 Col: 4
```

Hiển thị kí tự có mã Unicode được nhập như minh họa:

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Nhập mã kí tự muốn tra: 0xf2
Mã Unicode 0xf2 là của kí tự ò
Nhập mã kí tự muốn tra: 0x2264
Mã Unicode 0x2264 là của kí tự ≤
Nhập mã kí tự muốn tra: 0x1f600
Mã Unicode 0x1f600 là của kí tự 😊
Nhập mã kí tự muốn tra:
Done!
Ln: 14 Col: 4
```

Gợi ý: Xem lại Unicode trong Phần 4.6. Dùng hàm `ord` để lấy về mã Unicode của kí tự, hàm `chr` để lấy về kí tự theo mã Unicode, hàm `hex` hoặc định dạng với chuỗi đặc tả định dạng "x" để được chuỗi số viết theo cơ số 16, và hàm `int` với giá trị cho đối có tên `base` là 16 để nhập số nguyên viết theo cơ số 16. Bạn tra cứu thêm để rõ hơn.

7.11 Dùng Python để tìm 2 số tự nhiên, biết rằng: tổng của chúng bằng 1006 và nếu lấy số lớn chia cho số nhỏ thì được thương là 2 và số dư là 124.²⁴

²⁴SGK Toán 9 Tập 2.

7.12 Dùng Python giải bài toán cổ sau:²⁵

“Quýt, cam mười bảy quả tươi
 Đem chia cho một trăm người cùng vui.
 Chia ba mỗi quả quýt rồi
 Còn cam mỗi quả chia mười vừa xinh.
 Trăm người, trăm miếng ngọt lành.
 Quýt, cam mỗi loại tính rành là bao?”

7.13 Dùng Python giải bài toán cổ sau:

“Trăm trâu, trăm cỏ
 Trâu đứng ăn năm
 Trâu nằm ăn ba
 Ba trâu già ăn một
 Hỏi mỗi loại bao nhiêu con?”

Gợi ý: để mò lời giải của bài toán 2 ẩn (nguyên) ta đã dùng vòng lặp lồng “2 cấp” (tức là “lặp trong lặp”). Tương tự, để mò lời giải của bài toán 3 ẩn (nguyên) ta có thể dùng vòng lặp lồng “3 cấp” (tức là “lặp trong lặp trong lặp”).

7.14 Python trực tuyến. Như đã biết từ Phần 1.1 và Phần 4.1, ta có thể dùng **Python trực tuyến** (online Python) để lập trình Python. Ngoài việc không cần cài đặt Python trên máy, lợi ích khác của Python trực tuyến là ta có thể dùng nó trên mọi thiết bị có kết nối Internet (như iPad, iPhone, smartphone Android, ...). Ta cũng có thể lưu trữ các file mã nguồn trên các ứng dụng Web này (thường yêu cầu tạo tài khoản).

Thử các trang Python trực tuyến sau, tạo tài khoản (miễn phí) để có thể lưu trữ file mã nguồn:²⁶

- Repl.it (<https://repl.it/languages/python3>)
- PythonAnywhere (<https://www.pythonanywhere.com/>)
- Paiza.IO (<https://paiza.io/en/languages/python3>)
- JDoodle (<https://www.jdoodle.com/python3-programming-online/>)
- OnlineGDB (https://www.onlinegdb.com/online_python_compiler)

Chọn ra trang ưng ý nhất rồi tạo file, viết mã cho các chương trình trong Bài này. Thử dùng trang này trên các thiết bị khác như iPad, iPhone, ... để lập trình Python.

²⁵SGK Toán 9 Tập 2.

²⁶Search Google với từ khóa “online Python”.

Bài 8

Tự viết lấy hàm

Ta đã dùng nhiều hàm (hàm dựng sẵn và hàm của module) trong các bài trước. Trong bài này, ta tự mình viết lấy hàm, nghĩa là, ta đóng vai trò của người cung cấp dịch vụ thay vì người dùng các dịch vụ có sẵn. Ta cũng thấy rằng, trong sự thống nhất của Python, hàm cũng là đối tượng có thể được truyền nhận và thao tác như dữ liệu. Ta cũng học cách tự viết lấy module và biết sơ lược về lập trình hướng sự kiện, một kĩ thuật được dùng trong nhiều công nghệ lập trình.

8.1 Định nghĩa hàm/gọi hàm và tham số/đối số

Tôi đã xuất ra lời chào trang trọng đến mình trong một cái khung ở Phần 1.2. Thế Guido van Rossum hay ai đó cũng muốn làm vậy thì sao? Chẳng lẽ chép lại đoạn mã rồi chỉnh sửa (cũng vất vả á, sửa tên rồi phải điều chỉnh cái khung cho vừa). Rồi 10 người, 100 người muốn vậy thì sao? Rõ ràng “chép-sửa” là không ổn (tương tự như “chép-dán” là không ổn trong minh họa 1 triệu đóa cúc ở Phần 7.1). Python cho phép ta giải quyết rất đẹp vấn đề này bằng *phương tiện quan trọng nhất của lập trình là hàm* (function). Thử chương trình minh họa sau:

<https://github.com/vqhBook/python/tree/master/lesson08/hello.py>

```
1 def sayhello(name):
2     hello_string = " Chào " + name + " "
3     print("=" * (len(hello_string) + 2))
4     print("|" + hello_string + "|")
5     print("=" * (len(hello_string) + 2))
6
7 sayhello("Python")
8 sayhello("Vũ Quốc Hoàng")
9 sayhello("Guido van Rossum")
10 a_name = input("Tên của bạn là gì? ")
11 sayhello(a_name)
```

Rõ ràng, cái khung tên cho tôi, hay cho Guido van Rossum, hay cho mọi người

đều giống nhau ở cái khung còn khác nhau ở cái tên cụ thể. Như trong Phần 7.2 ta đã biết, việc xuất ra khung tên này có thể được tham số hóa thành $S(i)$ với ý nghĩa “xuất ra tên i trang trọng trong một cái khung”. Đây chính là việc ta đã làm ở các Dòng 1-5, **định nghĩa** (define) một hàm có tên là `sayhello` (tên gọi nhớ hơn S) với **tham số** (parameter) `name` (tên gọi nhớ hơn i) và **thân hàm** (function body) là khối lệnh xuất ra cái khung tên đó.

Cú pháp định nghĩa một hàm nói chung là:

```
def <Name>(<Paras>) :
    <Suite>
```

Trong đó, `def` là từ khóa, các dấu đóng mở ngoặc tròn bọc `<Paras>` và dấu hai chấm (`:`) là bắt buộc. `<Paras>` là danh sách các tham số, đó là các tên cách nhau bằng dấu phẩy (`,`) nếu có nhiều tham số hoặc để trống nếu không có. `<Name>` là tên hàm và `<Suite>` là khối lệnh mô tả công việc của thân hàm.

Tôi sẽ không bàn chi tiết các lệnh trong thân hàm `sayhello` trên. Đến giờ, công lực của bạn cũng đã khá và tôi chỉ bàn những gì mới hay đáng bàn. Dĩ nhiên, dòng trống sau thân hàm (Dòng 6) là không bắt buộc nhưng nó thường hay được dùng để phân cách **định nghĩa hàm** (function definition) với phần chương trình bên dưới.¹

Sau khi đã kí hiệu $S(i)$ là việc “xuất ra tên i trang trọng trong một cái khung” thì kí hiệu S (Vũ Quốc Hoàng), S (Guido van Rossum), ... ám chỉ các công việc cụ thể khi thay Vũ Quốc Hoàng vào i là “xuất ra tên Vũ Quốc Hoàng trang trọng trong một cái khung”, thay Guido van Rossum vào i là “xuất ra tên Guido van Rossum trang trọng trong một cái khung”, ... Tương tự như vậy, sau khi định nghĩa hàm `sayhello(name)` thì việc **gọi hàm** (calling function) `sayhello("Vũ Quốc Hoàng")`, `sayhello("Guido van Rossum")`, ... sẽ thực thi công việc cụ thể tương ứng, tức là thực thi thân hàm với **đối số** (argument) "Vũ Quốc Hoàng", "Guido van Rossum", ... thay vào (tức là gán vào) tham số `name`.

Như vậy, tương tự việc đặt tên cho một giá trị để có thể dùng lại nó sau đó, ta cũng có thể đặt tên cho một khối lệnh để có thể dùng lại nó. Trường hợp đầu ta có tên biến, trường hợp sau ta có tên hàm. Việc dùng lại biến (trong biểu thức) sẽ được giá trị tương ứng, việc dùng lại hàm (gọi hàm) sẽ chạy lại khối lệnh tương ứng. Hơn nữa, việc thực thi khối lệnh có thể phụ thuộc vào một số tham số mà ta có thể cung cấp giá trị cụ thể khác nhau qua các đối số cho mỗi lần chạy. Đây chính là các biến thể khác nhau của công việc được tham số hóa mà ta đã biết.

Cũng lưu ý, *các hàm phải được định nghĩa trước khi dùng*. Chẳng hạn, nếu bạn dời định nghĩa hàm `sayhello` ở trên (Dòng 1-5) ra sau lời gọi hàm `sayhello` (Dòng 7 chẳng hạn) thì Python báo lỗi thực thi `NameError (name 'sayhello' is not defined)`. Lỗi này tương tự lỗi dùng một (tên) biến mà chưa được định nghĩa (gán).

Trước khi qua phần tiếp theo, bạn cũng nên nghiền ngẫm lại để *phân biệt rõ ràng định nghĩa hàm với gọi hàm, tham số với đối số*. Nhiều người nhầm lẫn những

¹Đây cũng là khuyến cáo của PEP 8.

cặp khái niệm này, nhất là tham số với đối số. Để rõ ràng, tham số được gọi là **tham số/đối số hình thức** (formal parameter/argument) vì nó được đặt trong định nghĩa hàm (cũng còn gọi là khai báo hàm) còn đối số được gọi là **tham số/đối số thực tế** (actual parameter/argument) vì nó là giá trị cung cấp khi gọi thực thi hàm.

8.2 Hàm tính toán, thủ tục và lệnh return

Hàm sayhello ở trên thường được gọi là **thủ tục** (procedure) vì nó không trả về giá trị nào khi thực thi. Ngược lại, trường hợp điển hình hơn, ta mong đợi kết quả trả về từ hàm. Ta thường gọi các hàm này là **“hàm tính toán”** với thuật ngữ tính toán được dùng theo nghĩa rộng. Để khỏi nhầm (thuật ngữ tính toán thường được hiểu theo nghĩa hẹp là tính toán số học) ta có thể gọi chúng là **“hàm có trả về giá trị”**. Thuật ngữ hàm của Python ám chỉ cả 2 loại.

Ta đã biết rằng, Python cung cấp hàm sqrt trong module math để thực hiện việc khai căn (ta cũng có thể dùng hàm pow hay toán tử mũ). Trong minh họa sau đây, ta tự mình viết lấy hàm tính căn này theo phương pháp Newton (xem lại Bài tập 2.7 và Bài tập 4.2):

```

1  import math
2
3  def Newton_sqrt(x):
4      y = x
5      for i in range(100):
6          y = y/2 + x/(2*y)
7      return y
8
9  print(f"Căn của 2 theo Newton: {Newton_sqrt(2):.9f}")
10 print(f"Căn của 2 theo math's sqrt: {math.sqrt(2):.9f}")
11
12 x = float(input("Nhập số cần tính căn: "))
13 print(f"Căn của {x} theo Newton: {Newton_sqrt(x):.9f}")
14 print(f"Căn của {x} theo math's sqrt: {math.sqrt(x):.9f}")

```

Trong hàm, **lệnh return** (return statement) với cú pháp:

return <Expr>

giúp trả về giá trị cho hàm. Cụ thể, khi gặp lệnh này, Python lượng giá biểu thức <Expr> để được giá trị Value, kết thúc thực thi hàm và trả Value về làm **giá trị trả về** (return value) của hàm. Lệnh return cũng có thể được dùng mà không có biểu thức trả về (gọi là “return không đối số”), khi đó, giá trị trả về là None, giá trị đặc biệt mô tả “không có giá trị” như ta đã biết ở Phần 4.4.

Vì yêu cầu kết thúc thực thi hàm (dù đang ở bất kì đâu trong hàm) nên lệnh đơn return là lệnh điều khiển luồng thực thi. Hơn nữa, nếu một hàm không dùng tường

minh lệnh `return` (nghĩa là quá trình thực thi hàm không dùng lệnh `return`) thì khi thực thi xong thân hàm, Python sẽ tự động thực thi lệnh `return` không đối số (và do đó hàm có giá trị trả về là `None`) như hàm `sayhello` trên. Như vậy, *hàm nào trong Python cũng trả về giá trị*. Đặc biệt, nếu giá trị trả về là `None` thì được hiểu là “không trả về giá trị” mà thường ứng với tình huống “việc trả về giá trị là không có ý nghĩa”.

Thật ra, ranh giới giữa hàm tính toán và thủ tục là không rõ ràng. Chẳng hạn, vì số âm không có căn nên hàm `Newton_sqrt` cần được viết kĩ hơn như sau:

```

1 def Newton_sqrt(x):
2     if x < 0:
3         return
4     if x == 0:
5         return 0
6
7     y = x
8     for i in range(100):
9         y = y/2 + x/(2*y)
10    return y

```

Lưu ý, ta không cần dùng lệnh `if` đủ ở Dòng 2 vì khi gặp lệnh `return`, hàm sẽ kết thúc, do đó không thực thi các phần còn lại theo luồng thực thi thông thường. Nói cách khác, phần còn lại (Dòng 4-10) là phần `else` của lệnh `if` ở Dòng 2.

Lệnh `return` ở Dòng 3 nếu thực thi (khi tham số `x` nhận giá trị nhỏ hơn 0) sẽ trả về giá trị `None` (để báo số âm không có căn). Hơn nữa, để tránh lỗi chia cho 0 trong cách khởi động thuật toán Newton ($y = x$ ở Dòng 7 và sau đó chia cho `y` ở Dòng 9), ta tách riêng trường hợp tính căn của 0 (Dòng 4-5). Với cách cài đặt này thì hàm `mysqrt` có lúc trả về `None` có lúc lại trả về giá trị thông thường nên không biết xếp nó vào loại gì.

Việc trả về `None`, nghĩa là không có giá trị, trong trường hợp tính căn số âm có vẻ hợp lý và đúng bản chất nhưng không phải là lựa chọn “chuẩn mực” hay dùng. Chẳng hạn, sau khi định nghĩa hàm trên (nghĩa là chạy chương trình có chứa định nghĩa hàm đó hoặc gõ hàm trực tiếp trong chế độ tương tác), tương tác với Python như sau:

```

1 >>> Newton_sqrt(2)
  1.414213562373095
2 >>> Newton_sqrt(-2)
3 >>> print(Newton_sqrt(-2))
  None
4 >>> math.sqrt(-2)
  ...
  ValueError: math domain error

```

Cách xử lý chuẩn mực của Python trong trường hợp trên (tính căn số âm) là

phát sinh lỗi thực thi (ValueError) như kết quả của lời gọi hàm `math.sqrt(-2)` cho thấy. Ta sẽ biết cách xử lý tương tự như vậy sau, tạm thời, lựa chọn dùng None để báo “không giá trị” cho các trường hợp đặc biệt như thế này là hợp lý.

8.3 Cách truyền đối số và đối số mặc định

Thật ra, hàm `sayhello` ở trên có thể được gọi bằng cách chỉ rõ tên tham số lúc truyền đối số như `sayhello(name="Vũ Quốc Hoàng")`. Rõ ràng, cách gọi này rườm rà hơn `sayhello("Vũ Quốc Hoàng")`; tuy nhiên, đó là vì hàm `sayhello` chỉ có 1 tham số, trường hợp hàm có nhiều tham số thì khác.

Bạn xem lại Bài tập 2.2, trong đó, ta được yêu cầu tính giá trị của biểu thức:

$$N = 8x^3 - 12x^2y + 6xy^2 - y^3$$

tại $x = 6$ và $y = -8$. Cũng vậy, nếu người dùng muốn tính N tại nhiều trường hợp x, y khác nhau thì ta nên viết hàm tính giá trị N theo công thức trên (giả sử công thức này rất phức tạp để nhu cầu viết hàm là chính đáng). Rõ ràng và tự nhiên, hàm này có 2 tham số với tên x, y vì giá trị của biểu thức phụ thuộc vào giá trị cụ thể của x, y . Hơn nữa, ta đặt tên hàm là N luôn cho gọi nhớ:

```

1 https://github.com/vqhBook/python/tree/master/lesson08/expression\_value.py
2 def N(x, y):
3     return 8*(x**3) - 12*(x**2)*y + 6*x*(y**2) - y**3
4
5 print(N(6, -8))
6 print(N(-8, 6))
7 print(N(x=6, y=-8))
8 print(N(y=-8, x=6))
9 print(N(x=-8, y=6))
10 print(N(6, y=-8))
    # print(N(y=-8, 6)) # SyntaxError

```

Ở Dòng 4 ta tính giá trị của biểu thức N tại $x = 6$ và $y = -8$ bằng cách gọi hàm N với 2 đối số 6, -8 thay tương ứng cho 2 tham số x, y . Cách Python thay (chính là gán) giá trị của đối số cho tham số được gọi là cách **truyền đối số** (argument passing) (rõ hơn phải là “truyền đối số cho tham số”). Mà tự nhiên nhất là cách gán tương ứng đối số cho tham số theo thứ tự, được gọi là **truyền đối số theo thứ tự** (passing argument by position). Rõ ràng, với cách này, bạn cần nhớ thứ tự của các tham số (tham số thứ nhất là gì? thứ 2 là gì? ...). Chẳng hạn, nếu bạn quên thứ tự, nhớ nhầm y trước x sau (đúng phải là x trước y sau), thì lời gọi $N(-8, 6)$ dự định tính giá trị tại $y = -8$, $x = 6$ lại tính giá trị tại $x = -8$, $y = 6$.

Một cách truyền đối số khác là **truyền đối số theo tên** (passing argument by name, pass-by-name) của tham số. Theo cách này, ta không cần đặt đối số theo thứ tự của tham số vì ta chỉ rõ đối số được truyền cho tham số nào theo cách viết `<Para>=<Arg>` như trong cách gọi ở Dòng 6-8. Như vậy kết quả của Dòng 6 giống

của Dòng 7 và giống Dòng 4 vì đều tính giá trị của N tại $x = 6, y = -8$. Ngược lại, kết quả của Dòng 8 giống Dòng 5.

Ta cũng có thể dùng kết hợp cả hai cách (vị trí và tên) một cách “thích hợp”, chẳng hạn, lời gọi ở Dòng 9 chính là tính N tại $x = 6, y = -8$ (trong đó ta đã dùng vị trí cho tham số x nhưng tên cho tham số y). Ngược lại, lời gọi ở Dòng 10 (nếu dùng), $N(y = -8, 6)$, sẽ báo lỗi cú pháp do qui tắc khá hợp lý là “*đối số được truyền theo tên phải được viết sau các đối số được truyền theo vị trí*”.

Cũng lưu ý là các tham số thông thường ta dùng tới giờ đều là **tham số có tên** (named parameter) mà ta đã gọi chệch thuật ngữ **đối số từ khóa** (keyword argument) (tức là truyền đối số theo tên tham số) thành **đối số có tên** (named argument). Thuật ngữ ổn nhất, như vậy, rất dài là “truyền đối số cho tham số có tên”. Ta sẽ thấy tham số không có tên sau, mà hiển nhiên khi đó ta không thể truyền đối số cho các tham số đó bằng tên được (mà phải dùng vị trí).

Tham số cũng có thể được khai báo nhận **giá trị mặc định** (default value) với cách viết:

`<Para>=<Expr>`

trong định nghĩa hàm.² Khi đó, nếu lời gọi hàm không cung cấp đối số cho tham số dạng này thì nó sẽ được nhận giá trị mặc định. Thử minh họa sau:

<https://github.com/vqhBook/python/tree/master/lesson08/hello2.py>

```

1 def sayhello(name="World", language="en"):
2     hello_verb = ("Chào" if language == "vi" else "Hello")
3     hello_string = f"{hello_verb} {name} "
4     print("=" * (len(hello_string) + 2))
5     print("|" + hello_string + "|")
6     print("=" * (len(hello_string) + 2))
7
8 sayhello()
9 sayhello("Python")
10 sayhello(language="en")
11 sayhello("Vũ Quốc Hoàng", language="vi")
12 sayhello(language="en", name="Guido van Rossum")
13 a_name = input("Tên của bạn là gì? ")
14 sayhello(a_name, "vi")

```

Ta đã viết lại hàm sayhello với 2 tham số. Tham số thứ nhất, name, xác định tên như trước đây; tham số thứ 2, language, xác định ngôn ngữ cho lời chào. Ta đã khai báo cho cả 2 tham số nhận giá trị mặc định tương ứng là "World" và "en" (nghĩa là English - Tiếng Anh).³ Và do đó, khi sayhello được gọi mà thiếu đối số cho tham số tương ứng thì giá trị mặc định được dùng.

²Ta thường mô tả giá trị mặc định bằng hằng nhưng nó có thể là một biểu thức như ta sẽ bàn kĩ sau.

³Ở đây ta dùng **mã ngôn ngữ** (language code) 2 kí tự như qui định trong chuẩn ISO 639-1. Cũng theo đó, mã của ngôn ngữ Tiếng Việt là vi.

Bạn chạy thử và đối chiếu mã để nắm rõ nhé. Chẳng hạn, lời gọi ở Dòng 8, thiếu cả 2 đối số nên cả 2 tham số được nhận giá trị mặc định, nên lời chào “Hello World” được đưa ra. Các tham số có khai báo nhận giá trị mặc định còn được gọi là **tham số/đối số** tùy chọn (optional parameter/argument) vì ta có thể không cung cấp đối số cho nó khi gọi hàm (nó sẽ nhận giá trị mặc định khi đó).

Bạn đã gặp những vấn đề của tham số/đối số này ở hàm dựng sẵn `print`. Các thứ mà bạn đưa cho `print` xuất ra sẽ được truyền cho tham số không tên. 2 tham số có tên `sep` và `end` lần lượt nhận các giá trị mặc định là " " (kí tự trắng, Space) và "\n" (kí tự xuống dòng, Enter). Ngoài ra, bạn buộc lòng phải dùng cách truyền đối số theo tên cho các tham số `sep` và `end` (mà không dùng vị trí được) vì lí do mà bạn sẽ biết sau.⁴

8.4 Hàm cũng là đối tượng

Tôi đã nói mọi thứ trong Python đều là đối tượng. Hàm cũng vậy. Thử các minh họa sau:

```
1 >>> def f(x): return x**2
2
3 >>> g = f
4 >>> print(f, f(10), g(10))
<function f at 0x03804BF8> 100 100
5 >>> print(type(f), type(print))
<class 'function'> <class 'builtin_function_or_method'>
```

Tương tự như dữ liệu, hàm cũng là đối tượng và nó có thể được tham chiếu đến bởi một hoặc nhiều tên hàm (hoặc thậm chí là không có tên như ta sẽ thấy trong phần sau). Tuy nhiên, *ngữ nghĩa của hàm là “thực thi khối lệnh được tham số hóa” chứ không phải là bản thân nó như dữ liệu*. Kiểu của các hàm ta viết là `function`, còn kiểu của các hàm dựng sẵn là `builtin_function_or_method`. Nhân tiện, trường hợp thân hàm ngắn, ta có thể viết nó chung dòng với **tiêu đề hàm** (function header), là phần từ từ khóa `def` đến dấu `:`, như thân các lệnh ghép đã biết.

Trong Bài tập 3.4, ta đã được yêu cầu khai phương một vài số bằng 4 cách khác nhau. Ta có thể làm điều này khá đẹp như sau:

https://github.com/vqhBook/python/tree/master/lesson08/sqrt_cal.py

```
1 import math
2
3 def builtin_pow_sqrt(x):
4     return pow(x, 0.5)
5
6 def math_pow_sqrt(x):
7     return math.pow(x, 0.5)
```

⁴Thật ra `print` còn 2 tham số có tên nữa với các giá trị mặc định tương ứng. Bạn tra cứu để rõ hơn nhé.

```

8
9 def exp_operator_sqrt(x):
10     return x ** 0.5
11
12 def Newton_sqrt(x):
13     y = x
14     for i in range(100):
15         y = y/2 + x/(2*y)
16     return y
17
18 def cal_sqrt(method, method_name):
19     print(f"Tính căn bằng phương pháp {method_name}:")
20     print(f"a) Căn của 0.0196 là {method(0.0196):.9f}")
21     print(f"b) Căn của 1.21 là {method(1.21):.9f}")
22     print(f"c) Căn của 2 là {method(2):.9f}")
23     print(f"d) Căn của 3 là {method(3):.9f}")
24     print(f"e) Căn của 4 là {method(4):.9f}")
25     print(f"f) Căn của {225/256} là {method(225/256):.9f}")
26
27 cal_sqrt(math.sqrt, "math's sqrt")
28 cal_sqrt(builtin_pow_sqrt, "built-in pow")
29 cal_sqrt(math.pow_sqrt, "math's pow")
30 cal_sqrt(exp_operator_sqrt, "exponentiation operator")
31 cal_sqrt(Newton_sqrt, "Newton's sqrt")

```

Việc tính và xuất ra căn của 6 số trong các câu (a)-(f) được lặp lại cho 4 phương pháp khác nhau. Ta có thể tham số hóa nó với tham số “phương pháp tính căn”. Điều này dẫn đến việc định nghĩa hàm `cal_sqrt` với tham số `method` xác định “phương pháp tính căn” còn thân hàm sẽ thực hiện việc tính và xuất ra căn của 6 số theo phương pháp cụ thể được truyền cho `method`.

Khác với các tham số khác tới giờ, tham số `method` xác định một công việc, “việc tính căn theo cách nào đó”, do đó nó sẽ là hàm tính căn (hàm nhận một số và trả về căn của số đó theo phương pháp nào đó). Tóm lại, `method` là một hàm tính căn (hàm nào thì tùy theo đối số lúc gọi), do đó các lời gọi `method` ở Dòng 20-25 thực thi (và nhận kết quả) tính căn của hàm tính căn được cho trong `method` trên 6 số yêu cầu. Hàm `cal_sqrt`, ngoài ra, còn có thêm tham số `method_name` cho biết tên phương pháp (là một chuỗi).

Các phương pháp tính căn được dùng trong chương trình trên (Dòng 27-31) là:

- Dùng hàm `math.sqrt` (hàm `sqrt` của module `math`). Để tính nó ta chỉ cần truyền đối số là hàm này cho tham số `method` của hàm `cal_sqrt` (Dòng 27).
- Dùng hàm dựng sẵn `pow` với số mũ là 0.5. Lưu ý, bản thân `pow` không tính căn (mà tổng quát hơn là tính mũ, với số mũ bất kì) nên ta cần viết một hàm

đặc biệt cho nó, `builtin_pow_sqrt`, nhận một số và trả về căn của số đó bằng cách dùng `pow` với số mũ cụ thể là 0.5. Hàm `builtin_pow_sqrt`, như vậy, còn được gọi là **hàm bọc** (wrapper function) của hàm `pow`.

- Dùng hàm `pow` của module `math` với hàm bọc `math_pow_sqrt`.
- Dùng toán tử mũ với hàm bọc `exp_operator_sqrt`.
- Dùng hàm `Newton_sqrt` ở trên. Dĩ nhiên, bạn đừng bọc hàm này bằng hàm như:

```
def Newton_sqrt_wrapper(x): return Newton_sqrt(x)
```

Hàm `Newton_sqrt` đã ở dạng sẵn dùng (nhận một số và trả về căn của nó) nên nếu bọc nó lại như trên thì vừa kém hiệu quả (tốn thời gian gọi, truyền) vừa “cướp công của nó”. Tóm lại là vô duyên lắm!

Minh họa trên cho thấy: *hàm cũng là đối tượng, nó cũng có thể được tham chiếu đến, truyền nhận và thao tác như các dữ liệu khác*. Thuật ngữ kỹ thuật gọi các hàm thỏa mô hình này là **first-class function**.⁵

8.5 Biểu thức lambda và hàm vô danh

Như bạn thấy, việc bọc hàm `Newton_sqrt` bằng hàm `Newton_sqrt_wrapper` ở trên là rất vô duyên và nên tránh. Hơn nữa, việc bọc toán tử mũ bằng hàm `exp_operator_sqrt` tưởng minh và các hàm bọc khác cũng có thể tránh bằng cách viết đơn giản hơn là dùng hàm vô danh như sau:

— https://github.com/vqhBook/python/tree/master/lesson08/sqrt_cal2.py —

```
1 import math
2
3 def Newton_sqrt(x):
4     y = x
5     for i in range(100):
6         y = y/2 + x/(2*y)
7     return y
8
9 def cal_sqrt(method, method_name):
10    print(f"Tính căn bằng phương pháp {method_name}:")
11    print(f"a) Căn của 0.0196 là {method(0.0196):.9f}")
12    print(f"b) Căn của 1.21 là {method(1.21):.9f}")
13    print(f"c) Căn của 2 là {method(2):.9f}")
14    print(f"d) Căn của 3 là {method(3):.9f}")
15    print(f"e) Căn của 4 là {method(4):.9f}")
16    print(f"f) Căn của {225/256} là {method(225/256):.9f}")
```

⁵Trong nhiều ngôn ngữ thì khả năng này của hàm bị hạn chế, chúng thường được xếp vào nhóm “second-class” để phân biệt với nhóm của dữ liệu là “first-class”.

```

17 cal_sqrt(math.sqrt, "math's sqrt")
18 cal_sqrt(lambda x: pow(x, 0.5), "built-in pow")
19 cal_sqrt(lambda x: math.pow(x, 0.5), "math's pow")
20 cal_sqrt(lambda x: x ** 0.5, "exponentiation operator")
21 cal_sqrt(Newton_sqrt, "Newton's sqrt")
22

```

Ta không cần định nghĩa (tường minh) các hàm bọc nữa mà khai báo nó trực tiếp tại nơi cần (là các đối số cho tham số method của hàm `cal_sqrt` như các Dòng 19-21) bằng **biểu thức lambda** (lambda expression) với cú pháp:

lambda <Paras>: <Expr>

Trong đó, `lambda` là từ khóa và dấu `:` là bắt buộc. Giá trị của biểu thức này là một hàm và vì không có tên cho nó nên được gọi là **hàm vô danh** (anonymous function) hay **hàm lambda** (lambda function). Hàm này nhận các tham số cho bởi <Paras> và có “thân hàm” rất ngắn và đơn giản là biểu thức <Expr>. Khi hàm này được gọi thực thi, sau khi truyền đối số cho tham số, Python lượng giá <Expr> và trả về giá trị của nó làm giá trị trả về của hàm. Lưu ý là không có cặp ngoặc tròn bao <Paras> như trong định nghĩa hàm thông thường. Hơn nữa, ta cũng sẽ để trống nếu không có tham số.

Cũng lưu ý, “thân hàm” `lambda` chỉ là một biểu thức (không phải là lệnh hay khối lệnh) nên nó chỉ phù hợp để viết các hàm ngắn như hàm bọc, do đó, ta không thể viết hàm `lambda` cho hàm `Newton_sqrt` trên. Cũng tránh vô duyên khi viết:

```

18 cal_sqrt(lambda x: math.sqrt(x), "math's sqrt")

```

Cũng cần biết, `lambda` có thể được xem là toán tử (vì nó giúp tạo biểu thức `lambda` với kết quả là một hàm) và toán tử này có độ ưu tiên rất thấp, chỉ cao hơn toán tử `:=`.

8.6 Lập trình hướng sự kiện

Trong chương trình `turtle_draw.py` ở Phần 7.7, người dùng hướng dẫn con rùa bò bằng các chuỗi lệnh để vẽ hình. Việc này không được thuận tiện lắm. Trong phần này, dùng kỹ thuật tương tự phần trên (truyền/nhận hàm như các đối tượng dữ liệu khác và biểu thức `lambda`), ta có thể viết chương trình để người dùng hướng dẫn con rùa bò bằng các phím mũi tên trên bàn phím như sau:

```

— https://github.com/vqhBook/python/tree/master/lesson08/turtle\_draw.py —
1 import turtle as t
2
3 def forward(deg):
4     t.setheading(deg)
5     t.forward(d)
6

```

```

7  d = 20
8  t.shape("turtle")
9  t.speed("slowest")
10 t.onkey(lambda: forward(180), "Left")
11 t.onkey(lambda: forward(0), "Right")
12 t.onkey(lambda: forward(90), "Up")
13 t.onkey(lambda: forward(270), "Down")
14 t.onkey(t.bye, "Escape")
15 t.listen()
16 t.mainloop()

```

Hàm `onkey` của module `turtle` cho phép “**đăng kí**” (register) thực hiện một “hành động” nào đó khi người dùng nhấn một phím nào đó (đúng ra là nhả phím). “Hành động” này được mô tả bởi một hàm (mà trong trường hợp này là một thủ tục không tham số) được gọi là **trình xử lý sự kiện** (event handler), mà sẽ được **kích hoạt** (trigger) thực thi khi **sự kiện** (event) phím nhấn tương ứng xảy ra.

Chẳng hạn, ở Dòng 10, ta đăng kí một thủ tục làm trình xử lý sự kiện cho sự kiện nhấn phím mũi tên trái (phím Left). Vì thủ tục này đơn giản là bọc hàm `forward` (với đối số 180°) nên ta dùng biểu thức `lambda` để tạo nó. Kết quả, trong cửa sổ con rùa, nếu người dùng nhấn (đúng ra là nhả) phím mũi tên trái thì hàm vô danh này được gọi chạy, mà như “thân” của nó cho thấy, hàm `forward` được gọi chạy với đối số 180°. Mà điều này có nghĩa là ta di chuyển con rùa qua trái một khoảng `d` (Dòng 4-5). Biến `d` xác định khoảng cách mỗi lần di chuyển của con rùa và ta đặt nó ở ngoài chương trình thay vì trong hàm `forward` (Dòng 7) để cho người dùng nhập giá trị này nếu muốn.

Tương tự, ta đăng kí xử lý các phím mũi tên phải, lên, xuống bằng các hàm vô danh (đúng hơn là thủ tục vô danh) tương ứng mà công việc của chúng là bọc hàm `forward` để di chuyển tương ứng con rùa qua phải, lên trên, xuống dưới. Ta cũng đăng kí thủ tục `turtle.bye` cho phím Escape (Esc) mà công việc của nó là đóng cửa sổ con rùa. Lưu ý là hàm `forward` của chương trình “tình cờ” trùng tên với hàm `forward` của module `turtle` nhưng Python phân biệt để dàng hàm nào bằng cách viết (`forward` viết không là của chương trình còn `t.forward` là của module `turtle` mà ta đã đặt lại tên là `t`).

Dòng 15 gọi hàm `listen` để đặt **tiêu điểm bàn phím** (keyboard focus) cho cửa sổ con rùa (nhờ đó con rùa mới nhận được sự kiện nhấn phím). Cuối cùng, ta cần gọi hàm `mainloop` để con rùa thực hiện **vòng lặp thông điệp** (event loop, message loop): theo dõi, nhận sự kiện và xử lý sự kiện tương ứng; cho đến khi cửa sổ con rùa bị đóng (nhấp nút đóng (X)) hoặc hàm `bye` được gọi thì kết thúc.

Mô hình lập trình trên được gọi là **lập trình hướng sự kiện** (event-driven programming). Nó hay được dùng cho các chương trình dạng **giao diện người dùng đồ họa** (Graphical User Interface, GUI), là các chương trình có kết xuất và tương tác đa dạng với người dùng, chứ không chỉ là qua chuỗi (người dùng nhập

chuỗi và chương trình xuất ra chuỗi) như đa số các chương trình của ta đến giờ.⁶ Ta sẽ tìm hiểu thêm mô hình này sau.

8.7 Lệnh biểu thức, hiệu ứng lề và docstring

Thử chương trình minh họa sau:

https://github.com/vqhBook/python/tree/master/lesson08/expression_stm.py

```

1 # This is a comment ...
2 # ... and comment
3
4 """ This is a multiline string literal ...
5     ... but is used as a comment ... """
6
7 1, 2, ..., 100
8 s = 'This is a "real" string literal'
9 print((s + "\n") * 10)
10 2 ** 1000 == pow(2, 1000)
11 pow(2, "1000")
12 print(print(print))

```

Chương trình này khá “vô duyên” nhưng là chương trình Python hợp lệ! Trừ 2 ghi chú ở Dòng 1, 2 và 2 dòng trống 3, 6 không tham gia vào lệnh (Python “không để ý” đến chúng) thì mã này gồm 7 lệnh, đủ 7 lệnh:

- Lệnh 1 (Dòng 4-5): là hằng chuỗi! Là hằng nên nó là một biểu thức (có giá trị là chuỗi tương ứng) mà khi đứng riêng rẽ thì đủ tư cách là một lệnh gọi là **lệnh biểu thức** (expression statement). Tuy nhiên, lệnh này “vô nghĩa” vì khi chạy trong chế độ chương trình, Python bỏ qua giá trị của nó.
- Lệnh 2 (Dòng 7): là một biểu thức nên cũng là lệnh biểu thức. Biểu thức này mô tả một bộ gồm 4 thành phần (ta đã thấy bộ 2 thành phần ở Phần 2.4 và bộ 3 thành phần ở Bài tập 4.1, 5.3, 5.6). Đặc biệt, thành phần thứ 3 là **dấu chấm lửng** (ellipsis) được mô tả bởi 3 dấu chấm viết liền (do đó thường được gọi là dấu 3 chấm). Lệnh biểu thức này cũng “vô nghĩa”.
- Lệnh 3 (Dòng 8): là một lệnh gán bình thường mà sau đó biến s được gán giá trị là chuỗi tương ứng. Rõ ràng lệnh này “có nghĩa”.
- Lệnh 4 (Dòng 9): là một lời gọi hàm mà hiển nhiên là rất “có nghĩa” vì chuỗi tương ứng được xuất ra. Đặc biệt, nên nhớ, nó cũng là một lệnh biểu thức. Bản thân `print((s + "\n") * 10)` là một biểu thức như bao biểu thức khác. Tuy nhiên, ý nghĩa không nằm ở giá trị biểu thức (là None), mà nằm ở **“hiệu ứng lề”** (side effect) của nó, cụ thể, trong quá trình “tính” biểu thức này, Python gọi thực thi hàm `print` mà hàm này xuất ra một chuỗi.

⁶Các chương trình dạng này được gọi là có **giao diện console** hay **giao diện dòng lệnh** (Command-line Interface, CLI hay Command/Console User Interface, CUI).

- Lệnh 5 (Dòng 10): lại là một lệnh biểu thức “vô nghĩa”, mặc dù, với biểu thức này, Python phải tính khá vất vả.
- Lệnh 6 (Dòng 11): là lệnh biểu thức, dù đơn giản hơn nhưng lại có hiệu ứng lẽ. Cụ thể, có lỗi thực thi khi Python thực hiện lệnh này (lỗi `TypeError` vì đối số thứ 2 không là số). Thế nào là “có hiệu ứng” hay “có nghĩa” cũng khó nói rõ ràng bây giờ. Bạn có thể hình dung, nếu ta bỏ đi được một lệnh biểu thức (mà kết quả thực thi chương trình không “khác biệt” gì) thì biểu thức tương ứng đó là không có hiệu ứng lẽ, nghĩa là vô nghĩa (nếu để). Chẳng hạn, vì nếu bỏ đi lệnh này, thì chương trình không còn bị lỗi thực thi và có thêm kết xuất của lệnh ở dưới nên lệnh này có hiệu ứng lẽ.
- Lệnh 7 (Dòng 12): là lệnh biểu thức khá thú vị. Bạn tự phân tích thử xem (nhớ rằng hàm cũng là đối tượng).

Nhân tiện, nhiều lập trình viên “lợi dụng” việc không có hiệu ứng lẽ của Lệnh 1 để tạo các ghi chú trên nhiều dòng (chẳng hạn để “tắt” một đoạn mã dài). Đây là mảnh khoe “hay” vì Python chỉ có ghi chú trên 1 dòng với dấu `#` nên bạn phải gõ nhiều lần dấu này cho ghi chú trên nhiều dòng; tuy nhiên, bạn không nên dùng như vậy vì nó không “chính danh” lắm.⁷

Dấu chấm lửng ở Lệnh 2 cũng đáng để bàn thêm. Trước hết, nó mô tả giá trị duy nhất của kiểu `ellipsis` mà cũng có thể được mô tả bằng tên dựng sẵn `Ellipsis`. Khác với `None`, giá trị này được hiểu là có giá trị (bằng chứng là theo cách hiểu luận lý mở rộng, Python xem dấu chấm lửng là `True`) nhưng là giá trị chưa cụ thể (nên nó được gọi là “lửng”). Cách hiểu này tương tự cách hiểu thông thường của dấu chấm lửng.⁸ Chẳng hạn, dấu chấm lửng trong Lệnh 2 đã được dùng như trong Toán để mô tả “tiếp tục tương tự” (lưu ý, điều đó không có nghĩa là bộ mô tả ở Lệnh 2 gồm 100 thành phần, nó vẫn chỉ có 4 thành phần với thành phần thứ 3 là dấu chấm lửng).

Phổ biến hơn, dấu chấm lửng được dùng để mô tả giá trị chưa cụ thể mà sẽ được cụ thể sau. Cách hiểu này tương tự như `pass` nhưng mạnh hơn vì `pass` là lệnh còn dấu chấm lửng là hằng (mà như vậy cũng có thể dùng như lệnh biểu thức). Chẳng hạn, ta có thể dùng dấu chấm lửng như sau:

```

1 def f(x): ...
2
3 a = ...
4 if a != 0: ...
5 else: print("a is zero!")

```

Các lệnh trên hoàn toàn hợp lệ nhưng, dĩ nhiên, ta sẽ phải “điền vào 3 chấm”. Lưu ý, ta không được phép dùng `pass` thay cho dấu 3 chấm ở Dòng 3. Ta sẽ thấy vài cách dùng hay khác của dấu chấm lửng trong các bài sau.

⁷Đây là điều khá tào lao trong Python. Các IDE thường hỗ trợ việc tạo ghi chú và bỏ ghi chú (như chức năng “Comment Out Region” và “Uncomment Region” trong IDLE).

⁸Ta đã hoang mang với `None`, giờ thì mơ hồ với ...

Thử một chương trình khác có lệnh biểu thức với hiệu ứng lề thú vị sau:

```

1  https://github.com/vqhBook/python/tree/master/lesson08/docstring.py
2  def Newton_sqrt(x):
3      "Tính căn theo phương pháp Newton."
4      y = x
5      for i in range(100):
6          y = y/2 + x/(2*y)
7      return y
8
9  print(f"Căn của 2 theo Newton: {Newton_sqrt(2):.9f}")
10 print(Newton_sqrt.__doc__)
    help(Newton_sqrt)

```

Hàm `Newton_sqrt` được viết như ở Phần 8.2, ngoại trừ việc có thêm lệnh biểu thức, mà thực ra chỉ là một hằng chuỗi, ở lệnh đầu tiên của thân hàm (Dòng 2). Python cho phép lệnh đầu tiên của một hàm có thể là một hằng chuỗi, gọi là **chuỗi sớ liệu** (documentation string, docstring), vì chuỗi này (nếu có) sẽ được lưu trữ cùng với hàm làm thông tin mô tả thêm cho hàm. Thông tin này có thể được truy cập bằng “thuộc tính” đặc biệt `__doc__` (Dòng 9) và thường được phân tích thêm bởi các công cụ tự động tạo thông tin tra cứu như hệ thống tra cứu nội tại với hàm `help` (Dòng 10).

Việc dùng chuỗi sớ liệu cho hàm là khá phổ biến và chuỗi này thường rất dài (dùng dấu 3-nháy `"""`), mô tả nhiều thông tin chứ không đơn giản như ví dụ trên của ta. Cũng lưu ý, ngoài hiệu ứng lề “nhỏ nhỏ” là sớ liệu thì chuỗi sớ liệu không có bất kì hiệu ứng nào khác lên hàm hay chương trình (theo như hứa hẹn của Python).

8.8 Tự viết lấy module

Ta đã phải chép lại nhiều lần định nghĩa hàm `Newton_sqrt` trên trong các chương trình có dùng nó. Rõ ràng, đây không phải là cách làm tốt vì *mục đích của hàm là “viết một lần, gọi khắp nơi” (Define One, Call Anywhere)!⁹* Để giải quyết vấn đề này, ta viết riêng hàm trong một module để có thể dùng (gọi) nó trong nhiều chương trình khác nhau. Một **module** là một file mã Python như ta đã viết trước giờ, tuy nhiên, nó được dùng với ý nghĩa là cung cấp các hàm hỗ trợ các dịch vụ nào đó (tương tự như các module trong thư viện chuẩn mà ta đã dùng). Bạn tạo file mã `sqrt.py` chứa mã sau:

```

1  https://github.com/vqhBook/python/tree/master/lesson08/sqrt.py
2  """Module hỗ trợ việc tính căn bằng phương pháp Newton.
3  Tác giả: Vũ Quốc Hoàng.

```

⁹Bạn sẽ không hiểu hết câu chơi chữ này nếu bạn không biết về **Java**, một ngôn ngữ lập trình phổ biến khác.

```

4  Ngày viết: 10-03-2020.
5  """
6
7  def Newton_sqrt(x):
8      """Tính căn theo phương pháp Newton.
9
10     Số cần tính căn, x, phải là số dương.
11     """
12     y = x
13     for i in range(100):
14         y = y/2 + x/(2*y)
15     return y

```

Bạn cần nhận thấy rằng mã nguồn của module này chỉ có định nghĩa hàm nên khi chạy như chương trình thì chẳng có điều gì xảy ra (thật ra các hàm trong module sẽ được tạo và được gán cho các tên hàm tương ứng). Đúng vậy! Nó được tạo ra với mục đích khác: cung cấp các dịch vụ cho các chương trình (hay module khác) dùng. Tuy nhiên bạn có thể gọi hàm `Newton_sqrt` nếu muốn như minh họa sau:

```

===== RESTART: D:\Python\lesson08\sqrt.py =====
1  >>> Newton_sqrt(2)
    1.414213562373095

```

Đây là vì khi chạy module `sqrt` trên (file mã `sqrt.py`), Python đã tạo hàm `Newton_sqrt` mà khi chạy xong, do IDLE không kết thúc phiên làm việc nên ta vẫn có thể tiếp tục dùng hàm `Newton_sqrt` (xem lại Phần 4.2).

Cách dùng module thông thường nhất là nạp nó vào chương trình và dùng các hàm (cùng với các “thứ” khác nếu có) của module như cách ta đã làm với các module chuẩn. Tuy nhiên, khác với module chuẩn, ta cần có cách để cho Python biết file mã chứa module mà đơn giản nhất là đặt file mã module cùng thư mục với file mã chương trình dùng nó (xem Bài tập 8.8). Chẳng hạn, tạo file mã `sqrt_program.py` cùng thư mục với file mã `sqrt.py` có nội dung:

```

1  import math
2  import sqrt
3
4  x = float(input("Nhập số cần tính căn: "))
5  print(f"Căn của {x} theo Newton: {sqrt.Newton_sqrt(x):.9f}")
6  print(f"Căn của {x} theo math's sqrt: {math.sqrt(x):.9f}")

```

rồi chạy như thông thường. Lưu ý là cách ta dùng module `sqrt` hoàn toàn giống cách ta dùng module `math` và các module chuẩn khác (ngoại trừ vấn đề đường dẫn file mã như đã nói).

Lưu ý, trong module `sqrt`, ta dùng hằng chuỗi trên nhiều dòng để tạo sùu liệu chi tiết hơn cho hàm `Newton_sqrt`. Ngoài ra, ta cũng tạo chuỗi sùu liệu cho

module bằng hàng chuỗi viết ở đầu module. Để ý là trong chuỗi sưu liệu nhiều dòng thì sau dòng đầu tiên mô tả chung là một dòng trống rồi mới đến các dòng mô tả chi tiết. Đây (cùng với các qui định khác nữa) là khuyến cáo của PEP 8.¹⁰ Cũng lưu ý là, trong Python, chương trình và module không khác biệt nhiều (xem Bài tập 8.10).

Bạn hãy xem lại Bài tập 7.7, chẳng hạn Câu (a). Giả như ta có một hàm hỗ trợ ta vẽ (và tô) hình vuông với kích thước nào đó, tại vị trí nào đó, với màu nào đó; thì ta có thể dễ dàng dùng nó vẽ hình bàn cờ với các hình vuông khác màu xen kẽ (Bài tập 7.7a). Ta cũng thấy rằng một hàm như vậy khá là “cơ bản”, nó có thể hỗ trợ ta vẽ nhiều hình phức tạp khác nữa, nghĩa là nó có thể được dùng trong nhiều chương trình khác nhau.

Rất tiếc, module turtle (hay Python nói chung) không hỗ trợ hàm nào như vậy cả. OK, Fine! Ta sẽ tự mình viết lấy module chứa hàm này (và các hàm tương tự khác). Tạo file mã figures.py như sau:

<https://github.com/vqhBook/python/tree/master/lesson08/figures.py>

```

1 import turtle as t
2
3 def square(x, y, width, line_color="black", fill_color=None):
4     t.up(); t.goto(x, y); t.down()
5     t.setheading(0); t.pencolor(line_color)
6     if fill_color:
7         t.fillcolor(fill_color)
8         t.begin_fill()
9     for _ in range(4):
10         t.forward(width)
11         t.left(90)
12     if fill_color:
13         t.end_fill()
```

Ta dự định tạo một module hỗ trợ vẽ các hình cơ bản (hình vuông, hình tròn, tam giác, ...) nên ta đặt tên module là figures mà Python qui định tên file mã của module là tên module nên ta đặt tên file là figures.py. Hơn nữa, hiện giờ module mới hỗ trợ vẽ hình vuông nên mới chỉ có hàm square như định nghĩa trên, ta sẽ thêm các hàm hỗ trợ vẽ các hình khác vào sau.

Tôi không bàn gì thêm về thân hàm square, bạn đã quen thuộc tất cả (nếu không thì tự coi lại nhé), ngoại trừ cách dùng giá trị None thông minh cho tham số fill_color (và cách mô tả điều kiện của 2 lệnh if, nhớ rằng None được Python xem là False). Đây là tham số xác định màu tô (còn line_color xác định màu nét vẽ) và giá trị None (cũng là giá trị mặc định) ám chỉ là không có màu tô tức là không tô màu. Màu là None, nghĩa là không có màu! Đây cũng là cách dùng phổ biến của None: *tham số nhận giá trị None nghĩa là không có giá trị*.

¹⁰Để tiết kiệm giấy, từ đây, tôi sẽ không dùng các chuỗi sưu liệu (cũng như ghi chú) trong các chương trình minh họa nhưng bạn nên tập thói quen viết chuỗi sưu liệu vì nó có nhiều ích lợi (tương tự và hơn ghi chú).

Chương trình sau minh họa việc dùng hàm `square` trong module trên để vẽ hình bàn cờ như ta đã nói (nhớ đặt file mã module chung với file mã chương trình):

https://github.com/vqhBook/python/tree/master/lesson08/chess_board.py

```
1 import turtle as t
2 import figures
3
4 t.hideturtle(); t.tracer(False)
5 width = 30
6 for i in range(8):
7     for j in range(8):
8         color = ("blue" if (i + j) % 2 == 0 else "green")
9         figures.square(i*width, j*width, width, color, color)
10 t.update()
```

Cũng hàm `square` trong module `figures` lại được ta dùng trong một chương trình khác như sau (cũng vậy, đặt file mã module chung với file mã chương trình):

https://github.com/vqhBook/python/tree/master/lesson08/random_square.py

```
1 import turtle as t
2 import random
3 import figures
4
5 t.hideturtle(); t.speed("fastest"); t.colormode(1.0)
6 for _ in range(100):
7     w_width = t.window_width()
8     w_height = t.window_height()
9     x = random.randint(-w_width//2, w_width//2)
10    y = random.randint(-w_height//2, w_height//2)
11    width = min(random.randint(0, w_width//2 - x),
12                random.randint(0, w_height//2 - y))
13    figures.square(x, y, width,
14                  (random.random(), random.random(), random.random()))
```

Ta đã gọi hàm `square` mà không có đối số cho tham số `fill_color` và do đó nó nhận giá trị mặc định `None`, cho nên các hình vuông được vẽ ra không được tô màu (quan trọng trong chương trình này vì ta không muốn các hình vuông ở trên che các hình vuông bên dưới). Dĩ nhiên, ta có thể đưa tường minh đối số `None` cho tham số `fill_color`, nhưng cách viết này hay hơn: không đưa hiểu là không có, do đó hiểu là không tô!

Tóm tắt

- Hàm là phương tiện quan trọng nhất của lập trình, đó là công việc được tham số hóa. Định nghĩa hàm giúp tạo hàm và gán nó cho tên hàm. Gọi hàm giúp dùng hàm, tức là gán đối số cho tham số, thực thi thân hàm, và trở về với giá

trị trả về của hàm.

- Hàm trả về None được xem là không có giá trị trả về và được gọi cụ thể hơn là thủ tục. Các hàm khác được gọi là hàm có trả về giá trị (hay gọi chung là hàm). Lệnh `return` xác định giá trị trả về của hàm và nó cũng kết thúc việc thực thi hàm. Khi xong thân hàm, Python cũng tự động thực thi lệnh `return` không đổi số mà ý nghĩa chính là `return None`.
- Việc truyền đối số cho tham số có thể được thực hiện theo thứ tự hoặc/và theo tên tham số. Hơn nữa, tham số cũng có thể được khai báo để nhận giá trị mặc định khi không có đối số tương ứng cho nó trong lời gọi hàm.
- Hàm cũng là các đối tượng có thể được truyền nhận và thao tác như dữ liệu. Tuy nhiên, ngữ nghĩa của hàm là “thực thi khối lệnh được tham số hóa” chứ không phải là bản thân nó (value) như dữ liệu.
- Hàm bọc là các hàm nhỏ, ngắn cung cấp “giao diện” cho các hàm hay toán tử khác. Trong đa số trường hợp, ta có thể dùng biểu thức lambda để tạo hàm vô danh thay cho việc định nghĩa tường minh hàm bọc.
- Lập trình sự kiện là kỹ thuật đăng ký các trình xử lý sự kiện, là các hàm được gọi chạy khi sự kiện tương ứng xảy ra. Mô hình này được dùng nhiều trong lập trình GUI và các công nghệ khác.
- Các biểu thức, thường có hiệu ứng lề, viết riêng trên một dòng là lệnh biểu thức mà điển hình là lời gọi các thủ tục.
- Chuỗi đầu tiên của thân hàm (hay chương trình, module) được gọi là doc-string. Nó thường mô tả chức năng chung của hàm (hay chương trình, module) đó và được truy cập bằng thuộc tính `__doc__`.
- Module cũng là file mã như chương trình nhưng được dùng với ý nghĩa cung cấp các hàm hỗ trợ các dịch vụ nào đó. Các hàm có thể được định nghĩa 1 lần trong các module và dùng nhiều lần ở các module hay chương trình khác.

Bài tập

8.1 Viết hàm (với tham số thích hợp) và chương trình minh họa việc dùng hàm đó:

- Tính giai thừa.
- Tính mũ lũy thừa của 2 (x^{2^n}) (Bài tập 7.4).
- Xuất bài hát “Happy birthday” cho một tên (Bài tập 4.4).
Gợi ý: nên đặt giá trị mặc định “you” cho tham số.
- Xuất đoạn trích “Roméo và Juliet” cho 2 tên tương ứng (Bài tập 4.5).
Gợi ý: nên đặt giá trị mặc định tương ứng cho các tham số là Roméo và Juliet. Bạn cũng nên thử dùng cách truyền đối số theo tên khi gọi hàm.
- Chuyển đổi giữa độ F và độ C tùy theo chế độ đổi (F sang C hay C sang F) (Bài tập 6.3).

Gợi ý: bạn có thể dùng tham số luận lý cho chế độ đối (chẳng hạn giá trị True là F sang C, còn False là C sang F).

8.2 Viết hàm (với tham số thích hợp) và chương trình minh họa việc dùng hàm đó:

- (a) Tính tiền điện sinh hoạt từ lượng điện tiêu thụ (Bài tập 6.5).
- (b) Tính $S(x, n)$ ở Bài tập 7.1f.
- (c) Tính tổng các chữ số của một số nguyên không âm (Bài tập 7.3c).
- (d) Dùng con rùa, vẽ hình “ngôi sao” có bán kính và số cánh được cho (Bài tập 7.5).

8.3 Số nguyên tố. Một số nguyên lớn hơn 1, được gọi là **số nguyên tố** (prime number) nếu nó chỉ có 2 ước số dương là 1 và chính nó, ngược lại, số đó được gọi là **hợp số** (composite number). Một số nguyên a (khác 0) được gọi là **ước số** (divisor) của số nguyên b nếu b chia hết cho a , tức là b chia a dư 0.

Viết hàm kiểm tra một số nguyên dương được cho có là số nguyên tố hay không và dùng hàm này giúp kiểm tra các số người dùng nhập có nguyên tố không.

Gợi ý:

- Hàm này nên trả về giá trị luận lý (True là nguyên tố còn False là không).
- Trước mắt, bạn có thể thử dùng phương pháp “mò nghiệm” ở Phần 7.4. Ta sẽ còn trở lại với các thuật toán tốt hơn vì bài toán **kiểm tra tính nguyên tố** (primality testing) có vai trò rất quan trọng trong khoa học máy tính.

8.4 Viết lại chương trình `hello.py` ở Phần 8.1 (đặc biệt là hàm `sayhello`) để có phiên bản “đồ họa”, nghĩa là dùng con rùa để thực sự xuất ra cái khung tên (như Bài tập 5.7c).

Gợi ý: bạn cũng có thể cho người dùng nhập tên bằng hộp thoại nhập của con rùa như Bài tập 5.7. Để hiển thị khung tên cho nhiều tên, bạn có thể cho “lần lượt” hiển thị từng khung tên trong cửa sổ con rùa trong một khoảng thời gian nào đó (dùng `turtle.reset` và `time.sleep`).

8.5 Trong chương trình `sqrt_cal.py` ở trên, ta đã phải dùng tham số `method_name` cho biết “tên” phương pháp. Viết lại chương trình, dùng `docstring` của hàm tương ứng làm “tên” phương pháp (bỏ tham số `method_name` cũng như các đối số tương ứng khi gọi).

8.6 Dùng kĩ thuật hàm vô danh, tương tự chương trình `sqrt_cal2.py` ở Phần 8.5, viết chương trình làm lại Bài tập 3.5(b)-(e), 3.6, 3.7.

8.7 Viết lại chương trình `turtle_draw.py` ở Phần 8.6, cũng dùng kĩ thuật hàm vô danh, để:

- Chỉ cho con rùa quay đầu trái, phải, trên, dưới khi nhấn các phím mũi tên Left, Right, Up, Down.
- Cho con rùa bò tới và vẽ (theo hướng hiện tại của nó) khi nhấn phím Enter (tên phím là "Return").

- Cho con rùa bò tới nhưng không vẽ (tức là “nhảy”) khi nhấn phím Space (tên phím là "space").

8.8 Đường dẫn hệ thống. Khi gặp lệnh nạp module (`import`), Python dò tìm file mã có tên tương ứng (tên file trùng tên với module và có đuôi `.py` hoặc một số đuôi khác) trong một danh sách các thư mục. Danh sách này được gọi là **đường dẫn hệ thống** (system path). Ta có thể cấu hình danh sách này trên máy của mình (trên Windows là biến môi trường `PYTHONPATH`).¹¹ Trên IDLE, ta có thể xem danh sách này bằng `File → Path Browser`. Danh sách này gồm nhiều thư mục được liệt kê theo thứ tự ưu tiên (Python tìm file mã trùng tên đầu tiên). Các module chuẩn được đặt trong các thư mục này. Chẳng hạn, dùng `File → Open Module`, rồi gõ tên `turtle` trong hộp thoại hiện ra, rồi nhấn OK, Python sẽ mở file mã `turtle.py`, đây chính là file mã của module `turtle` yêu cầu.¹² Quan sát trên đầu cửa sổ IDLE ta sẽ thấy đường dẫn của thư mục đặt các module chuẩn này.

Cách “bá đạo” để ta có thể dùng một module là đặt nó vào một thư mục nằm trong đường dẫn hệ thống, chẳng hạn, chung thư mục với file mã `turtle.py`. Khi đó module có thể được dùng giống như module chuẩn. Tuy nhiên, cách này khá nguy hiểm vì ta có thể lỡ ghi đè, xóa, sửa các file mã của module chuẩn.

Cách an toàn và đơn giản nhất là đặt các file mã module chung với file mã chương trình dùng module. Cách linh hoạt hơn là để file mã module cố định trong một thư mục và thêm đường dẫn thư mục này vào đường dẫn hệ thống khi dùng module. Để làm điều này, ta có thể truy cập biến `sys.path` (biến `path` trong module chuẩn `sys`) hoặc tốt hơn, dùng hàm `os.chdir` (hàm `chdir` của module chuẩn `os`) để thay đổi **thư mục làm việc** (working directory) mà thư mục này là thư mục đầu tiên (ưu tiên nhất) trong đường dẫn hệ thống.¹³ Chẳng hạn, với file mã `sqrt.py` đặt ở thư mục `D:\Python\lesson08` (module `sqrt` ở Phần 8.8), khởi động IDLE và dùng module này như sau:

```
1 >>> import os
2 >>> os.chdir(r"D:\Python\lesson08")
3 >>> import sqrt
4 >>> sqrt.Newton_sqrt(2)
    1.414213562373095
5 >>> help(sqrt.Newton_sqrt)
    Help on function Newton_sqrt in module sqrt:
    ...
6 >>> help(sqrt)
```

¹¹ Bạn không nên làm nếu không rành về hệ thống

¹² Các module chuẩn khác cũng có thể được xem tương tự nhưng một số module được viết bằng ngôn ngữ khác như `math` thì không có file mã Python.

¹³ Thật ra Python có phân biệt các module trong thư viện chuẩn. Các **module dựng sẵn** (built-in module) như `math`, `time`, `os`, `sys`, ... là cực kì quan trọng nên luôn được ưu tiên nạp (dù tên module của ta có trùng) còn các module khác thì không quan trọng (sẽ nạp module của ta nếu trùng tên) như `turtle`, `random`, ...

Help on module `sqrt`:

...

Dĩ nhiên, bạn cũng thể làm tương tự để dùng module trong file mã chương trình. Sở dĩ việc để file mã module chung thư mục với file mã chương trình có tác dụng vì khi chạy chương trình, Python sẽ tự động đặt thư mục làm việc là thư mục chứa file mã. Bạn có thể xem thư mục hiện hành bằng hàm `os.getcwd`.

- (a) Tra cứu `sys.path`, `os.chdir` và `os.getcwd`.
- (b) Dùng module `figures` từ IDLE mà không chạy từ file mã của module.
- (c) Dùng module `figures` từ file mã đặt ở thư mục khác.

8.9 Tương tự như hàm vẽ hình vuông trong module `figures`, bổ sung vào module này:

- (a) Hàm `line` vẽ đoạn thẳng nối 2 điểm có tọa độ được cho với màu được cho và dùng nó trong các chương trình vẽ các hình trong Bài tập 5.4(b), (c).

Lưu ý: nhớ vẽ cả các ô lưới (có thể dùng màu xám (gray)).

- (b) Hàm `triangle` vẽ và tô tam giác (với các tham số thích hợp) và dùng nó trong các chương trình vẽ các hình còn lại trong Bài tập 7.7.
- (c) Hàm `circle` vẽ và tô hình tròn (với các tham số thích hợp) và sửa lại mã `random_square.py` để có các hình tròn ngẫu nhiên thay vì hình vuông.
- (d) Hàm `rectangle` vẽ và tô hình chữ nhật (với các tham số thích hợp).

Gợi ý: Hình chữ nhật là hình “cơ bản” hơn hình vuông vì hình vuông là hình chữ nhật đặc biệt (có chiều dài bằng chiều rộng). Do đó ta có thể mở rộng mã của hàm vẽ hình vuông để có hàm vẽ hình chữ nhật. Sau khi có hàm vẽ hình chữ nhật (`rectangle`), ta cũng nên viết lại hàm vẽ hình vuông (`square`) bằng cách gọi hàm vẽ hình chữ nhật (`rectangle`) với tham số `width` làm đối số cho cả chiều dài và chiều rộng.

- (e) Hàm `pie` vẽ và tô hình quạt (với các tham số thích hợp).

Gợi ý: 2 tham số quan trọng là góc bắt đầu và góc kết thúc. Chẳng hạn, góc tư hình tròn trên phải là hình quạt có góc bắt đầu là 0° và góc kết thúc là 90° (tính theo chiều ngược kim đồng hồ). Sau khi có hàm này (`pie`), ta cũng nên viết lại hàm vẽ hình tròn (`circle`) bằng cách gọi hàm vẽ hình quạt (`pie`) với tham số góc bắt đầu là 0° và góc kết thúc là 360° . Bạn cũng nên đặt các giá trị mặc định này cho các tham số tương ứng, khi đó, thậm chí, ta không cần định nghĩa hàm `circle` vì nó chính là `pie` với giá trị mặc định.

8.10 Khác biệt giữa chương trình và module. Không có khác biệt rõ ràng giữa chương trình và module Python (chúng thường được gọi chung là module). Cả 2 đều được viết trong một file mã Python và được thực thi theo cùng cách. Khác biệt đến từ ý định của ta, chương trình thường chứa mã thực hiện một

công việc “lớn” nào đó còn module chứa các hàm (và các thứ khác) cung cấp cho các chương trình (hay module) khác dùng.

Python hỗ trợ ta phân biệt 2 trường hợp dùng này bằng cách: nếu file mã được thực thi như chương trình (chẳng hạn, bởi chức năng Run Module (F5) của IDLE) thì biến đặc biệt `__name__` sẽ được đặt giá trị là `"__main__"`, ngược lại, nếu file mã được nạp như module (chẳng hạn, bởi các biến thể khác nhau của lệnh `import`) thì nó được đặt giá trị là tên module (chính là tên file mã mà không có phần mở rộng `.py`).

Hơn nữa, một file mã có thể được dự định dùng vừa là chương trình vừa là module: nó cung cấp các hàm (và các thứ khác) cho module (bao gồm chương trình) khác dùng và nó có khối lệnh ngoài cùng (khối lệnh không thụt đầu dòng) chạy như là chương trình. Như vậy, để kiểm tra là được thực thi như chương trình, các module thường dùng lệnh `if` sau:

```
if __name__ == "__main__":
    <các lệnh của chương trình>
```

Lưu ý, không có gì đặc biệt với chữ `main` ngoài qui ước giá trị `"__main__"` cho biến `__name__` nói trên.

Cũng lưu ý là lệnh `import` chỉ nạp module một lần, nghĩa là nếu có nhiều lần thực thi `import` thì chỉ có lần đầu tiên là “chạy và nạp” module. Ngữ nghĩa này là phù hợp với module vì việc chạy và nạp nhiều lần sẽ tốn kém. Tuy nhiên ngữ nghĩa này không phù hợp với chương trình khi ta muốn chạy nó nhiều lần. Do đó ta không nên lạm dụng lệnh `import` để chạy chương trình mà nên chạy bằng các cách “chính thống” khác.

Các chương trình lớn thường gồm nhiều module (về mặt Vật lý là gồm nhiều file mã), trong đó có một module chính, đúng hơn là file mã chính làm **điểm vào** (entry point) hay **file mã chính** (main top-level file) của chương trình. Nghĩa là chạy chương trình từ file mã đó và file mã sẽ dùng thêm các module khác để hoàn thành chương trình. Việc tổ chức nhiều module được hỗ trợ bởi package mà ta sẽ bàn sau.

Viết lại module `sqrt` để nó cũng có thể được dùng như chương trình (tương tự mã `sqrt_program.py`).

8.11 Thử các chức năng IDLE sau trong menu File: Open Module, Path Browser, Module Browser.