

Bí kíp luyện Lập trình nhập môn với Python

Vũ Quốc Hoàng
(Nguyễn Thị Bích Quyên và Đào Thị Kim Ngọc Châu soạn LaTeX)

Bản thảo phần I-II, ngày 01/06/2020.
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	24
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	75
6.1	Luồng thực thi và tiến trình	75
6.2	Lệnh chọn và khối lệnh	76
6.3	if lồng	79
6.4	Toán tử điều kiện và toán tử luận lý	80
6.5	Cuộc bỏ trốn khỏi cửa sổ của con rùa	83
6.6	Bài toán, thuật toán và mã giả	84
6.7	Lệnh pass và chiến lược thiết kế chương trình từ trên xuống	87
	Tóm tắt	87
	Bài tập	88
7	Vượt qua hữu hạn với lệnh lặp	91
7.1	Lặp số lần xác định với lệnh for	91
7.2	Công việc được tham số hóa và biến lặp	93
7.3	Lặp linh hoạt với lệnh while	95
7.4	Vòng lặp lồng	96
7.5	Điều khiển chi tiết lệnh lặp	99
7.6	Lặp vô hạn	100
7.7	Duyệt chuỗi	103
	Tóm tắt	104
	Bài tập	105
8	Tự viết lấy hàm	111
8.1	Định nghĩa hàm/gọi hàm và tham số/đối số	111
8.2	Hàm tính toán, thủ tục và lệnh return	113
8.3	Cách truyền đối số và đối số mặc định	115
8.4	Hàm cũng là đối tượng	117
8.5	Biểu thức lambda và hàm vô danh	119
8.6	Lập trình hướng sự kiện	120
8.7	Lệnh biểu thức, hiệu ứng lề và docstring	122
8.8	Tự viết lấy module	124
	Tóm tắt	127
	Bài tập	128

9	Case study 2: Sức mạnh của lặp	133
9.1	Sức mạnh thực sự của lặp	133
9.2	Phương pháp vét cạn	134
9.3	Phương pháp chia đôi và phương pháp Newton	137
9.4	Tính lặp	139
9.5	Vẽ hình bất kì	142
9.6	Hoạt họa	146
	Tóm tắt	149
	Bài tập	149
10	Đối tượng và quản lý đối tượng	155
10.1	Dữ liệu, thao tác và đối tượng	155
10.2	Cuộc đua kỳ thú của 4 thiếu niên ninja rùa đột biến	157
10.3	Danh tính, kiểu và giá trị	159
10.4	Bất biến và khả biến	161
10.5	Phương thức và trường	163
10.6	Không gian tên	165
10.7	Phạm vi tên	167
10.8	Thuộc tính	171
	Tóm tắt	173
	Bài tập	174
11	Danh sách và chuỗi	179
11.1	Duyệt qua nhiều đối tượng với danh sách	179
11.2	Lưu trữ dữ liệu với danh sách và xử lý ngoại tuyến	182
11.3	Các thao tác trên danh sách	184
11.4	Các danh sách song hành và duyệt với zip	187
11.5	Danh sách lồng	188
11.6	Thao tác trên từng phần tử và bộ tạo danh sách	189
11.7	Chuỗi	190
	Tóm tắt	191
	Bài tập	192
12	Bộ, tập hợp và từ điển	199
12.1	Bộ	199
12.2	Xử lý bảng dữ liệu với danh sách và bộ	201
12.3	Hàm có số lượng đối số tùy ý	203
12.4	Tập hợp	205
12.5	Từ điển	206
12.6	Đối số từ khóa	209
	Tóm tắt	210
	Bài tập	211

13 Case study 3: Ngẫu nhiên, mô phỏng và trực quan hóa	217
13.1 Vẽ biểu đồ với Matplotlib	217
13.2 Đồ họa tương tác với Matplotlib	220
13.3 Tung đồng xu	221
13.4 Trò chơi may rủi	223
13.5 Ngẫu nhiên và trực giác	225
13.6 Vẽ ngẫu nhiên	227
13.7 Tính toán ngẫu nhiên	230
Tóm tắt	232
Bài tập	232
14 Tự viết lấy lớp	239
14.1 Lớp là khuôn mẫu của các đối tượng thể hiện	239
14.2 Đóng gói dữ liệu với thao tác	243
14.3 Tự quản hóa và nhân cách hóa	244
14.4 Kế thừa	245
14.5 Đa hình và nạp chồng toán tử	248
14.6 Tính riêng tư và che dấu dữ liệu	254
14.7 Lập trình hướng đối tượng	255
14.8 Đối tượng duyệt được và bộ duyệt	256
Tóm tắt	258
Bài tập	259

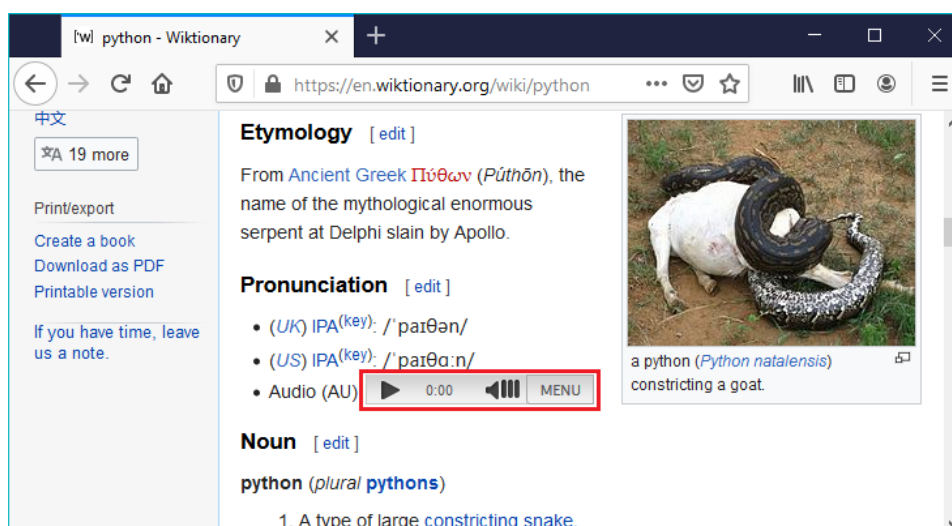
Bài 1

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.

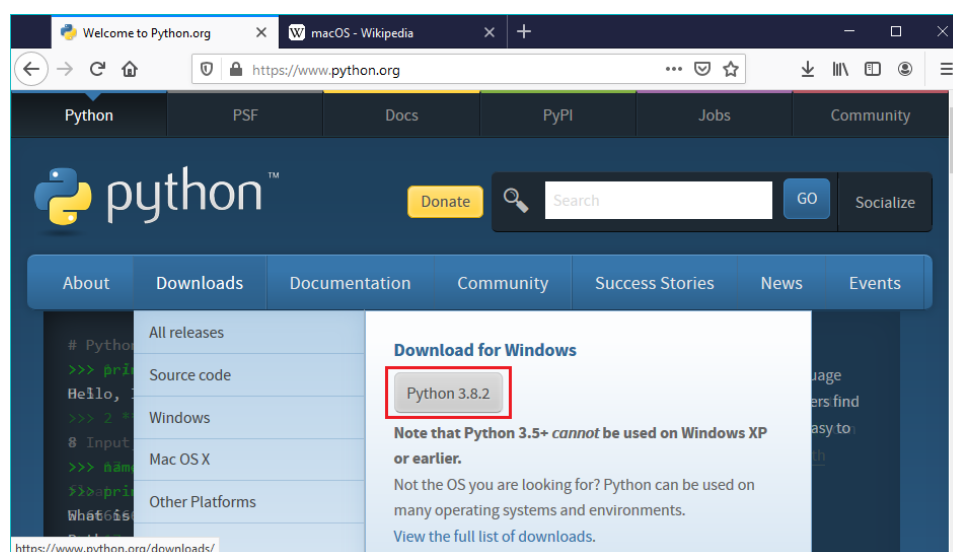
1.1 Cài đặt Python

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.

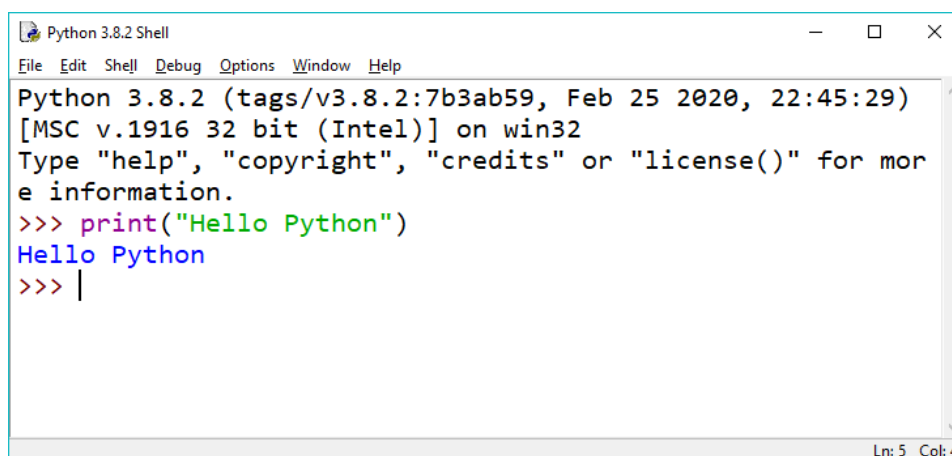
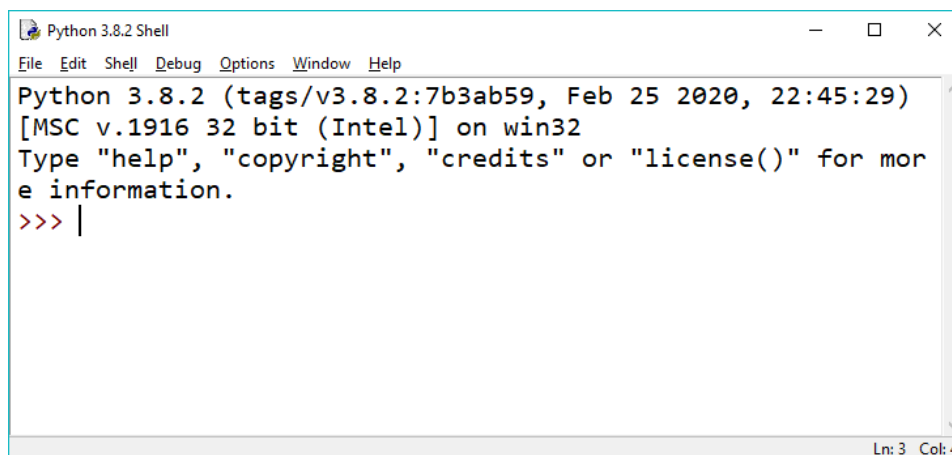
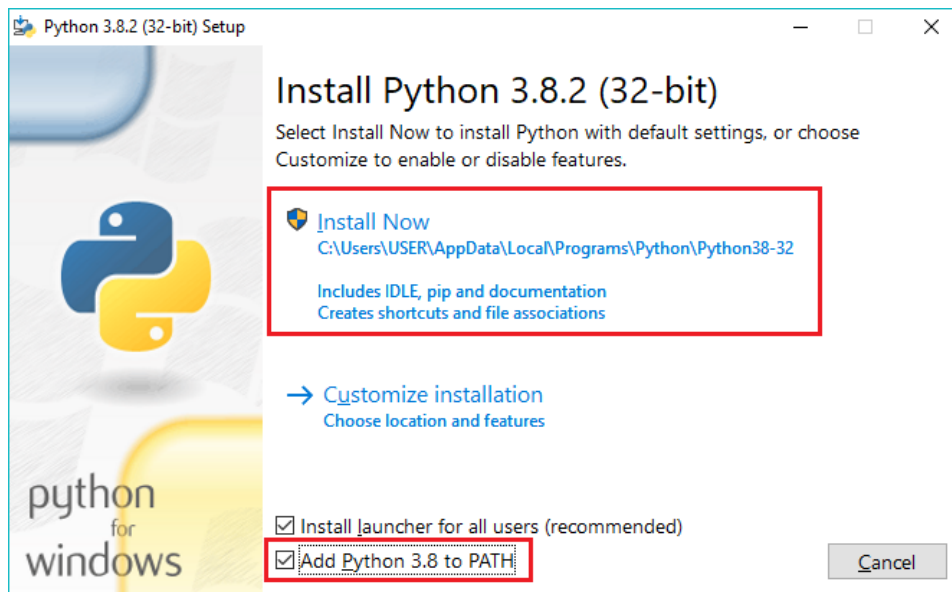
Để cài đặt Python, bạn vào trang chủ của Python ở địa chỉ <https://www.python.org/>, trong thẻ Downloads bạn sẽ thấy link tải file cài đặt Python. Nếu bạn dùng máy Windows thì nhấp nút “Python 3.8.2” như hình dưới. Lưu ý, con số 3.8.2 là số **phiên bản** (version) của Python, nó sẽ thay đổi về sau (càng 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**.



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.

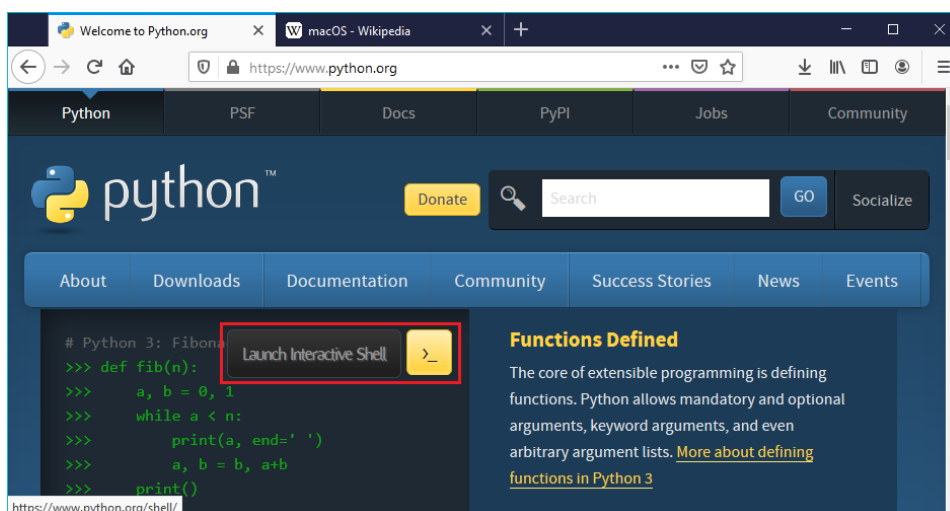
Để chạy Python, từ nút Windows trên thanh Taskbar, bạn nhấp chạy “IDLE (Python 3.8)”. **IDLE** hiện ra và sẵn sàng giúp Python nhận lệnh như hình dưới.² Cửa sổ IDLE này còn được gọi là **Python Shell** và kí hiệu >>> được gọi là **dấu đợi lệnh** (prompt). Bạn nhập yêu cầu, tức là ra lệnh, và Python **thực hiện/thực thi** (execute). Thử **lệnh** (statement): `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.

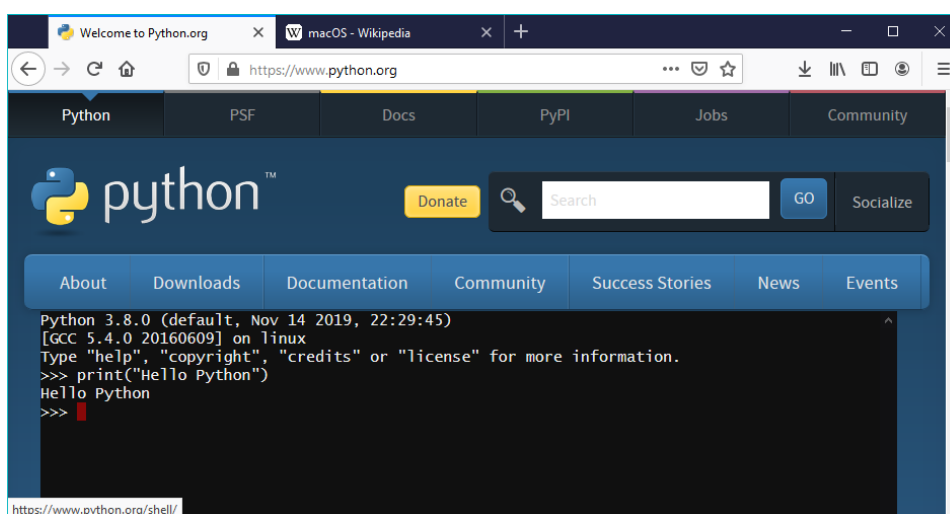


Chúc mừng!!! Bạn đã cài đặt thành công Python và thực hiện suôn sẻ lệnh đầu tiên trên IDLE. Thuật ngữ kỹ thuật gọi IDLE là một **môi trường phát triển tích hợp** (Integrated Development Environment, viết tắt IDE) cho Python, nghĩa là một tập các công cụ giúp bạn làm việc với Python. Có nhiều IDE như 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
```

Các số nhỏ ngoài lề chỉ số dòng của lệnh, bạn không gõ chúng cũng như không gõ dấu đội 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), giờ ta 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 thẩm 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, ...). Với mục đích mô tả các yêu cầu, nó hay và tốt hơn ngôn ngữ của ta nhiều. Bạn coi

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.

lại đi, câu “Xuất ra dòng chữ "Hello Python" cái coi!” 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 \leftrightarrow 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” 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, ta 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:

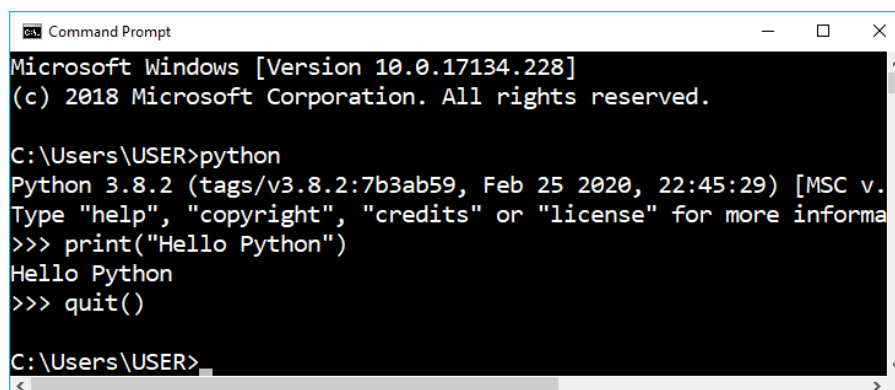
```
1 >>> print("\n is the escape sequence of newline")  
  
   is the escape sequence of newline  
2 >>> print(r"\n is the escape sequence of newline")  
\n is the escape sequence of newline
```

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 Settings.)
- 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! 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 lối 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 Điều quan trọng cần học trong cách làm Python trên 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!*

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

³Cũng trong SGK Toán 8 Tập 1.

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

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:

```
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

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).

viết ở dạng $-x$, chỉ có 1 toán hạng.⁷ Như thông thường, toán tử này giúp thực hiện **phép đổi** (negation), tức là “đổi dấu” giá trị số. Lưu ý, toán tử đổi 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	đổi (-)	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
```

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.

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.

```
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 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. Kết quả không như ta mong đợi vì Python làm việc với số thực “khá tệ” như ta đã thấy.¹¹ 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 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:

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.

¹¹Ta sẽ học cách giải quyết vấn đề này trong bài sau.

`<Name> = <Expr>`

Khi thực thi lệnh này, Python lượng giá biểu thức `<Expr>` để được giá trị `<Value>`, tạo **tên** (name)¹² `<Name>` và đặt tên đó kí hiệu cho, còn gọi là **tham chiếu** (refer) 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”.¹³

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).¹⁴ 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
```

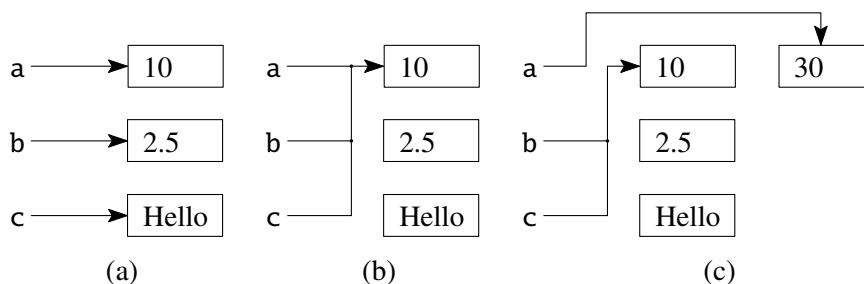
Dòng đầu tiên gồm 3 lệnh (phân cách bởi dấu ; như ta đã biết). Lệnh gán đầu tiên định nghĩa tên (hay biến) a là số nguyên 10, được mô tả bởi **hằng số nguyên**

¹²Thuật ngữ kĩ thuật là **định danh** (identifier).

¹³Dấu = 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. Kì hơn, trong Toán, người ta thường dùng dấu := hoặc $\stackrel{\text{def}}{=}$ với ý nghĩa là “kí hiệu cho” hay “được định nghĩa là”.

¹⁴Rõ 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!:

(integer literal) 10.¹⁵ 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** 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 3.¹⁶ 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 2³² như sau:

```
1 >>> 2 ** 32
4294967296
```

¹⁵Bạn cần nhận ra khác biệt tinh tế chỗ này, đây 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.

¹⁶Bạn 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 10.6.)

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 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
```

Ở Dòng 1, vì không viết liền 2 kí tự $*$ nên ta bị lỗi cú pháp (các toán tử $//$, $==$, $!=$, $<=$, $>=$ cũng vậy). IDLE bôi màu chỗ sai, cụ thể là kí 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).¹⁸

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”.¹⁹

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.²⁰

Nếu Python chấp nhận thực thi lệnh nhưng bị lỗi khi thực thi thì ta có loại lỗi thứ hai, **lỗi thực thi** (runtime error). Một lỗi hay gặp như vậy là “**lỗi chia cho 0**” (`ZeroDivisionError`) như minh họa sau.²¹

```
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.²²

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)
```

¹⁸Mô 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.

¹⁹Điều 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.

²⁰Có 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.

²¹Phần đầu của thông báo (Traceback ...) hơi lằng nhằng mà ta sẽ tìm hiểu sau.

²²Bạn 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.

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

Ta đã dùng biến name để chứa tên của một người. Lệnh xuất ở Dòng 2 dùng đúng tên biến name 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 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.²³ 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”. 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 đàng (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

²³Python không phải thần thánh!

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)`.²⁴

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 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 ký 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 ký tự trắng (tương ứng với phím Space) trước và sau toán tử `+` và không nên có ký 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.²⁵ 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 ký 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_ây`, ... 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
```

²⁴Ta sẽ hiểu hơn về cặp số và “đối số” sau.

²⁵Suy 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.

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 *không dùng tiếng Anh càng nhiều càng tốt*. Những qui ước này cũng là một phần quan trọng của phong cách viết.

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,²⁶ tính nhanh:²⁷

- (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:²⁸

- (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:²⁹

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:³⁰ (a) $n = 10$. (b) $n = 1000$. (c) $n = 1000000$.³¹ (d) $n = 10000000$.³² (e) Mọi số tự nhiên n .³³

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

²⁶Tất cả các bài tập trong bài học này đều dùng Python để làm!

²⁷SGK 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).

²⁸SGK Toán 8 Tập 1.

²⁹Chỉnh sửa từ SGK Toán 8 Tập 1.

³⁰Chỉnh 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.

³¹Bạn đợi Python xúu nhé!

³²Bạn đợi Python nổi không? Nếu không thì xem cách “ngắt” thực thi Python ở Bài tập 2.8.

³³Làm Toán câu này nhé. Làm Python được không ta? Bài tập này cho thấy sự “vi diệu” của Toán!

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 `-=`, `*=`, `/=`, `//=`, `%=`, `**=`.

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.³⁴ 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).

³⁴Trường 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!)

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:

```

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 Đọc 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 (có thể chỉnh sửa lệnh trước khi nhấn Enter). Ta cũng có thể dừng lại một lệnh trước đó bằng cách nhấp chuột vào dòng đó và nhấn Enter. Trong Command Prompt, ta dùng phím mũi tên lên/xuống 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 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 như 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 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 của Python khác biệt so với hàm số của Toán, chúng linh hoạt hơn và thực tế hơn.*

Để dễ hình dung về các hàm (trong Python), ta xem chúng như là các **dịch vụ** (service) mà trường hợp hay gặp là các dịch vụ tính toán (có thể xem là hàm số). Python cung cấp sẵn hầu hết các dịch vụ thiết yếu mà ta có thể dùng khi cần. Làm sao 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 cần biết dịch vụ đó yêu cầu những gì, tức là hàm yêu cầu 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`) yêu cầu một giá trị số (để nó tính trị tuyệt đối), dịch vụ cứu hỏa (truy cập bằng số 114) yêu cầu địa điểm bị cháy, ...

Cú pháp gọi hàm trong Python là:

<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 “vị trí” 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:

<Name>=<Expr>

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 `<Expr>` là một biểu thức mà kết quả lượng giá của nó sẽ được cung cấp cho đối số tương ứng có tên là `<Name>`. Cũng lưu ý là PEP 8 khuyến cáo không 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 dùng khoảng trắng trước và sau.

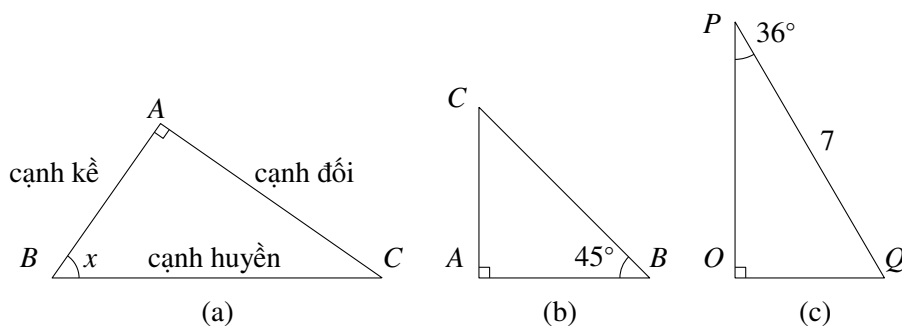
⁵Còn nhiều hàm dựng sẵn quan trọng nữa mà ta sẽ biết sau.

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. 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! Bạn sẽ gặp các module khác trong thư viện chuẩn (và các thư viện ngoài nổi tiếng khác) và học cách tự viết lấy hàm, module, thư viện trong các bài sau.

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?

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,¹² 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 các 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:¹³

```
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
```

- 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).

¹²Danh 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).

¹³Tôi 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 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 biến mà sau đó có thể được dùng lại. Ta cũng chỉ cần nạp module một lần và dùng sau đó. Ta cũng có thể gọi các hàm dựng sẵn bằng tên hàm. Rõ ràng, Python phải có cách nào đó để ghi nhớ những thứ này. Cũng tự nhiên khi Python dùng “**bộ nhớ**” (memory) của nó để nhớ.¹⁴ Ta sẽ tìm hiểu khái niệm cực kì quan trọng này sau (xem Phần 10.6), ở đây, ta “nghĩa” sơ nó thôi.¹⁵

```

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,¹⁶ trong đó có `__builtins__` chứa các hàm dựng sẵn.¹⁷ 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)¹⁸, Python đã biết các hàm dựng sẵn (và các thứ dựng sẵn khác như `True`, `False`, ...). 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 Python đụng (dùng) đến các tên thì nó sẽ lỗi 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

¹⁴Bộ nhớ là cốt lõi của mọi hệ “có trạng thái”, không nhớ thì không hoạt động được.

¹⁵Bạn cần chạy minh họa này từ đầu hoặc tắt và mở lại IDLE rồi chạy.

¹⁶Python 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 (`_`).

¹⁷Bạn có thể hình dung `__builtins__` là module chứa các hàm dựng sẵn (như `print`, `abs`, ...) mà Python tự động nạp khi bắt đầu làm việc.

¹⁸IDLE chỉ là một “vỏ bọc”, cái quan trọng là động cơ Python.

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)¹⁹ 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 một phiên làm việc, 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.²⁰ 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. Ta 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.

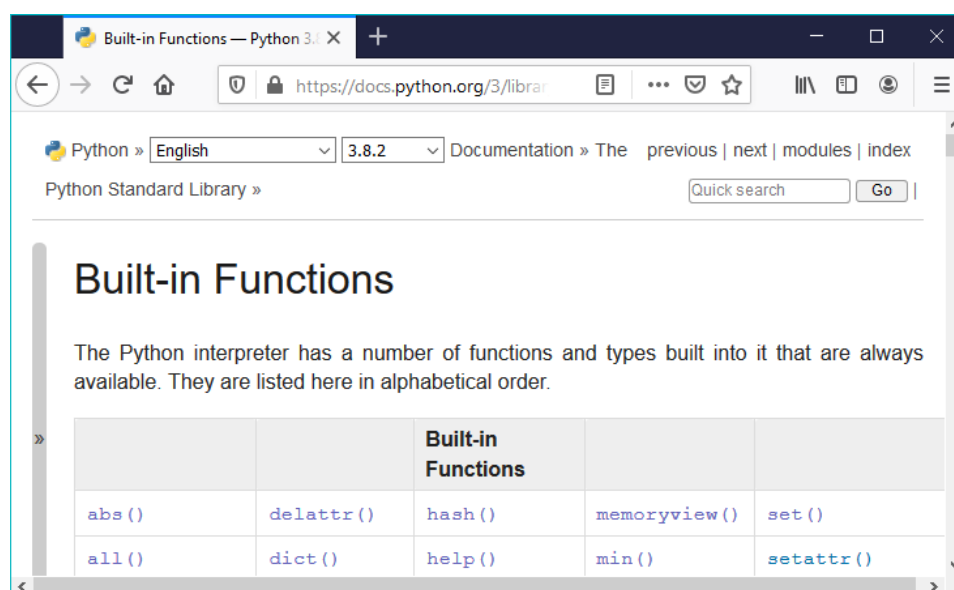
Ta có thể tra cứu bằng hệ thống tra cứu nội tại của Python với hàm dựng sẵn `help`. Chẳng hạn, ta đã dùng `help("keywords")` để tra danh sách các từ khóa, ta cũng có thể dùng `help()` để được hướng dẫn sử dụng hàm `help`, hay

¹⁹Chức 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.

²⁰Thuật 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.

`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.²¹ Chẳng hạn, để tra cứu các hàm dựng sẵn, ta 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, ta 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 kỹ 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 kỹ thuật mà bạn chưa quen thuộc.²² Tuy nhiên, suy cho cùng, *nghệ thuật tra cứu* là

²¹Hay “Python”, “Python 3”, “Python 3 docs”, ... để định hướng việc tìm kiếm.

²²Để 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”.²³ 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.

²³Tôi 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,²⁴ chứng minh các đẳng thức sau:²⁵

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

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

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
```

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:²⁶

- Tính: $\frac{\sin 25^\circ}{\cos 65^\circ}, \tan 58^\circ - \cot 32^\circ$
- Tính các tỉ số lượng giác sau (làm tròn đến chữ số thập phân thứ 4): $\sin 70^\circ 15', \cos 25^\circ 32', \tan 43^\circ 10', \cot 32^\circ 15'$
- Tìm góc nhọn x (làm tròn kết quả đến độ) biết rằng: $\sin x = 0,3495; \cos x = 0,5427; \tan x = 1,5142; \cot x = 3,163$

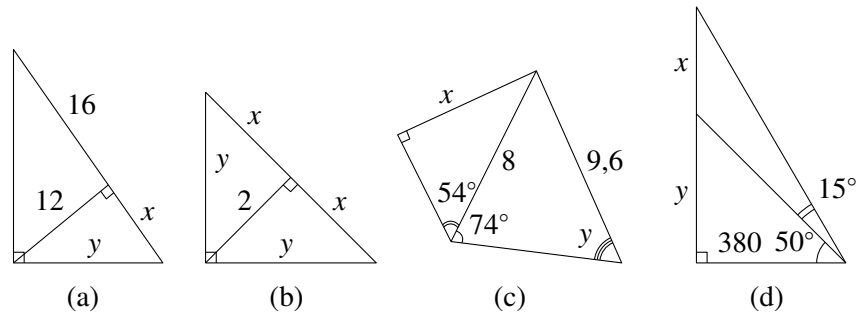
3.3 Tìm x và y trong mỗi hình sau:²⁷

²⁴Tất cả các bài tập trong bài học này đều dùng Python để làm!

²⁵SGK Toán 9 Tập 1. Tôi viết lại như cách SGK đã viết nhưng bạn cần phải dùng các kí hiệu phù hợp trong Python.

²⁶SGK Toán 9 Tập 1.

²⁷Chỉnh 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.²⁸ 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`.

²⁸ 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 ,²⁹ ϕ .³⁰
- (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. Có một cách “làm tròn” khác là dùng phân số để xấp xỉ. Hàm `Fraction` trong module `fractions` giúp tìm phân số xấp xỉ này.³¹ Hơn nữa, hàm dựng sẵn `float` giúp tìm số thực (gần đúng) tương ứng với một phân số.³² Thử minh họa sau:³³

```
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`.³⁴

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-

²⁹Số 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).

³⁰Số 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 $\frac{1+\sqrt{5}}{2}$.

³¹Đến 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!!!

³²Tra liền nè!

³³Một số calculator cũng có chức năng “chuyển đổi” dạng thập phân và phân số.

³⁴Hiển 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.³⁵ 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)
```

³⁵Bạn 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:³⁷

- (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).³⁸
- 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`.

³⁷ SGK Toán 7 Tập 1.

³⁸ Bạn không phải nhớ nhưng nghĩa sơ cho biết.

Bài 4

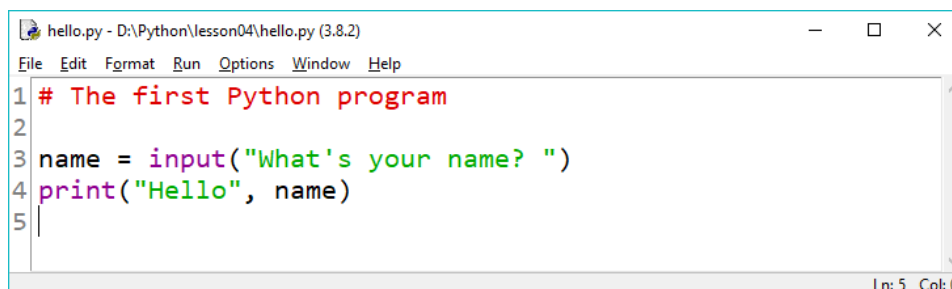
Bắt đầu lập trình

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 khi cần. Ta thường gọi tập tin này là **tập tin mã nguồn Python** (Python source code file), dãy 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 Settings, ta bật lựa chọn “Show line numbers in new windows”.

Ta cũng có thể dùng Python trực tuyến để tạo, lưu và chạy 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/blob/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!*

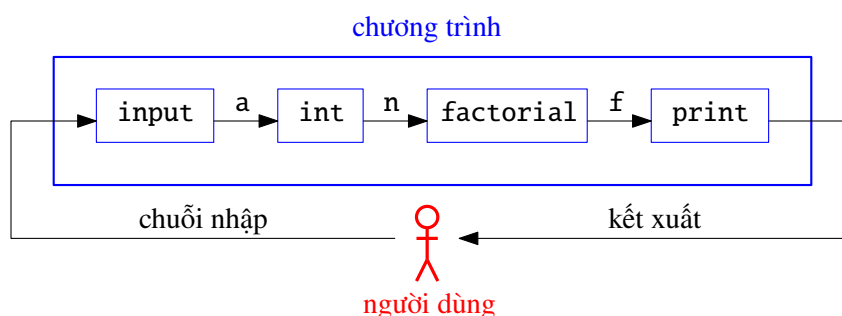
Ứng dụng khuôn chương trình trên, ta viết chương trình Python cho người dùng nhập một số nguyên (không âm), tính và xuất ra giai thừa của số nguyên đó như sau:

<https://github.com/vqhBook/python/blob/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/blob/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 10.

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 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/blob/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/blob/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/blob/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/blob/master/lesson04/hello3.py> _____

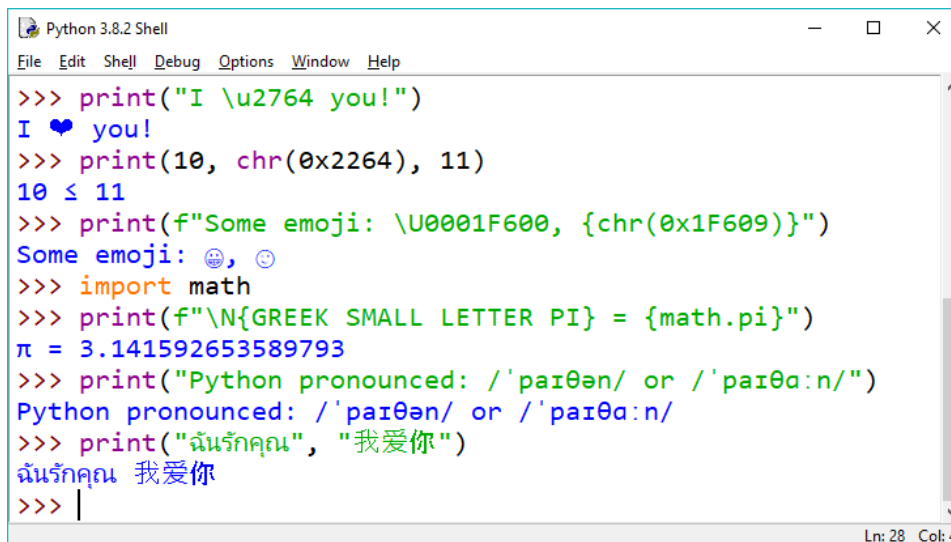
```

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
>>> |

```

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

```

\_____ \____.____./ ____ | | ____ \_____ \_____
|      ____<  |  | \   ____ \  |  \ /  _ \ /  \
|      |      \____ | | | |  Y  (  <>  )  |  \
|_____|      /  ____| | | | ____| /\_____/|____| /
          \      \      \      \

```

4.4 Viết chương trình cho nhập `<name>` rồi xuất ra bài hát “Happy birthday”:

```

Happy birthday to <name>.
Happy birthday to <name>.
Happy birthday, happy birthday, happy birthday to <name>.

```

4.5 Viết chương trình cho nhập tên `<Roméo>` và `<Juliet>` rồi xuất ra đoạn trích.²⁰

```

"... <Roméo> bước thơ thần trong vườn, tơ tưởng đến
<Juliette>. Bỗng nhiên, một cánh cửa sổ từ từ hé mở,
<Juliette> hiện ra, tựa vào bao lơn. Cô cũng bồn chồn và lo
lắng, rồi trong khi <Roméo>, ẩn mình trong bóng tối, so
sánh nàng với bình minh và khung cửa với phương Đông, chế
nhạo mặt trăng mờ nhạt vì hờn ghen với nhan sắc kiều diễm
của <Juliette> ..."

```

4.6 Viết chương trình cho nhập vào năm sinh của một người và xuất ra tuổi của người đó.

Lấy thời gian trên máy. Để chương trình vẫn chạy đúng cho tương lai (hay khi du hành về quá khứ), ta có thể lấy về năm hiện tại (trên máy) bằng cách dùng các hàm và giá trị phù hợp trong module `time`. Thử minh họa sau:²¹

```

1 >>> import time
2 >>> print(time.localtime().tm_year)
2020

```

Viết lại chương trình trên với kĩ thuật mới này.

4.7 Viết chương trình cho nhập số nguyên dương có không quá 3 chữ số. Xuất ra tổng các chữ số của số đó. Ví dụ: nhập số 123, xuất ra 6. (Vì $1 + 2 + 3 = 6$).

¹⁹Việc dùng các kí tự để tạo nên một tác phẩm đồ họa được gọi là **nghệ thuật ASCII** (ASCII art). Tôi đã dùng trang <http://patorjk.com/software/taag/#p=display&f=Graffiti&t=Python> để tạo nên chữ Python cách điệu như mẫu. Tuy nhiên, bạn hãy sáng tạo thêm các mẫu khác và tạo logo cho mình.

²⁰Đoạn trích này được “lượm lặt” đâu đó trên mạng và có thể không liên quan gì đến bản gốc.)

²¹Nhớ rằng bạn luôn luôn có thể tra cứu, tuy nhiên, giờ bạn chưa đủ công lực để hiểu hết. *Bắt chước, thử nghiệm và tra cứu là chìa khóa để bạn tiến triển lúc đầu.*

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.

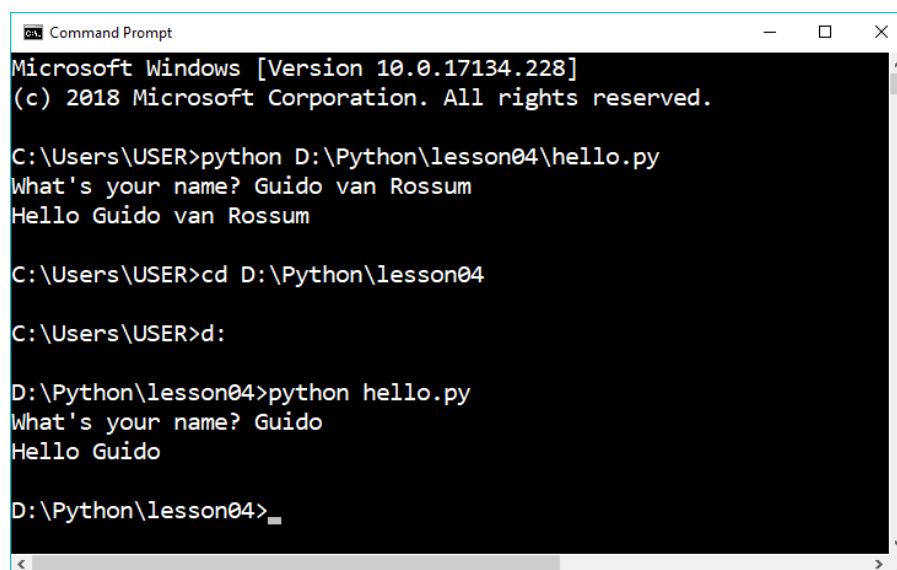
²³Python cũng hỗ trợ tốt số bát phân (cơ số 8) nhưng ít được dùng.

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.

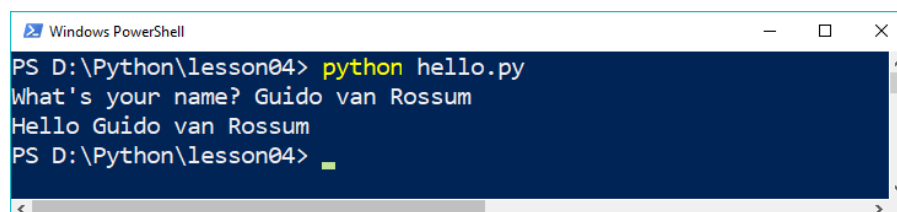
C:\Users\USER>python D:\Python\lesson04\hello.py
What's your name? Guido van Rossum
Hello Guido van Rossum

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:

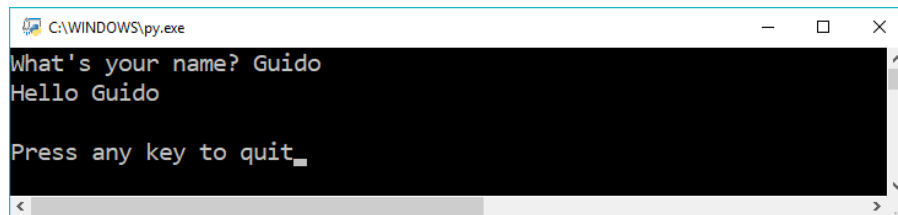
— <https://github.com/vqhBook/python/blob/master/lesson04/hello4.py> —

```

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 một 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")
```

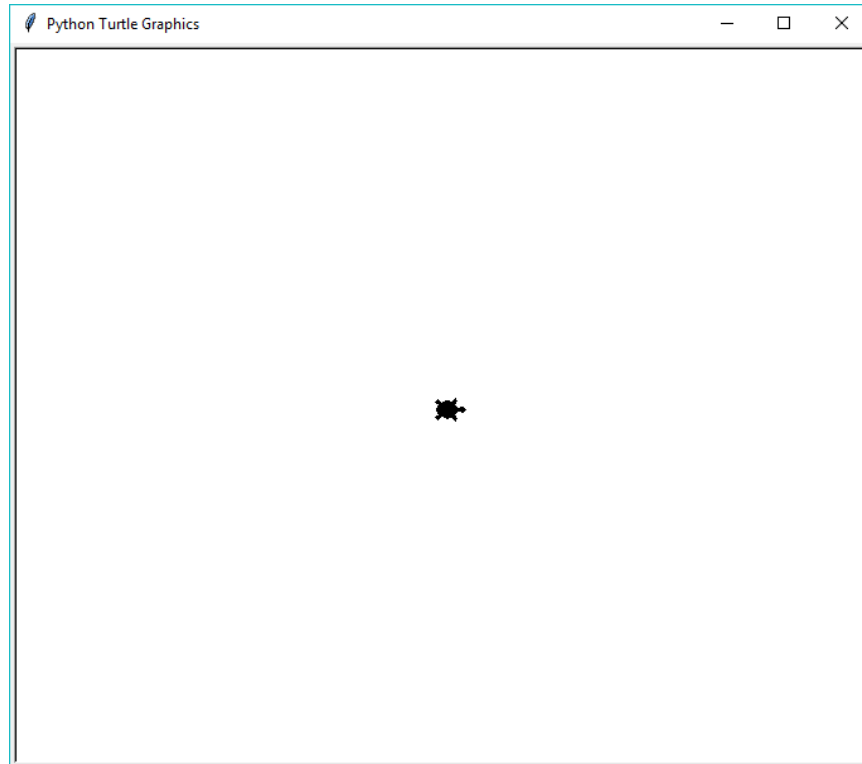
Lệnh đầu tiên nạp module `turtle` nhưng đặt lại tên module là `t` bằng từ khóa `as`, mà mục đích là để gõ ít phím hơn. Ta cũng đã dùng dạng `import` này ở Bài tập 3.10 nhưng không nên lạm dụng.¹ Lệnh thứ hai gọi một hàm của module `turtle` (mà bây giờ được viết gọn là `t`) là `showturtle`.² Như tên gọi, 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.

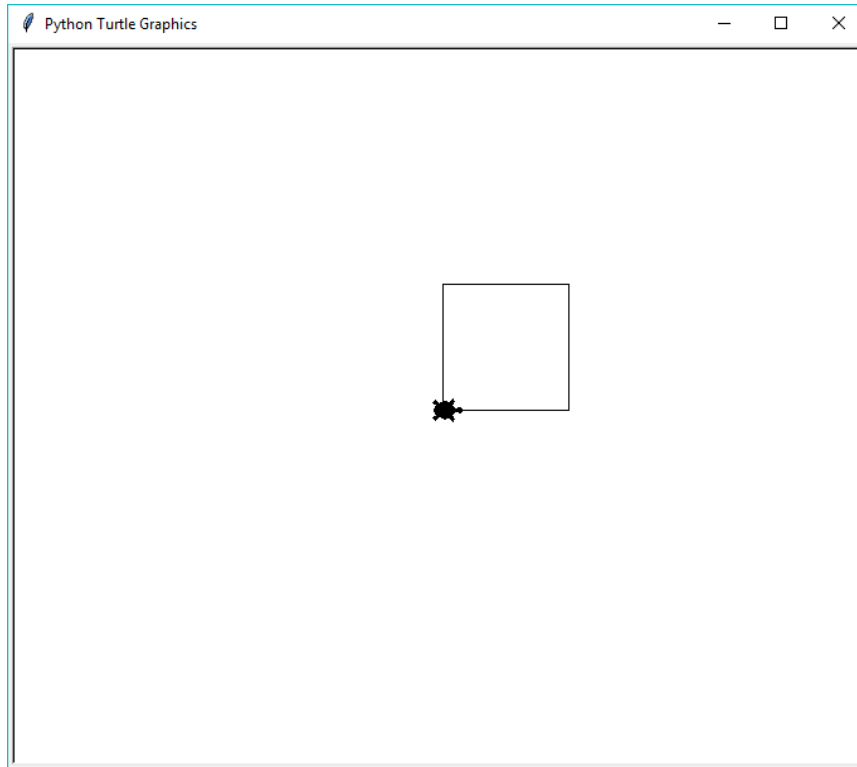
Tiếp theo, ta ra lệnh cho con rùa tiến tới 100 “bước” rồi quay trái 90° thêm 3 lần như vậy nữa để hoàn thành hình vuông cạnh 100 bằng các lệnh sau (mà thực chất là lặp lại 3 lần cặp lệnh trên):⁵

```
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 (search “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)
```

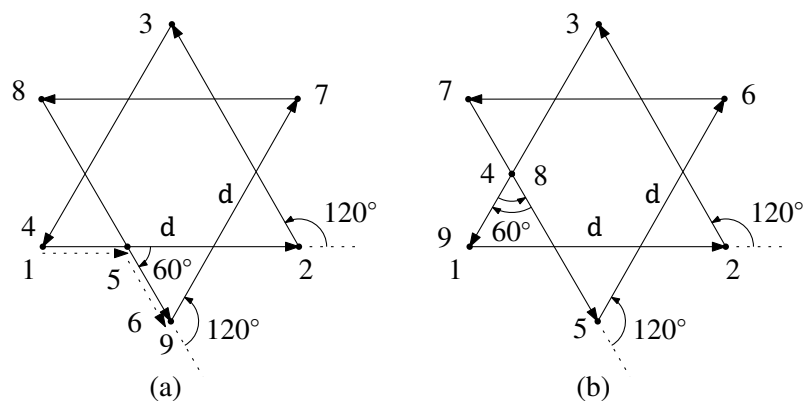
Ta cũng đã cho người dùng nhập kích thước hình vuông (độ dài cạnh). Thực thi chương trình này, ta có kết quả là hình vuông với kích thước như người dùng nhập. Cũng lưu ý, trong chế độ chương trình ta không nên viết nhiều lệnh trên một dòng, tuy nhiên với các lệnh ngắn và nhất là khi việc “gom lại” có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

<https://github.com/vqhBook/python/blob/master/lesson05/star.py>

```

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

<https://github.com/vqhBook/python/blob/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 cõn tên lửa với đối số `"fastest"`.⁸

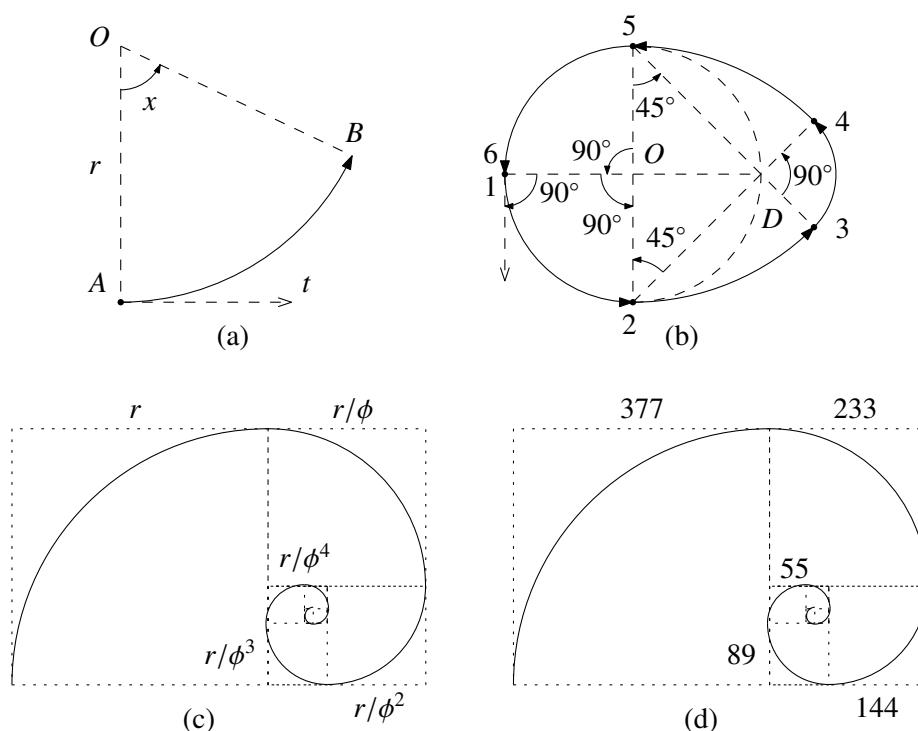
5.3 Bắt con rùa bò nhiều hơn nữa

Ta sẽ vẽ những hình phức tạp hơn trong phần này. Muốn vậy, ta phải có chiến lược phù hợp để bắt con rùa “bò” và “quay” theo hình. 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**

⁷ Có phải hình nào cũng vẽ một nét được không? Nếu có thì cách vẽ thế nào? Thú vị thay, Toán học lại cho câu trả lời đơn giản và rõ ràng. (Euler đã trả lời câu hỏi này với lý thuyết **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.

tròn (arc) với hàm `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/blob/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 $\widehat{1\ 2}$ có bán kính r (tâm O , góc ở tâm 90°), sau đó ta bảo con rùa bò theo cung $\widehat{2\ 3}$ 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/blob/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/blob/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/blob/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 5.9.

https://github.com/vqhBook/python/blob/master/lesson05/animation_off.py

```

1 import turtle as t
2
3 x = t.numinput("Hello", "Enter radius: ")
4 y = 2**0.5 * x
5 t.hideturtle()
6 t.color("black", "yellow")
7
8 t.tracer(False)
9 t.up(); t.forward(x); t.left(135); t.down()
10 t.begin_fill()
11 t.forward(y); t.left(90); t.forward(y); t.left(90)
12 t.forward(y); t.left(90); t.forward(y); t.left(90)
13 t.end_fill()
14 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.

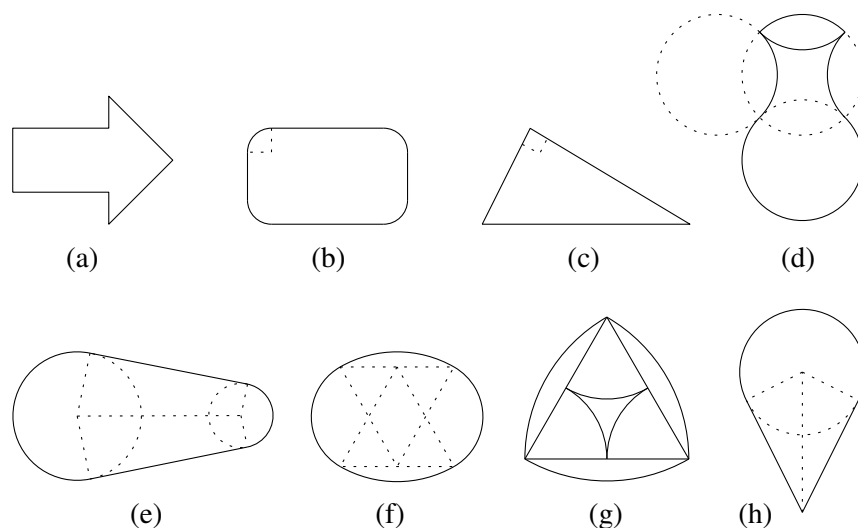
¹⁷Tra cứu <https://docs.python.org/3/library/turtle.html#module-turtle>.

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.

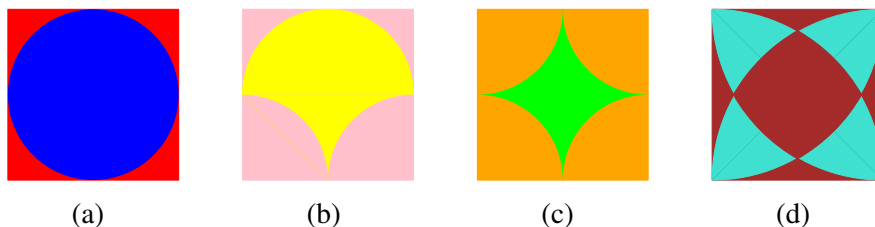
5.2 Vẽ bằng một nét các Hình (d) và (g) ở Bài tập 5.1.

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

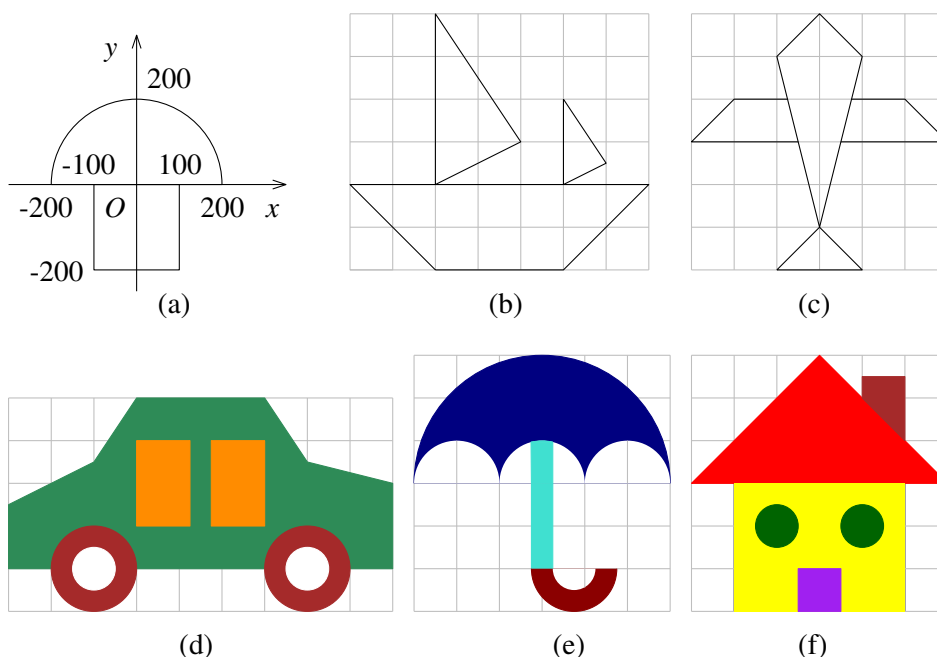
Ngoài tên màu, Turtle cho phép mô tả màu RGB bằng chuỗi **mã màu** (color code) có dạng #RRGGBB trong đó RR, GG, BB là 2 kí số cơ số 16 mô tả cường độ thành phần đỏ, xanh lục, xanh dương tương ứng (giá trị nguyên trong phạm vi 0-255). Chẳng hạn, mã màu đỏ là #FF0000, màu trắng là #FFFFFF, màu đen là #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 chương trình tại <https://github.com/vqhBook/python/blob/master/lesson05/color.py> để rõ hơn.

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



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 tại <https://github.com/vqhBook/python/blob/master/lesson05/mushroom.py> minh họa cách vẽ Hình (a) ở dưới theo “chế độ tuyệt đối”.

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



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

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.

nhau cho 2 trục. Chương trình tại <https://github.com/vqhBook/python/blob/master/lesson05/coordinates.py> 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” của 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.

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

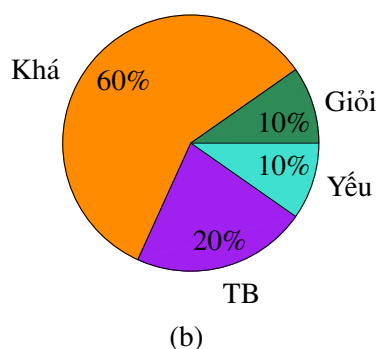
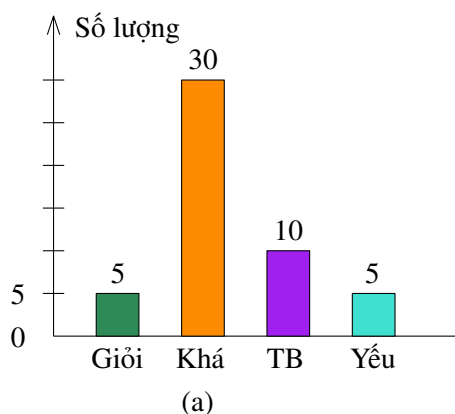
Sau khi di chuyển con rùa đến nơi cần xuất và chọn màu phù hợp, ta có thể xuất chuỗi bằng hàm `write`. Đối số quan trọng dĩ nhiên là chuỗi cần xuất, ngoài ra, đối số có tên `align` xác định cách đóng hàng (`left` – trái (mặc định), `right` – phải, `center` – giữa). Ta cũng có thể xác định font chữ (là 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 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 chart) ở Hình (b) dưới.



Dùng Python vẽ 2 biểu đồ trên.

5.8 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 5.7.²²

5.9 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?!

5.10 Vẽ logo Python như hình ở https://www.python.org/static/community_logos/python-logo-master-v3-TM.png.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 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.

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

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 5.7 và làm “tự động” bằng chương trình. “Tự động” chính là điều mà mọi chương trình hướng đến.

²³Không 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.

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:

<https://github.com/vqhBook/python/blob/master/lesson06/exam.py>

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

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

_____ <https://github.com/vqhBook/python/blob/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/v/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 nguyên 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 ngửa và 0 là sấp.

Sau khi cho người dùng chọn mặt với chuỗi "N" là ngửa và "S" là sấp, 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 (ngửa) hay 0 (sấp). 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/blob/master/lesson06/coin_toss2.py —

```

1  import random
2
3  coin = random.randint(0, 1)
4  choice = input("Bạn chọn ngửa hay sấp(N/S)? ")
5  print("Đồng xu ra " + ("ngửa" if coin == 1 else "sấp"))
6  print("Bạn chọn " + ("ngửa" if choice == "N" else "sấp"))
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.

https://github.com/vqhBook/python/blob/master/lesson06/turtle_escape.py

```

1 import turtle as t
2 import random
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 kỹ 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/blob/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 “tốn 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** (framework) hay **dàn ý** (outline) chương trình. Trong trường hợp này, pass 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.

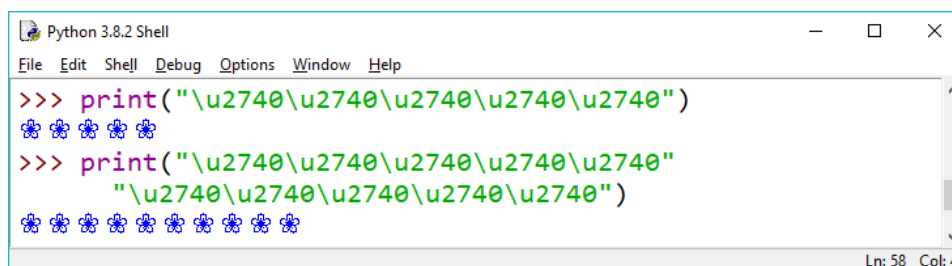
Bài 7

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” ngàn 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); <Name> là 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/blob/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/blob/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)
```

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 lập (<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 lập (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 (thân lập) có thể phức tạp hơn (không chỉ 1 lệnh mà là khối gồm nhiều lệnh tùy ý). Chẳng hạn, với thân lập phức tạp hơn, ta có chương trình đếm khá thú vị như sau:

— <https://github.com/vqhBook/python/blob/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 (đổi 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? 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, ta 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/blob/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 có đặc điểm chung là trông rất phức tạp nhưng lại có “kiến trúc” rất đơn giản: chúng chỉ là sự *lặp lại nhiều lần một vài bước đơn giản 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). Tương tự, bạn 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:

```

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

Bạn tra cứu thêm để “rõ sơ”. Ta sẽ tìm hiểu kĩ hơn hàm `range` và lệnh `for` 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 ... quảng gánh lo đi và vui sống:) Giỡn thôi, luyện lại từ đầu, một ngày không được thì ba bốn ngày!

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

7.3 Lập linh hoạt với lệnh while

Ta có thể dùng lệnh `for` vẽ “đường xoắn ốc vàng” rất gọn nhưng không dễ vẽ “đường xoắn ốc Fibonacci”. Lí do là bán kính đường tròn của mỗi lần lặp thay đổi khó mô tả theo biến lặp. Ta có thể dùng cấu trúc lập linh động hơn, **lệnh while** (while statement), vẽ đường xoắn ốc Fibonacci như sau:

```

1 https://github.com/vqhBook/python/blob/master/lesson07/Fibonacci\_spiral.py
2 import turtle as t
3
4 x, y = 0, 1
5 while x <= 377:
6     t.circle(x, 90)
7     x, y = y, x + y
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 gán cặp giá trị 0, 1 cho cặp biến `x, y` mà có nghĩa là lần lượt gán 0 cho `x`, 1 cho `y` (đây 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 “cấp 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, 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.

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/blob/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.

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:

⁶SGK Toán 9 Tập 2.

⁷Bạn sẽ thấy nhiều trường hợp Python “giải cứu” Toán trong các bài sau.

```

https://github.com/vqhBook/python/blob/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 “*về 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/blob/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("*", 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

Ta thấy rõ cấu trúc 2 vòng lặp lồng nhau trong ý tưởng trên. Vòng lặp ngoài lặp qua các dòng (i là dòng thứ) và vòng lặp trong xuất 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:)).

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.

Ta cũng có thể khử lồng để được chương trình viết đơn giản hơn như sau:

<https://github.com/vqhBook/python/blob/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/blob/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, chương trình tìm ra các số 16, 25, 36, 49, 64, 81, 100. Để kiểm tra một số có là số chính phương, ta dùng mẹo lấy phần nguyên của căn và bình phương lên xem có bằng số đó hay không. Ta cũng có thể kiểm tra đơn giản hơn bằng cách dùng điều kiện sau cho lệnh if ở Dòng 6: `(i**0.5).is_integer()`.⁹

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 bỏ qua (không thực thi) các lệnh bên dưới continue 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.

Lệnh đơn khác, lệnh break, rất giống lệnh continue chỉ trừ việc: khi gặp lệnh này, Python 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

⁹Ta sẽ học cách dùng “phương thức” (is_integer) ở Bài 10.

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` tạo 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:

https://github.com/vqhBook/python/blob/master/lesson07/perfect_square2.py

```

1 a, b = 10, 100
2 for i in range(a, b + 1):
3     if i % 2 == 0:
4         continue
5
6     if int(i**0.5)**2 == i: # i là số chính phương
7         print(i)
8         break
9
10 else:
11     print("Không tìm thấy số nào!")
12
13 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 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 (hay một số cách khác ta sẽ biết sau), Python không thực thi phần `else`. Trong chương trình trên, vòng lặp `for` 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 điều kiện của `if` ở Dòng 6; khi đó, phần `else` 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` được thực thi và lệnh xuất ở Dòng 11 đưa ra thông báo “không tìm thấy số nào”.

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/blob/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.¹⁰ 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 ta đóng cửa sổ đồng hồ (của sổ `turtle`) thì vòng lặp trên sẽ kết thúc.¹¹ Sâu xa hơn, nếu thời gian “dừng lại” thì lời gọi hàm `localtime` sẽ bị 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 nên biết rằng: *không có vô hạn trong thực tế, vô hạn chỉ có trong tưởng tượng mà thôi!*

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 bỏ. Thường thì việc xác định trước sẽ rõ ràng hơn nhưng đánh giá muộn cũng phù

¹⁰Tương 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 10.8 và 12.1.

¹¹Khi cửa sổ `turtle` bị đóng, lệnh `update()` ở Dòng 10 bị lỗi thực thi (do không còn cửa sổ để `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.

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 như 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: $n = \frac{n}{2}$,
- 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 **nghị vấn Collatz** (Collatz conjecture) hay **bài toán $3n + 1$** ($3n + 1$ problem) mà chưa có ai trả lời được.¹²

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 kĩ để thành thạo cách viết các lệnh lặp trong Python.

```

1 while text := input("Nhập số nguyên dương n: "):
2     n = int(text)
3     num = 0
4     print(n, end=" ")
5     while n != 1:

```

¹²Bạn có thể đọc thêm về bài toán này ở https://en.wikipedia.org/wiki/Collatz_conjecture.

```

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 phép ta thử nghiệm 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 nhấn 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.¹³ 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 kĩ thuật lặp.

https://github.com/vqhBook/python/blob/master/lesson07/turtle_draw.py

```

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

```

¹³Thật 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.

```

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 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.¹⁴ 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.

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

¹⁴Đây 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.

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. 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 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 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/blob/master/lesson07/factorial.py
2  n = int(input("Nhập số nguyên dương: "))
3  f = 1
4  for i in range(1, n + 1):
5      f *= i
6  print(f"{n}! = {f}")

```

hoặc bằng lệnh lặp `while`.¹⁵

¹⁵Dĩ nhiên, ta không cần “vắt vớ” như vậy để tính giai thừa. Ta chỉ cần nạp module `math` và gọi `math.factorial(n)` hoặc `math.prod(range(1, n+1))` để tính $n!$.

<https://github.com/vqhBook/python/blob/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.¹⁶

- (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) $S(n) = 1 + 2 + \dots + (n-1) + n$.¹⁷
- (d) $S(n) = 1^2 + 2^2 + \dots + (n-1)^2 + n^2$.¹⁸
- (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).¹⁹
- (f) $S(x, n) = 1 + x + \frac{x^2}{2!} + \dots + \frac{x^{n-1}}{(n-1)!} + \frac{x^n}{n!}$.²⁰
- (g) $\frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}}}$.²¹
- (h) $\frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \dots}}}}$.²²

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 (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:

¹⁶Nhớ 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.

¹⁷Bạn có thể đối chiếu kết quả với $\frac{n(n+1)}{2}$.

¹⁸Bạn có thể đối chiếu kết quả với $\frac{n(n+1)(2n+1)}{6}$.

¹⁹Kí 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 $\frac{1-x^{n+1}}{1-x}$.

²⁰Khi 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)`.

²¹Khi số lần lặp đủ lớn thì giá trị này xấp xỉ tỉ số vàng $\phi = \frac{1+\sqrt{5}}{2}$.

²²Khi số lần lặp đủ lớn thì giá trị này xấp xỉ \sqrt{x} . Như vậy, đây là một phương pháp khai phương khác.

- (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. Thiết kế thuật toán (tức là cánh vẽ hay chiến lược vẽ) và cài đặt chương trình vẽ hình ngôi sao:

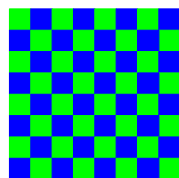
- (a) 6 cánh.
- (b) 10 cánh.
- (c) Có số lượng cánh do người dùng nhập (dĩ nhiên là từ 3 cánh trở lên).

7.6 “Vẽ” bằng kí tự. Viết chương trình “vẽ” bằng kí tự các hình sau:²³

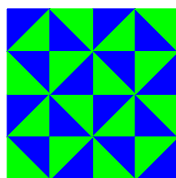
*	*****	*	*****
**	*****	***	**** *
***	*****	*****	*** **
****	*****	*****	** **
*****	*****	*****	* *
*****	*****	*****	** **
*****	*****	*****	*** **
*****	*****	***	**** **
*****	*****	*	*****

²³Chương 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).

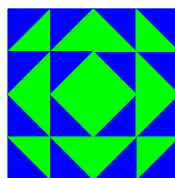
7.7 Viết chương trình dùng con rùa tô màu các hình sau:



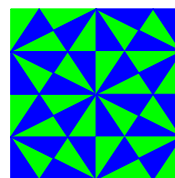
(a)



(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. 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)²⁴ là “tam giác các số” với dòng 1 gồm 1 số, dòng 2 gồm 2 số, ... Ví dụ sau là tam giác Pascal 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 cho thấy, các số của tam giác Pascal được xây dựng theo qui 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”.

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:

²⁴Đặt theo tên nhà Toán học Pháp Blaise Pascal.

```

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, hàm `int` với giá trị cho đối số 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 124.²⁵

7.12 Dùng Python giải bài toán cổ sau:²⁶

“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:

²⁵SGK Toán 9 Tập 2.

²⁶SGK Toán 9 Tập 2.

“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 4.1, ta có thể dùng **Python trực tuyến** (online Python) để lập trình Python. Ngoài việc không cần cài đặt, Python trực tuyến có lợi ích nữa là cho phép ta dùng Python 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:²⁷

- 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.

²⁷Search 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:

```
1 https://github.com/vqhBook/python/blob/master/lesson08/hello.py
2 def sayhello(name):
3     hello_string = " Chào " + name + " "
4     print("=" * (len(hello_string) + 2))
5     print("|" + hello_string + "|")
6     print("=" * (len(hello_string) + 2))
7
8 sayhello("Python")
9 sayhello("Vũ Quốc Hoàng")
10 sayhello("Guido van Rossum")
11 a_name = input("Tên của bạn là gì? ")
12 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 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 cặp

¹ Đây cũng là khuyến cáo của PEP 8.

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 sau 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):

https://github.com/vqhBook/python/blob/master/lesson08/Newton_sqrt.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 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 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:

https://github.com/vqhBook/python/blob/master/lesson08/expression_value.py

```

1 def N(x, y):
2     return 8*(x**3) - 12*(x**2)*y + 6*x*(y**2) - y**3
3
4 print(N(6, -8))
5 print(N(-8, 6))
6 print(N(x=6, y=-8))
7 print(N(y=-8, x=6))
8 print(N(x=-8, y=6))
9 print(N(6, y=-8))
10 # 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, ta cần nhớ thứ tự của các tham số (tham số thứ nhất là gì? thứ 2 là gì? ...). Chẳng hạn, nếu ta 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 $x = -8, y = 6$.

Có 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 , 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/blob/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. 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 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 (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 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/blob/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.

```

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:

- Hàm `sqrt` của module `math`. Để dùng nó ta chỉ cần truyền đối số là hàm này cho tham số `method` của hàm `cal_sqrt` (Dòng 27).
- 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`.

- Hàm `pow` của module `math` với hàm bọc `math_pow_sqrt`.
- Toán tử mũ với hàm bọc `exp_operator_sqrt`.
- 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 mình 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/blob/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}")
17
18 cal_sqrt(math.sqrt, "math's sqrt")
```

⁵Trong nhiều ngôn ngữ lập trình, 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”.

```

19 cal_sqrt(lambda x: pow(x, 0.5), "built-in pow")
20 cal_sqrt(lambda x: math.pow(x, 0.5), "math's pow")
21 cal_sqrt(lambda x: x ** 0.5, "exponentiation operator")
22 cal_sqrt(Newton_sqrt, "Newton's sqrt")

```

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, không có cặp ngoặc tròn bao <Paras> như trong định nghĩa hàm thông thường. Hơn nữa, ta để trống <Paras> nếu hàm 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 các kĩ thuật ở trên (truyền/nhận hàm như các đối tượng dữ liệu và hàm `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/blob/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 (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 dễ dàng phân biệt qua cách viết (`forward` viết không là của chương trình còn `t.forward` là của module `turtle`).

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). Sau 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 (người dùng 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.

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).

8.7 Lệnh biểu thức, hiệu ứng lề và docstring

Thử chương trình minh họa sau:

```

1  https://github.com/vqhBook/python/blob/master/lesson08/expression\_stm.py
2  # This is a comment ...
3  # ... and comment
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.
- 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 lề. 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 lề, 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 lề.

- 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 lề 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 (có thể dùng tên dựng sẵn Ellipsis thay cho ...). 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 chấm lửng ở Dòng 3. Ta sẽ thấy vài cách dùng hay khác của dấu chấm lửng trong các bài sau.

Thử một chương trình khác có lệnh biểu thức với hiệu ứng lề thú vị sau:

```
1 def Newton_sqrt(x):
2     "Tính căn theo phương pháp Newton."
3     y = x
4     for i in range(100):
5         y = (y + x/y) / 2
```

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 ...

```

5         y = y/2 + x/(2*y)
6     return y
7
8     print(f"Căn của 2 theo Newton: {Newton_sqrt(2):.9f}")
9     print(Newton_sqrt.__doc__)
10    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 sơ 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 sơ 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 lẻ “nhỏ nhỏ” là sơ liệu thì chuỗi sơ 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). Tạo file mã `sqrt.py` chứa mã sau:

```

1     https://github.com/vqhBook/python/blob/master/lesson08/sqrt.py
2     """Module hỗ trợ việc tính căn bằng phương pháp Newton.
3
4     Tác giả: Vũ Quốc Hoàng.
5     Ngày viết: 10-03-2020.
6     """
7
8     def Newton_sqrt(x):
9         """Tính căn theo phương pháp Newton.

```

⁹ Bạn không hiểu hết câu chơi chữ này nếu không biết về **Java**, một ngôn ngữ lập trình phổ biến khác.

```

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 cách nào đó báo cho Python biết “vị trí” file mã chứa module mà cách đơn giản nhất là đặt file mã module cùng thư mục với file mã chương trình dùng nó (xem Bài tập 8.9). Chẳng hạn, tạo file mã `sqrt_program.py` cùng thư mục với `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. Như mã nguồn cho thấy, 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 sơ liệu chi tiết hơn cho hàm `Newton_sqrt`. Ngoài ra, ta cũng tạo chuỗi sơ liệu cho module bằng hằng chuỗi viết ở đầu module. Để ý là trong chuỗi sơ 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

¹⁰Để tiết kiệm giấy, từ đây, tôi sẽ không dùng các chuỗi sơ liệu (cũng như ghi chú) trong các chương trình minh họa nhưng bạn nên tập thói quen viết chuỗi sơ liệu vì nó có nhiều ích lợi (tương

Cũng lưu ý là, trong Python, chương trình và module không khác biệt nhiều (xem Bài tập 8.10).

Bạn xem lại Bài tập 7.7, chẳng hạn Câu (a). Giả như có một hàm nào đó hỗ trợ ta vẽ (và tô) hình vuông với kích thước, vị trí, màu được cho; 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/blob/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*.

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 cùng thư mục với file mã chương trình):

https://github.com/vqhBook/python/blob/master/lesson08/chess_board.py

```

1  import turtle as t
```

tự và hơn ghi chú).

```

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 cùng thư mục với file mã chương trình):

https://github.com/vqhBook/python/blob/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 đối 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 kĩ thuật đăng kí 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 đổi (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 5.9c).

Gợi ý: bạn cũng có thể cho người dùng nhập tên bằng hộp thoại nhập của con rùa như Bài tập 5.9. 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 Tương tự như hàm vẽ hình vuông `square` trong module `figures`, bổ sung vào module này các hàm sau:

- (a) `line` vẽ đoạn thẳng nối 2 điểm có tọa độ được cho với màu được cho và dùng nó vẽ các hình trong Bài tập 5.4(b), (c).
- (b) `triangle` vẽ và tô tam giác (với các tham số thích hợp) và dùng nó vẽ các hình còn lại trong Bài tập 7.7.
- (c) `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) `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) `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° . Ta 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.

8.9 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 gọi là **đường dẫn hệ thống** (system path). 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 để trong các thư mục đặc biệt này.

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`. Ta cũng có thể mở file mã bằng `File` → `Open Module` và xem đường dẫn file trên đầu cửa sổ IDLE. Chẳng hạn, nhấp `File` → `Open Module`, gõ tên `turtle` trong hộp thoại hiện ra rồi nhấn `OK`, Python sẽ mở file mã `turtle.py` của module `turtle` yêu cầu.¹²

Cách “bá đạo” để dùng một module là đặt nó vào một thư mục trong đường dẫn hệ thống (chẳng hạn, chung thư mục với file mã `turtle.py`). Khi đó ta có thể dùng module như module chuẩn. Tuy nhiên, cách này khá nguy hiểm vì ta

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.

có thể lỡ ghi đè, xóa, sửa các file mã của module chuẩn. Cách an toàn và đơn giản nhất là đặt file mã module cùng thư mục với file mã chương trình như ta đã làm trong Phần 8.8.

Linh hoạt hơn, ta có thể 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` 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` (module `sqrt` ở Phần 8.8) đặt ở thư mục `D:\Python\lesson08`, 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)

    Help on module sqrt:
    ...

```

Dĩ nhiên, ta cũng thể dùng `os.chdir` trong mã chương trình. Sở dĩ việc để file mã module cùng thư mục với file mã chương trình có tác dụng vì khi chạy chương trình, Python tự động đặt thư mục làm việc là thư mục chứa file mã chương trình. Ta cũng có thể xem thư mục làm việc 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.

8.10 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 hai đều được viết trong một file mã Python và được thực thi theo cùng một 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 của IDLE)

¹³Thật 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`, ...

thì biến đặc biệt `__name__` sẽ được đặt giá trị là `"__main__"`, ngược lại, nếu file mã được nạp như module (chẳng hạn, bởi 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 module vừa là chương trình: nó cung cấp các hàm cho module khác dùng và nó có khối lệnh ngoài cùng (khối lệnh không thật đầu dòng) chạy như là chương trình. Các file mã dạng này thường dùng khuôn mẫu sau cho khối lệnh ngoài cùng:

```
if __name__ == "__main__":
    <Suite>
```

để chỉ chạy khối lệnh chương trình (<Suite>) khi được dùng như chương trình.

Lưu ý, 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 phù hợp với module vì việc chạy và nạp module nhiều lần rất tốn kém, nhưng 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** (main top-level file), làm **điểm vào** (entry point) của chương trình. Khi đó, chương trình được chạy từ file mã chính và mã này dùng thêm các module khác để hoàn thành chương trình. Python cũng hỗ trợ việc tổ chức nhiều module liên quan bằng **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.

Bài 9

Case study 2: Sức mạnh của lặp

Ta đã học các cấu trúc lặp trong Bài 7. Có thể nói, khả năng làm lặp lại nhiều lần các việc đơn giản là nền tảng sức mạnh của máy tính. Bài nghiên cứu này cho thấy rõ bản chất của lặp và sức mạnh của nó qua vài ứng dụng. *Lặp, thực sự, là phương pháp luận, là công cụ quan trọng của việc giải quyết vấn đề trên máy tính nên bạn cần nắm rõ nó.* Khối lượng kiến thức của bài này khá nhiều. Nếu cảm thấy phần nào đó quá khó, bạn có thể tạm thời bỏ qua và trở lại nghiên cứu sau.

9.1 Sức mạnh thực sự của lặp

Bạn hãy xem lại bài toán tính tổng các chữ số của một số nguyên dương trong Bài tập 4.7 và Bài tập 7.3a-b. Yêu cầu của các bài tập này khá đơn giản vì số chữ số là cố định hoặc được giới hạn. Trong các trường hợp như vậy, ta chỉ cần dùng cấu trúc **tuần tự** (sequential) hay **thẳng** (linear) đủ dài. Chẳng hạn, bạn có thể đã làm như sau:

```
1 https://github.com/vqhBook/python/blob/master/lesson09/number\_digits.py
2 n = int(input("Nhập số có không quá 3 chữ số: "))
3 đơn_vị, chục, ngàn = n % 10, (n % 100) // 10, n // 100
4 tổng = đơn_vị + chục + ngàn
5 print("Tổng các chữ số là:", tổng)
```

Cách làm này rõ ràng nhưng không hay bằng cách sau:

```
1 https://github.com/vqhBook/python/blob/master/lesson09/number\_digits2.py
2 n = int(input("Nhập số có không quá 3 chữ số: "))
3 tổng = 0
4
5 tổng += n % 10; n //= 10
6 tổng += n % 10; n //= 10
7 tổng += n % 10; n //= 10
8
9 print("Tổng các chữ số là:", tổng)
```

Cách làm này trông dài hơn nhưng đó chỉ là về “lượng”; về “chất” thì rất ngắn, ta không cần biết đến khái niệm chữ số hàng chục, hàng trăm (mà sẽ rất khó mở rộng cho số nhiều chữ số); ta chỉ cần biết chữ số hàng đơn vị (tức chữ số hàng thấp nhất). Tuy nhiên, ta cần một phát hiện quan trọng là: nếu chia nguyên một số cho 10 thì ta đẩy được số sang phải 1 vị trí, tức là mọi hàng đều tụt bậc. Bằng cách này, ta đã “thu” bài toán về bài toán nhỏ hơn cùng dạng (ta sẽ nghiên cứu kỹ thuật này sau). Cài đặt trông dài về lượng nhưng ngắn về chất ở trên nhanh chóng ngắn về lượng nếu ta dùng vòng lặp như sau:

https://github.com/vqhBook/python/blob/master/lesson09/number_digits3.py

```

1 n = int(input("Nhập số có không quá 3 chữ số: "))
2 tổng = 0
3
4 for _ in range(3):
5     tổng += n % 10
6     n //= 10
7
8 print("Tổng các chữ số là:", tổng)
```

Với nhận xét “cuối cùng thì n là 0”, ta có thể viết lại bằng vòng lặp while như sau:

https://github.com/vqhBook/python/blob/master/lesson09/number_digits4.py

```

1 n = int(input("Nhập số nguyên không âm: "))
2 tổng = 0
3
4 while n > 0:
5     tổng += n % 10
6     n //= 10
7
8 print("Tổng các chữ số là:", tổng)
```

Khi số n có không quá 3 chữ số thì 2 vòng lặp này tương đương. Nhưng ... hồ biến, chương trình sau (dùng vòng lặp while) “tự động” chạy được cho số có số lượng chữ số bất kỳ, tức là hữu hạn nhưng không cố định và không bị giới hạn. Đây là kiểu vô hạn của thực tế và các cấu trúc **lặp** (iterative) hay **vòng** (circular) giúp ta xử lý nó. Như vậy, *sức mạnh thực sự của lặp* là “vượt qua hữu hạn”.

9.2 Phương pháp vét cạn

Trong Phần 7.4, ta đã giải một bài toán cổ bằng cách “mò nghiệm”. Tên gọi chính thống hơn cho phương pháp này là **tìm kiếm vét cạn** (brute-force search, exhaustive search)¹ vì ta kiểm tra tất cả các trường hợp, không bỏ sót trường hợp nào. Đây là

¹ Còn được gọi là: “**thử sai**” (trial and error), “**sinh và thử**” (generate and test), “**đoán và kiểm tra**” (guess and check).

một **phương pháp giải quyết vấn đề** (problem-solving method/technique) rất đơn giản, tổng quát và mạnh mẽ. Vì dựa nhiều vào khả năng lập của máy tính (và Python) nên cũng dễ hiểu khi nó còn được gọi là phương pháp “trâu bò” và là điển hình của câu châm ngôn “cần cù bù thông minh”.

Phương pháp vết cạn áp dụng được khi **không gian tìm kiếm** (search space), tức tập các trường hợp phải thử, là hữu hạn. Với bài toán cổ ở Phần 7.4, ta chỉ có $36 \times 36 = 1296$ trường hợp, không chỉ hữu hạn mà còn rất nhỏ. Trường hợp không gian tìm kiếm hữu hạn nhưng rất lớn thì phương pháp vết cạn dù khả thi về lý thuyết nhưng không thực tế. Chẳng hạn, với yêu cầu viết hàm kiểm tra một số nguyên lớn hơn 1 có là số nguyên tố hay không ở Bài tập 8.3, dùng vết cạn, có thể bạn đã viết theo các cách sau:

<https://github.com/vqhBook/python/blob/master/lesson09/prime.py>

```

1  import math
2
3  def prime1(n): # n là số nguyên > 1
4      ndivs = 0 # số lượng ước số dương của n
5      for i in range(1, n + 1):
6          if n % i == 0: # i là ước của n
7              ndivs += 1
8      return (ndivs == 2) # n có đúng 2 ước số
9
10 def prime2(n): # n là số nguyên > 1
11     for i in range(2, n):
12         if n % i == 0:
13             return False # n có ước > 1 và < n
14     return True
15
16 def prime3(n): # n là số nguyên > 1
17     for i in range(2, math.isqrt(n) + 1):
18         if n % i == 0:
19             return False # n có ước > 1 và <= căn n
20     return True

```

Tôi đưa ra 3 phiên bản tùy theo bạn giỏi Toán tới đâu.² Tất cả các phiên bản đều dùng vết cạn để kiểm tra tất cả các trường hợp ước số “có thể” của n .

- Phiên bản 1 (hàm `prime1`), “**ngây thơ**” (naïve) nhất, đếm số ước số từ 1 đến n của n vì số lớn hơn n không thể là ước của n được (hiển nhiên không ta?!). Bạn cũng học hỏi cách `return` ở Dòng 8 mà không cần dùng `if-else`.
- Phiên bản 2 (hàm `prime2`), tinh tế hơn khi dùng chữ “chỉ” trong định nghĩa của số nguyên tố. Cụ thể, nếu n chia hết cho bất kì số nào (dù chỉ 1 số) lớn hơn 1 và nhỏ hơn n thì n không nguyên tố, ngược lại (n chỉ chia hết cho 1 và n) thì n là nguyên tố.

²Nhớ là bạn mới học cấp 2 thôi nhé!:)

- Phiên bản 3 (hàm `prime3`), tương tự phiên bản 2 với nhận xét thêm “nếu n không nguyên tố thì n phải có ước $\leq \sqrt{n}$ ”. Cũng dễ thấy: nếu n chia hết cho số nguyên dương a thì có số nguyên dương b để $n = ab$; khi đó a, b không thể cùng $> \sqrt{n}$ được vì như vậy ab sẽ lớn hơn n , nên một trong a, b phải $\leq \sqrt{n}$. Lưu ý, hàm `math.isqrt` giúp tìm phần nguyên của căn chính xác hơn cách lấy căn rồi lấy phần nguyên (`int(n**0.5)`).

Chạy chương trình trên để có định nghĩa của 3 hàm và tương tác tiếp trong phiên làm việc đó với Python như sau:

```
===== RESTART: D:\Python\lesson09\prime.py =====
1 >>> from time import time
2 >>> t = time(); prime1(100_000_000); time() - t
False
19.779755353927612
3 >>> t = time(); prime2(100_000_000); time() - t
False
0.06898260116577148
4 >>> t = time(); prime3(100_000_000); time() - t
False
0.023529529571533203
5 >>> t = time(); prime1(100_000_007); time() - t
True
20.29127287864685
6 >>> t = time(); prime2(100_000_007); time() - t
True
20.056764602661133
7 >>> t = time(); prime3(100_000_007); time() - t
True
0.011209726333618164
8 >>> t = time(); prime3(18_014_398_777_917_439); time() - t
True
31.349416494369507
9 >>> t = time(); prime3(9090_9090_9090_9090_9090_9090_91);
↳ time() - t
(Đợi ... đợi nữa ... đợi mãi!)
```

Ta đã dùng hàm `time.time` để đo thời gian chạy các lời gọi hàm tương ứng (tức là đo thời gian Python kiểm tra tính nguyên tố của các số theo các phương án khác nhau ở trên). Mẹo là tính hiệu thời gian sau và trước lời gọi hàm (đơn vị tính là giây). Trường hợp số 100,000,000 không là nguyên tố thì `prime2` chạy nhanh hơn `prime1` nhiều nhưng trường hợp số nguyên tố 100,000,007 thì `prime2` không nhanh hơn `prime1` bao nhiêu, còn `prime3` chạy nhanh hơn nhiều trong cả 2 trường hợp.

Tuy nhiên, kể cả với `prime3`, thời gian kiểm tra số 18,014,398,777,917,439 cũng khá lâu (gần 32 giây). Nếu bạn cho rằng tôi thiếu kiên nhẫn thì thử chạy với số nguyên tố 9090,9090,9090,9090,9090,9090,9090,91 nhé.³ (Lệnh cuối ở trên vẫn đang chạy khi tài liệu này đến tay bạn và sẽ tiếp tục chạy khi tài liệu này rời tay bạn và cả khi bạn ... rời khỏi thế gian này! Thật sự, nó cần vài chục triệu năm để chạy xong!)

Bạn có thể cho rằng máy “mạnh” hơn sẽ chạy nhanh hơn. Xin thưa, con số 9090,9090,9090,9090,9090,9090,9090,91 là con số nhỏ, nó chỉ có 30 chữ số thập phân! Nếu gấp đôi số lượng chữ số này lên thì sức mạnh tính toán của tất cả các máy tính trên thế giới này gộp lại cũng không thể giúp kiểm tra được (theo phương pháp vét cạn trên). Người ta thậm chí cần kiểm tra tính nguyên tố của những con số có hàng trăm chữ số.

Vấn đề nằm ở phương pháp chứ không phải độ trâu bò của máy. Đúng là Python (hay máy) không biết mệt (mặc dù trâu bò cũng biết mệt) nhưng “thời gian là vàng”, về nguyên tắc, cả 3 hàm trên có thể giúp ta kiểm tra tính nguyên tố của một con số nguyên lớn bất kì với thời gian hữu hạn nhưng có thể vượt xa tuổi của vũ trụ. Rõ ràng, ta cần phương pháp khác để vượt qua điều này. Như bạn sẽ thấy, Toán học lại soi đường cho ta.

9.3 Phương pháp chia đôi và phương pháp Newton

Làm thế nào tính căn bậc hai của một số thực dương? Quá dễ! Trong Python, ta có thể dùng nhiều cách như trong Bài tập 3.4. Câu hỏi là: ta có thể tự mình làm điều này không? Hay Python (hoặc máy) đã làm điều này như thế nào? Hay đúng hơn, làm thế nào tính căn bằng các phép toán cơ bản là cộng, trừ, nhân, chia? Khó! Ngược lại, phép bình phương rất dễ thực hiện. Ta sẽ dựa trên phép tính này để mò căn: cho số thực x dương, ta có $y = \sqrt{x}$ khi và chỉ khi y dương và $x = y^2$.

Câu hỏi bây giờ là làm sao mò y khi mà không gian tìm kiếm là vô hạn (có vô số số thực trong một khoảng bất kì). Rõ ràng, về lý thuyết, phương pháp vét cạn không khả thi trong trường hợp này. Tuy nhiên, vì ta chỉ tìm gần đúng $y \approx \sqrt{x}$,⁴ nên, dùng sức trâu bò của vét cạn kết hợp với 2 nhận xét sau đây:

(1) y nằm trong khoảng từ 0 đến x nếu $x > 1$ và từ 0 đến 1 nếu $x \leq 1$,

(2) nếu $0 \leq y_1 \leq y_2$ thì $y_1^2 \leq y_2^2$,

ta có một phương pháp mò nghiệm điển hình là **phương pháp chia đôi** (bisection method).

Cụ thể, ta mò căn trong khoảng $[l, r]$ ($l \leq r$) và cô thu hẹp khoảng này cho đến khi tìm thấy. Ban đầu, do nhận xét (1) ta có $l = 0$ và $r = \max(x, 1)$. Kiểm tra giá trị ở giữa khoảng $y = (l + r)/2$: nếu $y^2 = x$ thì ta đã tìm thấy \sqrt{x} , nếu $y^2 > x$ thì theo nhận xét (2) ta có \sqrt{x} nằm trong khoảng $[l, y]$, nếu $y^2 < x$ thì \sqrt{x} nằm trong khoảng $[y, r]$. Bằng cách lặp lại việc này, ta thu hẹp khoảng chứa căn cho đến khi tìm thấy.

³Tôi viết tách các chữ số “hơi kì” nhưng để thấy dạng dễ gõ.

⁴Ta thậm chí không thể tìm được chính xác \sqrt{x} vì nó là số vô tỉ nếu x không chính phương.

Phương pháp chia đôi rất đơn giản và có thể dùng để tìm nghiệm của nhiều loại phương trình (và giải được nhiều bài toán) khác nhau. Nếu ta tự mình thực hiện các phép toán thì phương pháp này không khả thi do số lần lặp có thể rất lớn, nhưng nếu ta viết chương trình để máy tính toán thì phương pháp này hoàn toàn khả thi và là phương pháp hay do tính đơn giản và khả năng ứng dụng cao.

Một phương pháp mò nghiệm khôn khéo hơn, **phương pháp Newton** (Newton's method), cũng có thể giúp ta tính căn bậc 2 của một số thực dương x bằng cách xuất phát từ một giá trị suy đoán ban đầu y_0 (chẳng hạn là x hay $\frac{x}{2}$), ta lần lượt tính các giá trị y_1, y_2, \dots cho đến khi được giá trị $y \approx \sqrt{x}$ (tức là $y^2 \approx x$) theo công thức cập nhật sau:⁵

$$y_{n+1} = \frac{y_n}{2} + \frac{x}{2y_n}$$

Phương pháp Newton thường hiệu quả hơn phương pháp chia đôi rất nhiều (tức là số lần lặp để đạt cùng mức sai số là ít hơn, hay cùng số lần lặp thì sai số nhỏ hơn). Tuy nhiên, phương pháp này đòi hỏi việc thiết lập công thức cập nhật (như ở trên) riêng cho từng bài toán (phương pháp chia đôi thì giống nhau, luôn cập nhật là số giữa khoảng) mà việc này cần kiến thức Toán “cao cấp”, cụ thể là việc tính “đạo hàm”.⁶

Mã sau đây cài đặt việc tính căn theo 2 phương pháp nói trên. Lưu ý cách “kiểm tra bằng trên số thực” trong Python (xem lại Phần 3.1 và Bài tập 3.1).

https://github.com/vqhBook/python/blob/master/lesson09/bisection_sqrt.py

```

1 import math
2
3 def bisec_sqrt(x): # x > 0
4     l, r = 0, max(x, 1)
5     y = (l + r)/2
6     while not math.isclose(y*y, x):
7         if y*y > x:
8             r = y
9         else:
10            l = y
11            y = (l + r)/2
12    return y
13
14 def Newton_sqrt(x): # x > 0
15    y = x/2
16    while not math.isclose(y*y, x):
17        y = y/2 + x/(2*y)
18    return y

```

Các minh họa trên cho thấy vết cặn là một phương pháp giải quyết vấn đề tốt nhưng trong nhiều trường hợp ta cần khéo léo hướng dẫn việc “vét” hay “mò” chứ

⁵Xem lại Bài tập 2.7, Bài tập 4.2 và mã Newton_sqrt.py ở Phần 8.2.

⁶Bạn nên để ý “đạo hàm” khi học Toán cấp 3 nhé. Đó là thứ thực sự quan trọng và thú vị.

không để tự nhiên, ngẫu thơ (“cạn”) được. Cách tốt nhất, như vậy là, tận dụng thể mạnh của máy (khả năng lặp không mệt mỏi) kết hợp với thể mạnh của ta (khéo léo, khôn ngoan, Toán) để có được những chương trình (hay thuật toán, cách thức) tốt giúp giải quyết vấn đề, bài toán.

9.4 Tính lặp

Một công thức tính toán được gọi là **công thức đóng** (closed-form expression) nếu nó dùng cố định (giới hạn và không quá nhiều) các **phép toán số học cơ bản** (basic arithmetic operation) là cộng, trừ, nhân, chia.⁷ Khi có công thức đóng, việc tính toán là đơn giản và hiệu quả. Ngược lại, khi không có công thức đóng, ta cần các kĩ thuật phức tạp (thường kém hiệu quả, kém chính xác) hơn để tính toán, đặc biệt là **kĩ thuật tính lặp** (iterative method).

Chẳng hạn, cho trước số nguyên không âm n , tổng các số nguyên từ 1 đến n , kí hiệu $S(n)$ có thể được tính bằng 2 cách:

$$S(n) = 1 + 2 + \dots + n = \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

Cách thứ nhất là dùng công thức đóng (công thức sau cùng) với chỉ 1 phép cộng, 1 phép nhân và 1 phép chia. Cách thứ 2 là tính lặp bằng việc thực hiện $n - 1$ phép cộng (không giới hạn vì n tùy ý) các số từ 1 đến n . **Kí pháp sigma** (Σ) là cách Toán hay dùng để mô tả tổng lặp này và ngữ nghĩa của nó rõ ràng là một vòng `for`:

```

1 def S(n):
2     s = 0
3     for i in range(1, n + 1):
4         s += i
5     return s

```

Đĩ nhiên, trong trường hợp này, để tính $S(n)$, ta nên dùng công thức đóng. Tuy nhiên, rất nhiều giá trị cần tính không có công thức đóng cho nó; khi đó, ta buộc lòng dùng các phương pháp như phương pháp lặp để tính nó. Chẳng hạn, một giá trị rất gần gũi với “tổng n số nguyên dương đầu tiên” ở trên là “tích n số nguyên dương đầu tiên”, $P(n) = n! = 1 \times 2 \times \dots \times n$, lại không có công thức đóng. Ta tính nó bằng kĩ thuật lặp như ở Bài tập 7.1. Cũng vậy, ta không có công thức đóng cho thao tác tính căn (và ta cũng đã dùng các kĩ thuật tính lặp như chia đôi hay Newton để tính) và nhiều thao tác quan trọng khác.

Bạn đã biết các hàm lượng giác sin, cos, tan, cotg và cách tính chúng bằng Python ở Phần 3.2. Thế Python tính chúng (chẳng hạn cos) bằng cách nào khi không có công thức đóng cho chúng. Một lựa chọn là dùng kĩ thuật tính lặp dựa

⁷Việc xem thao tác nào là cơ bản cũng tùy ngữ cảnh, nên rõ hơn, ta thường nói là đóng theo tập thao tác nào đó.

trên công thức sau:⁸

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}$$

Dấu 3 chấm (...), hay kí hiệu ∞ trong kí pháp sigma, nói rằng bạn cần cộng vô hạn thừa số, cho nên tổng này còn được gọi là **tổng vô hạn** (infinite sum, series). Hiển nhiên, không ai có thể tính tổng vô hạn này (ngoại trừ vài trường hợp đặc biệt như khi $x = 0$ thì $\cos 0 = 1$, ...). Cũng như hầu hết các trường hợp số thực vô tỉ khác, ta chỉ có thể tính gần đúng các giá trị này (các hàm `math.sin`, `math.cos`, `math.sqrt`, ... cũng chỉ tính gần đúng), mà một cách đó là thay tổng vô hạn bằng **tổng hữu hạn** (finite sum) với số lượng **số hạng** (term) lớn, nghĩa là:

$$\cos x \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^n \frac{x^{2n}}{(2n)!} = \sum_{i=0}^n (-1)^i \frac{x^{2i}}{(2i)!}$$

với n đủ lớn (ta đã thay vô cùng (∞) thành n lớn và dấu bằng (=) thành dấu xấp xỉ (\approx)). Khi đó, ta có thể dùng các vòng lặp của Python để tính. Bạn có thể sẽ viết theo các cách sau:

https://github.com/vqhBook/python/blob/master/lesson09/Maclaurin_cos.py

```

1 from math import factorial, radians, cos
2
3 def mycos1(x, n=20):
4     s = 0
5     for i in range(0, n + 1):
6         s += (-1)**i * x**(2*i)/factorial(2*i)
7     return s
8
9 def mycos2(x, n=20):
10    s = 0
11    for i in range(0, n + 1):
12        mu_2i, gt_2i = 1.0, 1.0 # tính x^(2i) và (2i)!
13        for j in range(1, 2*i + 1):
14            mu_2i *= x
15            gt_2i *= j
16        s += (1 if i%2 == 0 else -1) * mu_2i/gt_2i
17    return s
18
19 def mycos3(x, n=20):
20    s = term = 1.0 # tính (-1)^i * x^(2i)/(2i)!
21    for i in range(1, n + 1):
22        term *= -1 * x*x / ((2*i-1)*(2*i))

```

⁸Công thức này được gọi là chuỗi **Maclaurin** (Maclaurin series) trong Toán cao cấp. Nó cũng được dẫn ra từ cái đạo hàm mà ta đã nói.

```

23         s += term
24     return s

```

Tôi đã đưa ra 3 phiên bản tùy theo bạn giỏi Tin tới đâu:) Chạy chương trình trên để có định nghĩa của 3 hàm và tương tác tiếp với Python như sau:

```

===== RESTART: D:\Python\lesson09\Maclaurin_cos.py =====
1 >>> print(cos(radians(60)), mycos1(radians(60)))
0.5000000000000001 0.5000000000000001
2 >>> print(mycos2(radians(60)), mycos3(radians(60)))
0.5000000000000001 0.5000000000000001

```

Sau đây là các phân tích thêm cho 3 phiên bản:

- Phiên bản 1 (hàm mycos1), “tự nhiên” nhất, dùng toán tử mũ ($**$) và hàm factorial để tính giai thừa. Ngoài việc các thao tác mũ và tính giai thừa không phải là cơ bản thì hàm mycos1 có thể bị lỗi thực thi khi n lớn do “**tràn số**” vì chuyển số nguyên lớn (giai thừa lớn rất nhanh) thành số thực, chẳng hạn, gọi mycos1(radians(60), n=100) sẽ bị lỗi OverflowError.
- Phiên bản 2 (hàm mycos1), “trâu bò” thực sự khi tự tính lấy x^{2i} và $(2i)!$, do đó có thể tránh được tràn số bằng cách dùng số thực cho giai thừa (việc đặt `gt_2i = 1.0` ở Dòng 12 là rất quan trọng). Chẳng hạn, gọi mycos2(radians(60), n=100) cho kết quả tốt. Tuy nhiên, vì dùng vòng lặp lồng nên thời gian chạy rất lâu khi n lớn, chẳng hạn, gọi mycos2(radians(60), n=7000) sẽ đợi khá lâu.

Hơn nữa, hàm này tính tách bạch x^{2i} và $(2i)!$ rồi mới thực hiện phép chia sau cùng nên kết quả có thể “**không xác định**” vì chia số quá lớn cho số quá lớn, chẳng hạn, gọi mycos2(radians(60 + 360), n=1000) sẽ bị nan.⁹

- Phiên bản 3 (hàm mycos1), “tinh tế” hơn, khử vòng lặp lồng của phiên bản 2 bằng cách tính “kế thừa” các số hạng của tổng với nhận xét “số hạng sau bằng số hạng trước nhân thêm $\frac{-x^2}{(2i-1)(2i)}$ ”. Cần để ý đến việc khởi động số hạng (biến term) và tổng (biến s) bằng việc tách riêng số hạng đầu tiên ($i = 0$) mà giá trị của nó là 1 (bạn chỉ cần thế $i = 0$ vào công thức của số hạng $(-1)^i \frac{x^{2i}}{(2i)!}$). Phiên bản này chạy nhanh hơn nhiều và cũng khó bị nan hơn. Chẳng hạn, gọi mycos3(radians(60 + 360), n=7000) vẫn cho kết quả rất nhanh và tốt, thậm chí cả lời gọi mycos3(radians(60 + 5*360), n=100000) cũng vậy.

Tuy nhiên, kể cả với phiên bản này, thì lời gọi mycos3(radians(60 + 10*360), n=100000) cũng cho kết quả không tốt. Lí do là khi x lớn, ta cần n rất rất lớn để có thể xấp xỉ $\cos x$ bằng tổng hữu hạn trên. Rất may, vì hàm $\cos x$ là hàm “tuần hoàn” với “chu kì” 360° ,¹⁰ nên ta chỉ cần “chia lấy

⁹nan là viết tắt của not a number, là một giá trị thực (float) hợp lệ được Python (và máy tính nói chung) dùng để chỉ kết quả tính không xác định.

¹⁰Hướng theo góc nào đó, quay thêm 360° , ta sẽ vẫn ở hướng cũ.

đư” cho 360° để đưa góc về khoảng 0° đến 360° mà không thay đổi gì. Thêm dòng lệnh sau:

```
20 x = x % math.radians(360)
```

vào ngay sau dòng tiêu đề hàm `mycos3` (chen vào giữa Dòng 19, 20) ta sẽ được phiên bản hoàn hảo (đến giờ!).¹¹ Bạn thử liền cho nóng!

9.5 Vẽ hình bất kì

Ta đã thấy con rùa vẽ đoạn thẳng (hàm `forward`, `goto`) và cung tròn (hàm `circle`). Thật ra, con rùa chỉ biết vẽ mỗi đoạn thẳng mà thôi. Không tin, thử vẽ hình tròn bán kính 200 với tâm tại gốc tọa độ như sau:

```
1 >>> import turtle as t
2 >>> t.up(); t.forward(200); t.left(90); t.down();
   ↪ t.circle(200, steps=6)
```

Như bạn thấy, con rùa vẽ đa giác đều có 6 cạnh chứ không phải là hình tròn như mong đợi. Thật sự, con rùa không biết vẽ hình tròn, hàm `circle` chỉ vẽ đa giác đều với số cạnh do tham số `steps` qui định. Lạ nhỉ, chẳng phải ta đã vẽ hình tròn (và cung tròn) trong các bài trước đó sao. Thật ra, *hình tròn chỉ là ảo giác của đa giác đều khi số cạnh lớn*. Bạn thử đặt `steps` lớn hơn sẽ thấy hình tròn xuất hiện. Bạn cũng nên biết tham số `steps` có giá trị mặc định là `None` mà nếu không được truyền đối số lúc gọi (như cách ta đã dùng trong các bài trước) thì giá trị `None` này sẽ ra hiệu cho Python tính toán tự động số cạnh đủ lớn để đa giác đều trông như hình tròn. Đây cũng là một cách dùng `None` hay mà bạn nên học hỏi.

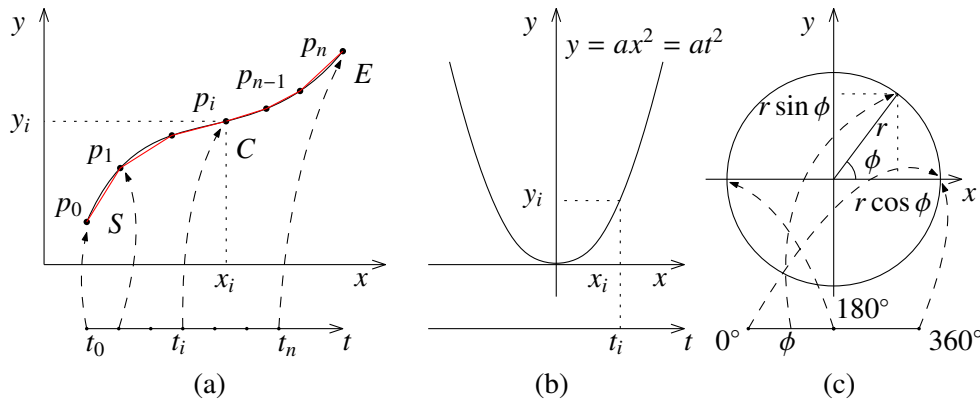
Vậy, nếu như con rùa chỉ biết vẽ mỗi đoạn thẳng thì làm sao có thể bảo nó vẽ các đường phức tạp hơn? Cũng cùng nguyên lý như vẽ đường tròn, bằng cách cho con rùa lặp lại việc vẽ nối các đoạn thẳng với nhau (một cách hợp lí cho đường cần vẽ) với số lần lặp đủ lớn thì “ảo giác” sẽ cho ta đường mong đợi.¹² Chẳng hạn, như minh họa ở Hình (a) dưới, để vẽ đường C từ điểm bắt đầu S đến điểm kết thúc E , ta cho con rùa bò từ S đến E qua nhiều điểm trung gian giữa S và E trên đường C .

Để dễ hình dung ta gọi t_S là thời điểm con rùa ở S (thời điểm bắt đầu) và t_E là thời điểm con rùa ở E (thời điểm kết thúc), hiển nhiên $t_S \leq t_E$. Chia khoảng thời gian từ t_S đến t_E ra n khoảng bằng nhau để có $n + 1$ thời điểm với thời điểm thứ i ($0 \leq i \leq n$) là $t_i = t_S + i \times \frac{t_E - t_S}{n}$ (như vậy thời điểm ban đầu là $t_0 = t_S + 0 \times \frac{t_E - t_S}{n} = t_S$ và thời điểm kết thúc là $t_n = t_S + n \times \frac{t_E - t_S}{n} = t_E$). Gọi p_i là vị trí (điểm) trên đường C mà con rùa bò đến tại thời điểm t_i . Bằng cách cho con rùa xuất phát ở $p_0 = S$ rồi

¹¹Nhớ lại là đơn vị đo góc được dùng mặc định là radian chứ không phải độ. Lệnh này cũng cho thấy toán tử chia lấy dư (%) có thể làm việc với số thực.

¹²Vì mắt người có “độ phân giải” giới hạn nên ảo giác chính là minh chứng cho việc không có vô hạn trong thực tế như đã nói.

bò (thẳng) đến p_1 rồi bò (thẳng) đến p_2, \dots và kết thúc ở $p_n = E$ ta sẽ có đường nối nhiều đoạn thẳng mà khi n (số điểm) đủ lớn thì “trông như” đường C .



Dĩ nhiên, điều quan trọng là phải xác định được các điểm p_i cho đường C , nghĩa là xác định các hoành độ x_i và tung độ y_i của điểm tại t_i . Ta dễ dàng làm việc này nếu **tham số hóa** (parameterization) được đường C theo tham số t , nghĩa là tìm được 2 hàm f và g sao cho:

$$\begin{cases} x = f(t) \\ y = g(t) \end{cases} \quad \text{với } t_S \leq t \leq t_E$$

Khi đó, với kỹ thuật **rời rạc hóa** (discretization) thời gian ở trên (tức là chỉ xét các thời điểm rời rạc t_i chứ không phải tất cả các thời điểm liên tục từ t_S đến t_E) ta có các x_i, y_i đơn giản là:

$$\begin{cases} x_i = f(t_i) \\ y_i = g(t_i) \end{cases} \quad \text{với } 0 \leq i \leq n$$

Tức là $(x_i, y_i) = (f(t_i), g(t_i))$ là vị trí của con rùa (trên đường C) tại thời điểm t_i .

Ta tạo một hàm trong module `figures` (từ các bài trước) để con rùa cung cấp dịch vụ này như sau:¹³

```

1 https://github.com/vqhBook/python/blob/master/lesson09/figures.py
2 def para_curve(f, g, tS, tE, n):
3     d = (tE - tS)/n
4     for i in range(n + 1):
5         ti = tS + i*d
6         x, y = f(ti), g(ti)
7         if i == 0:
8             t.up()
9         else:
```

¹³Ở đây chỉ nêu “trọng tâm”, bạn mở link tương ứng để có mã nguồn đầy đủ, chi tiết hơn.

```

9         t.down()
10        t.goto(x, y)

```

Tôi không giải thích gì thêm ngoài lưu ý là ta cần nhắc bút trong lần đầu để đưa con rùa đến điểm xuất phát ($S = p_0$). Cũng lưu ý là hàm trên rất mạnh, nó có thể giúp ta vẽ hình bất kì miễn là ta tham số hóa được hình đó (xác định được các hàm f, g và các biên thời gian t_S, t_E phù hợp). Tham số n xác định số lượng điểm chia, nó quyết định “độ phân giải” hay “độ mịn” của hình (n chính là steps của hàm vẽ đường tròn circle).

Chương trình sau sử dụng hàm trên để vẽ một vài hình phức tạp:

```

1  https://github.com/vqhBook/python/blob/master/lesson09/para\_curves.py
2  import turtle as t
3  from math import cos, sin, radians as rad
4  from figures import para_curve as pcur, coordinate_system
5
6  t.setup(600, 600)
7  t.tracer(False)
8  coordinate_system(-4, -2, 4, 6)
9
10 pcur(lambda t: t, lambda t: t**2, -4, 4, 30,
11       line_color="red")
12 pcur(lambda t: t, lambda t: t + 2, -4, 4, 1,
13       line_color="green")
14 pcur(lambda t: 2*cos(rad(t)), lambda t: 2*sin(rad(t)),
15       0, 360, 50, line_color="blue")
16 pcur(lambda t: 4*cos(rad(t)), lambda t: 2*sin(rad(t)),
17       0, 360, 50, line_color="orange")
18 t.update()

```

Như kết xuất cho thấy, ta đã vẽ một parabol ($y = x^2$), một đường thẳng ($y = x + 2$), một đường tròn tròn (tâm O bán kính 2) và một ellipse (tâm O bán trục lớn 4, bán trục nhỏ 2). Cũng lưu ý, tôi đã đặt lại hệ trục cho cửa sổ là hoành độ từ -4 đến 4 , tung độ từ -2 đến 6 và vẽ hệ trục bằng hàm `coordinate_system` của module `figures`. Con rùa cho phép ta làm điều này như đã biết ở Bài tập 5.5. Bạn xem thêm mã nguồn của hàm `coordinate_system` trong module `figures` để hiểu rõ hơn (<https://github.com/vqhBook/python/blob/master/lesson09/figures.py>).

Parabol là một đường bậc 2 vì nó có phương trình xác định tung độ theo hoành độ là $y = ax^2$ ($a \neq 0$).¹⁴ Để vẽ đường này, ta có thể tham số hóa nó bằng cách “dùng hoành độ làm thời gian” như minh họa ở Hình (b) trên. Chẳng hạn, để vẽ đồ thị hàm số $y = x^2$ trong khoảng $-4 \leq x \leq 4$ như minh họa, ta dùng cách tham số

¹⁴Đây chỉ là dạng đơn giản nhất được giới thiệu ở Toán Lớp 9 thôi. Parabol tổng quát thì phức tạp hơn. Cũng lưu ý là hàm `para_curve` trên giúp ta vẽ mọi dạng parabol.

hóa sau:

$$\begin{cases} x = f(t) = t \\ y = g(t) = t^2 \end{cases} \quad \text{với } -4 \leq t \leq 4$$

Vì các hàm f, g quá đơn giản nên ta đã dùng biểu thức lambda để tạo hàm vô danh mà không cần định nghĩa tường minh chúng.

Hình thứ 2 ta vẽ là đường thẳng có phương trình $y = x + 2$. Ta cũng đã “dùng hoành độ làm thời gian” để tham số hóa nó. Dĩ nhiên ta vẫn có thể dùng nhiều đoạn thẳng để vẽ đoạn thẳng! Tuy nhiên, ta chỉ cần dùng 1 đoạn thẳng để vẽ đoạn thẳng! (Tôi đã đặt 1 cho tham số n để dùng 1 khoảng chia, bạn có thể tăng số khoảng chia lên nhưng đường thẳng sẽ không mịn hơn mà tốn thời gian để vẽ hơn!)

Tham số hóa một đường tròn thì khó hơn. Tuy nhiên nếu nhớ lại cách ta hay dùng để vẽ đường tròn thì sẽ thấy dễ thôi. Để vẽ hình tròn tâm tại O bán kính r , ta cắm đỉnh compa tại O , dờ đầu compa rộng khoảng r , rồi quay compa đủ 360° ta sẽ có hình tròn. Ta, như vậy, đã tham số hóa đường tròn theo góc quay ϕ . Cố định tâm và bán kính, cho ϕ nhận các giá trị từ 0° đến 360° ta sẽ có đường tròn như minh họa ở Hình (c) trên. Hơn nữa nhớ lại hàm lượng giác để biết cách tìm các cạnh của tam giác vuông có góc là ϕ . Tóm lại, ta tham số hóa đường tròn như sau (với t là ϕ):

$$\begin{cases} x = f(t) = r \cos t \\ y = g(t) = r \sin t \end{cases} \quad \text{với } 0 \leq t \leq 360$$

Ta chưa học về đường ellipse trong Toán cấp 2 nhưng ta gặp hoài ngoài thực tế. Bạn tưởng tượng, ta ép hay kéo dãn đường tròn sẽ có hình ellipse (giống như đập trái banh xuống). Hình ellipse, như vậy, linh hoạt hơn hình tròn, nó có 2 bán kính (thay vì chỉ có 1 bán kính). Tuy nhiên, cách tham số hóa nó chỉ là mở rộng cách tham số hóa trên của đường tròn:

$$\begin{cases} x = f(t) = a \cos t \\ y = g(t) = b \sin t \end{cases} \quad \text{với } 0 \leq t \leq 360$$

Với a là bán kính ngang (theo phương Ox) còn b là bán kính dọc (theo phương Oy).

Có thể, con rùa cũng đã dùng cách này của ta để vẽ đường tròn. Chẳng hạn, thử lệnh sau đây (sau khi chạy module figures để có các hàm tương ứng), ta sẽ thấy kết xuất là “đường tròn” bán kính 200 với tâm tại gốc tọa độ như đã nói:

```
===== RESTART: D:\Python\lesson09\figures.py =====
1 >>> para_curve(lambda t: 200*math.cos(math.radians(t)),
2       lambda t: 200*math.sin(math.radians(t)), 0, 360, 6)
```

9.6 Hoạt họa

Hoạt họa hay **trình diễn** (animation) là kỹ thuật tạo các hình ảnh động có cảm giác di chuyển nhờ việc trình chiếu liên tiếp nhiều hình ảnh tĩnh với tốc độ đủ nhanh. Mánh lới đánh lừa thị giác này cũng tương tự như ảo giác về các đường cong ở phần trên và là nguyên lý đằng sau các loại hình nghệ thuật hay công nghệ như phim hoạt hình (animated cartoon), vô tuyến truyền hình (television), phim ảnh (movie, film) và video.

Hoạt họa là công việc rất cực khổ khi làm “thủ công” vì cần rất nhiều **hình** (frame). Chẳng hạn, một đoạn phim 10 phút cần ít nhất $10 \times 60 \times 24 = 14400$ hình khi trình chiếu ở tốc độ thông thường **24 hình trên giây** (frames per second, fps). Các công nghệ **trình diễn bằng máy tính** (computer-generated animation) giúp giảm thiểu công sức nhờ việc trình diễn lặp lại các đối tượng với “cấu hình” thay đổi theo thời gian. Chẳng hạn, minh họa sau đây “trình diễn” một hình vuông co giãn theo thời gian (nhớ đặt file `figures.py` chung thư mục với file chương trình):

```

https://github.com/vqhBook/python/blob/master/lesson09/bloom\_square.py
1  import turtle as t
2  import time
3  import figures
4
5  def toggle_bloom():
6      global bloom
7      bloom = not bloom
8
9  def run():
10     global bloom, width
11
12     max_width = min(t.window_width(), t.window_height())
13     if width <= 0 and not bloom: bloom = True
14     if width >= max_width and bloom: bloom = False
15     width += (1 if bloom else -1)
16
17     t.clear()
18     figures.square(-width//2, -width//2, width, "red", "red")
19     t.update()
20     t.ontimer(run, 1000//24) # 24 fps
21
22 width = 0; bloom = True
23 t.hideturtle(); t.tracer(False)
24 t.onkey(toggle_bloom, "space"); t.listen()
25 run(); t.exitonclick()

```

Trình diễn trên tuy đơn giản nhưng có một đặc trưng rất mạnh là có thể **tương**

tác (interactive animation). Cụ thể, người dùng có thể nhấn phím Space (phím cách) để thay đổi trạng thái nở hay thu của hình vuông (biến bloom). Nguyên lý trình diễn nằm ở các lệnh từ Dòng 17 đến 20: xóa khung hình cũ, vẽ khung hình mới (cấu hình mới), và tạm dừng trước khi lặp lại (với tốc độ phù hợp). Hàm `turtle.ontimer` cho phép ta đăng ký một hàm (ở đây là `run`) mà sẽ được con rùa gọi thực thi sau một khoảng thời gian (tính theo mili giây). Hàm này, do đó, thường được gọi là **hàm gọi lại** (callback function).

Trong mã trên, ta cũng đã dùng hàm `turtle.exitonclick` để yêu cầu con rùa đóng cửa sổ khi người dùng nhấp chuột lên cửa sổ con rùa và do đó chương trình sẽ kết thúc (thay vì nhấn phím Escape như ở mã `turtle_draw.py` Phần 8.6). Hàm này cũng thực hiện vòng lặp thông điệp (nó tự động gọi `mainloop`).

Lưu ý, trong hàm `toggle_bloom` ta có đặt lại biến `bloom` được định nghĩa bên ngoài hàm nên trong hàm `toggle_bloom` ta phải khai báo nó là “toàn cục” bằng lệnh `global` ở Dòng 6 (ta sẽ tìm hiểu kỹ hơn ở Phần 10.7). Tương tự với khai báo toàn cục cho các biến `bloom` và `width` ở hàm `run`. Bạn cũng thử chạy chương trình bằng cách nhấp đúp file mã.

Trình diễn trên thật ra không chỉ đơn giản mà còn ... rất nghiệp dư vì ta không kiểm soát được cấu hình các đối tượng theo thời gian là yếu tố cực kì quan trọng. Chẳng hạn, bạn có thấy rằng hình vuông co/giãn không được “đều” lắm, nhanh lúc nhỏ nhưng chậm lúc lớn, nếu ta muốn co/giãn đều thì sao? Minh họa sau đây tái hiện một thí nghiệm nổi tiếng của Galilè (Galileo Galilei, nhà khoa học người Ý) trên tháp nghiêng Pisa (Pisa Tower):

https://github.com/vqhBook/python/blob/master/lesson09/Pisa_tower.py

```

1 import turtle as t
2 import time
3 import figures as fg
4
5 def log():
6     print(f"Hit(s): {hit}, Time: {time.time() - st:.2f}s, "
7           f"Speed: {abs(v):.2f}m/s, Height: {h:.2f}m.")
8
9 def render():
10    t.clear()
11    fg.line(0, 0, 3, 0); fg.line(1, 0, 1, HEIGHT)
12    fg.string(1, h, f"{h:.1f}", align="right", font_size=12)
13    fg.line(1, h, 2, h, "blue"); fg.dot(2, h, 10, "red")
14    fg.string(2, h, str(hit), color="green", align="center",
15             font_size=20)
16    t.update()
17
18 def run():
19     global v, h, hit, t0, v0, h0
20
21     tm = time.time() - t0

```

```

21     v, h = v0 - G*tm, h0 + v0*tm - (G/2)*(tm**2)
22     if h < 0:
23         h = 0; hit += 1; t0 = time.time()
24         v0, h0 = K * (VMAX if hit == 1 else v0), 0
25         if v0 == 0:
26             v, h = 0, 0
27             render()
28             return
29
30     render()
31     t.ontimer(run, 1000//24)
32
33 HEIGHT = 55.86; G = 9.8; VMAX = (2*G*HEIGHT)**0.5; K = 0.9
34 st = time.time(); hit = 0
35 t0, v0, h0 = st, 0, HEIGHT
36
37 fg.coordinate_system(0, -HEIGHT/10, 3, HEIGHT*11/10, None)
38 t.hideturtle(); t.tracer(False)
39 t.onkey(log, "space"); t.listen()
40 run(); t.exitonclick()

```

Khác biệt quan trọng trong trình diễn này là ta dùng thời gian thật (`time.time()`) để tham số hóa “cấu hình” của các đối tượng (ở đây là chiều cao so với mặt đất (h) và vận tốc (v) của “quả banh”). Tham số thời gian giúp ta đồng bộ hóa mọi thứ còn hàm `ontimer` và vòng lặp thông điệp chỉ giúp “làm tươi” (refresh) khung hình.

Vì việc vẽ khung hình, còn gọi là **kết xuất** (`render`), trong trình diễn này khá phức tạp nên ta đã tách riêng nó vào hàm `render`. Lưu ý, hàm kết xuất sẽ vẽ khung hình hiện tại theo cấu hình các đối tượng tại thời điểm hiện tại mà ta nên tính trước đó để việc vẽ diễn ra nhanh chóng (tránh giụt khung hình).¹⁵

Cấu hình của quả banh (chiều cao h và vận tốc v) được xác định theo các qui luật thực tế (Vật Lý) như sau:

$$\begin{cases} v = v_0 - gt \\ h = h_0 + v_0t - \frac{1}{2}gt^2 \end{cases}$$

Ngoài ra, để tăng phần thú vị, ta đã cho quả banh bật lên khi chạm đất theo hệ số suy giảm k nhận giá trị trong khoảng $[0, 1]$. Nếu $k = 0$, ta có va chạm “mềm”, nghĩa là vật sẽ mất hoàn toàn động năng khi va chạm và không bật lên. Nếu $k = 1$, ta có va chạm “đàn hồi”, nghĩa là vật sẽ bảo toàn động năng khi va chạm và bật lên với cùng vận tốc nhưng đổi chiều (và do đó sẽ lên lại độ cao ban đầu). Các trường hợp thực tế thường ứng với $0 < k < 1$ (như trường hợp quả banh ở trên, tôi đã đặt $k = 9/10$).

¹⁵Mặc dù vậy, khung hình của ta vẫn còn bị giụt. Đây là một hạn chế của module `turtle`. Ta sẽ xử lý vấn đề này sau bằng kĩ thuật quan trọng gọi là **double buffer**.

Biến hit đếm số lần quả banh chạm đất. Dĩ nhiên, nếu $k = 0$ thì quả banh chỉ chạm đất 1 lần (và dừng), còn $k = 1$ thì quả banh sẽ chạm đất vô hạn lần. Tuy nhiên, các trường hợp khác khá thú vị. Theo bạn, nếu $k = 1/10$ thì quả banh có dừng không? $k = 9/10$ thì sao?¹⁶

Mặc dù trông có vẻ chẳng liên quan gì đến các phần trước nhưng kỹ thuật trình diễn cũng dùng chung một *mô típ lặp quen thuộc* là: *tham số hóa công việc theo một hoặc nhiều tham số, rồi rạc hóa chúng rồi cho làm lặp lại nhiều lần theo dãy giá trị rời rạc đó.*

Tóm tắt

- Sức mạnh thực sự của lặp là giúp xử lý các việc có “khối lượng”, số lượng, “kích thước” không giới hạn.
- Tìm kiếm vét cạn, tức là thử tất cả các trường hợp, là một phương pháp giải quyết vấn đề rất đơn giản và hữu ích. Trong nhiều tình huống, ta phải vận dụng khéo léo vét cạn hoặc phải dùng phương pháp khác để giải bài toán vì vét cạn “ngây thơ” thường không hiệu quả.
- Phương pháp chia đôi và phương pháp Newton là các phương pháp lặp hay được dùng để giải nhiều bài toán.
- Rất nhiều việc tính toán không có cách tính đơn giản bằng các công thức đóng mà phải tính bằng các phương pháp lặp.
- Ta có thể vẽ đường bất kì bằng cách lặp lại việc vẽ nối nhiều đoạn thẳng dựa trên một tham số hóa thích hợp cho đường.
- Trình diễn được hiện thực bằng kỹ thuật trình chiếu lặp lại nhiều “khung hình” tham số hóa theo thời gian.

Bài tập

9.1 Viết hàm xuất một số nguyên không âm (có số chữ số bất kì) với các dấu phẩy (,) phân cách mỗi 3 chữ số (tương tự kết xuất của `print` cho kết quả định dạng bằng hàm `format` với chuỗi đặc tả định dạng “,” như ở Bài tập 3.7). Ví dụ: số 1234567 được xuất ra là 1,234,567.

Gợi ý: nên có thêm tham số xác định kí tự phân cách với giá trị mặc định là kí tự phẩy.

¹⁶Lưu ý, qui luật Vật Lý “bó tay” trong trường hợp này. Đây là không phận của Tin học nên sẽ bị các qui luật của máy chi phối, cụ thể là qui luật của các số thực trên máy (số float):) Tin hay không thì tùy, với $k = 9/10$, quả banh sẽ chạm đất vô hạn lần, nhưng $k = 1/10$ thì sẽ chạm đất hữu hạn lần. What!? Bạn có thể tưởng tượng rằng có con số thực “lượng tử” trên máy, hoặc là bạn không có gì (0.0) hoặc là bạn có đầy đủ nó chứ không thể có một nửa hay 9/10, 1/10 nó. Con số lượng tử như vậy có thể là 5×10^{-324} .

9.2 Dùng kĩ thuật tương tự ở Phần 9.4, khử lồng Bài tập 7.1e để có cài đặt hiệu quả hơn (chạy nhanh hơn) và dùng kĩ thuật đo thời gian chạy ở Phần 9.2 để đánh giá 2 cài đặt (ngây thơ với vòng lặp lồng và tinh vi hơn với khử lồng).

Lưu ý: như chú thích của Bài tập 7.1e cho thấy tổng cần tính có giá trị là $\frac{1-x^{n+1}}{1-x}$, tuy nhiên, công thức này không được gọi là công thức đóng vì phép mũ không là phép toán cơ bản (x^{n+1} không tính được bằng số lần cố định các phép cộng, trừ, nhân, chia).

9.3 Chuỗi Maclaurin. Trong Phần 9.4 ta đã dùng chuỗi Maclaurin để tính gần đúng giá trị $\cos x$. Tương tự, tính gần đúng các hàm sau (không có công thức đóng) với chuỗi Maclaurin được cho:

$$(a) e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

Lưu ý: Trong Bài tập 7.1f ta đã tính gần đúng giá trị này, tuy nhiên, bạn cần khử lồng để có cài đặt chạy nhanh hơn. Hàm e^x tính trong Python bằng hàm `math.exp`.

Gợi ý: Khi x gần 0 thì chuỗi trên “hội tụ nhanh” đến e^x , nghĩa là tổng n số hạng đầu với n nhỏ đã cho kết quả gần đúng. Khi x xa 0 (nghĩa là $|x|$ lớn), ta cần tính tổng rất nhiều số hạng đầu để có giá trị xấp xỉ tốt. Tuy nhiên, tương tự kĩ thuật dùng chu kì tuần hoàn của hàm $\cos x$ ở trên, với công thức sau:

$$e^{y2^k} = (e^y)^{2^k}$$

ta có thể đưa việc tính e^x với x lớn về tính e^y với y nhỏ rồi tính mũ lũy thừa của 2. Chẳng hạn:

$$e^{100} = e^{\frac{100}{2^7} 2^7} = (e^{0.78125})^{2^7}$$

Dùng gợi ý này cùng với hàm tính mũ lũy thừa của 2 (Bài tập 8.1b) để xử lý trường hợp x lớn. (Khó quá thì bỏ qua!)

$$(b) \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}$$

Gợi ý: tương tự \cos , hàm \sin cũng tuần hoàn với chu kì 360° .

9.4 Tương tự các hàm ở Phần 9.3, viết hàm tính căn bậc 3 của một số thực dương bằng:

(a) Phương pháp chia đôi.

(b) Phương pháp Newton.

Gợi ý: để tính $y \approx \sqrt[3]{x}$ (tức là $y^3 \approx x$) dùng công thức cập nhật sau:

$$y_{n+1} = \frac{2y_n}{3} + \frac{x}{3y_n^2}$$

9.5 Ước số chung lớn nhất. Số nguyên d được gọi là **ước số chung lớn nhất** (greatest common divisor, gcd) của 2 số nguyên dương a, b nếu d là số nguyên lớn nhất mà là ước của cả a và b .

Viết hàm tìm ước số chung lớn nhất của 2 số nguyên dương được cho và dùng hàm này tìm ước số chung lớn nhất của các số người dùng nhập (có thể nhiều hơn 2 số).

Gợi ý:

- Có một thuật toán cực kì đẹp để giải bài toán này là **thuật toán Euclid** (Euclid's algorithm). Tuy nhiên, trước khi tôi giới thiệu cho bạn trong bài sau thì bạn nên tự mình suy nghĩ và thử nghiệm các cách (thuật toán) khác nhau.
- Module `math` cũng cung cấp hàm `gcd` để tính ước số chung lớn nhất. Bạn tra cứu hàm này và dùng nó để đối chiếu kết quả với hàm bạn đã cài.

9.6 Vẽ “hình trái tim” theo các phương trình được cho sau:

(a) $y = |x| \pm \sqrt{4 - x^2} \quad (-2 \leq x \leq 2)$

(b)
$$\begin{cases} x = 16 \sin^3(t) \\ y = 13 \cos(t) - 5 \cos(2t) - 2 \cos(3t) - \cos(4t) \end{cases} \quad (0 \leq t \leq 2\pi)$$

Đặt hệ trục tọa độ bằng lời gọi `coordinate_system(-20, -20, 20, 20, coord_color=None)`.

9.7 Viết chương trình (tương tự Phần 9.6) trình diễn các yêu cầu sau:

- Hình tròn co giãn kích thước.
- Hình tròn lăn trên mặt nằm ngang.
- Hình tròn lăn trên mặt nghiêng (có thể điều chỉnh góc nghiêng).
- Hình vuông xoay quanh tâm (có thể điều chỉnh tốc độ quay).
- Con lắc đơn dao động.
- Con lắc lò xo dao động.
- Ngôi sao thay đổi số cánh (xem Bài tập 7.5).
- Hiệu ứng âm bản dương bản: đổi màu hình trong Bài tập 7.7.
- Hiệu ứng Newton với ánh sáng trắng: vẽ hình tròn với 7 màu cầu vồng (đỏ, cam, vàng, lục, lam, chàm, tím) thay đổi đủ nhanh sẽ cho cảm giác là màu trắng.
- Điểm được ném lên theo góc nghiêng sẽ chuyển động theo quỹ đạo là parabol (có thể điều chỉnh góc nghiêng).

9.8 Đo thời gian thực thi với `timeit`. Cách đo thời gian thực thi bằng hàm `time.time` ở Phần 9.2 thực ra rất kém chính xác. Thử minh họa sau để thấy rõ hơn:

```
1 >>> from time import time
2 >>> import math
3 >>> t = time(); math.factorial(10000); time() - t
...
0.35926222801208496
4 >>> def f(N): math.factorial(N); return
5
6 >>> t = time(); f(10000); time() - t
```

```

0.015576839447021484
7 >>> t = time(); f(10000); time() - t
0.031212806701660156
8 >>> t = time(); f(10000); time() - t
0.04689955711364746

```

Ở Dòng 3, ta dự định đo thời gian Python tính $10000!$ bằng hàm `math.factorial`, tuy nhiên, kết quả (khoảng 0.3593 giây) là con số dễ gây hiểu nhầm. Thật ra `math.factorial(10000)` chạy nhanh hơn nhiều, thời gian giữa 2 lần gọi `time()` chủ yếu là đợi Python xuất kết quả ra cửa sổ Console. Để tránh điều này, ở Dòng 4, ta viết hàm `f` mà công việc của nó chỉ là gọi lại hàm `math.factorial` và tránh việc xuất kết quả bằng cách trả về `None`.

Như kết quả của Dòng 6 cho thấy, thời gian đúng hơn cho việc tính $10000!$ là chỉ khoảng 0.0156 giây. Tuy nhiên, kết quả này cũng không đáng tin cậy vì Dòng 7, Dòng 8 lại cho kết quả rất khác. Thật ra, đo thời gian thực thi trên máy là việc không đơn giản vì nó phụ thuộc vào nhiều yếu tố (như tình trạng máy lúc đo) nên mỗi lần đo thường cho kết quả khác nhau.

Để cho kết quả chính xác hơn, ta nên đo nhiều lần và tính thời gian trung bình. Module `timeit` hỗ trợ các chức năng đo thời gian như vậy (ngoài ra còn nhiều yếu tố khác nữa). Chẳng hạn, tiếp tục minh họa trên với `timeit` như sau:

```

9 >>> import timeit
10 >>> timeit.timeit("math.factorial(10000)", number=1000,
    ↪ globals=globals())/1000
0.012902951282000232
11 >>> timeit.timeit("math.factorial(10000)", number=1000,
    ↪ globals=globals())/1000
0.012899772055000085

```

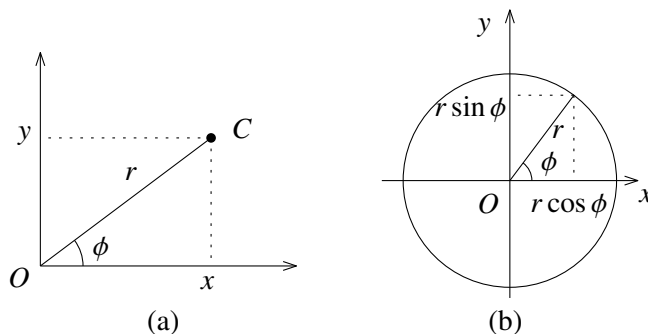
Hàm `timeit.timeit` đo thời gian thực thi một đoạn mã để trong chuỗi lệnh với số lần lặp lại được xác định bởi `number`. Ta đặt `globals` để có thể dùng các tên được định nghĩa trong chương trình, ở đây là module `math` được nạp vào chương trình (ta sẽ tìm hiểu `globals()` ở bài sau). Lưu ý, `timeit` trả về tổng thời gian thực thi nên ta cần chia cho số lần thực thi để tính trung bình.

Dùng `timeit`:

- (a) Đo lại thời gian thực thi ở Phần 9.2.
- (b) So sánh thời gian tính căn bằng phương pháp chia đôi và phương pháp Newton ở Phần 9.3.
- (c) So sánh thời gian thực thi các hàm `cos` ở Phần 9.4.

9.9 Tọa độ cực và hình tham số hóa với tọa độ cực. Ta hay dùng hệ tọa độ Descartes (Cartesian coordinate system) để xác định vị trí của điểm trên mặt phẳng. Trong hệ tọa độ này, vị trí điểm được xác định bởi cặp số (x, y) gồm hoành độ x là khoảng cách theo chiều ngang từ điểm đến trục đứng Oy và tung độ y là khoảng cách theo chiều dọc từ điểm trên trục ngang Ox .

Có một hệ tọa độ khác cũng hay được dùng là **hệ tọa độ cực** (Polar coordinate system). Trong hệ tọa độ này, vị trí điểm được xác định bởi cặp số (r, ϕ) với r là khoảng cách từ điểm đến gốc tọa độ O và ϕ là góc tạo bởi trục ngang Ox và tia nối gốc O đến điểm như minh họa ở Hình (a) dưới đây.



Cũng từ hình minh họa, ta thấy cách chuyển đổi giữa 2 hệ tọa độ:

$$\begin{cases} x = r \cos(\phi) \\ y = r \sin(\phi) \end{cases} \quad \begin{cases} r = \sqrt{x^2 + y^2} \\ \phi = \arctan \frac{y}{x} \end{cases}$$

Trong vài trường hợp, việc dùng tọa độ cực để mô tả các điểm của hình là thuận lợi hơn tọa độ Descartes. Chẳng hạn, ở Phần 9.5 ta đã tham số hóa đường tròn bằng tọa độ Descartes, tuy nhiên, tham số hóa bằng tọa độ cực thì đơn giản hơn như sau:

$$\begin{cases} r = f(t) = R \\ \phi = g(t) = t \end{cases} \quad \text{với } 0 \leq t \leq 360 \text{ và } R \text{ là hằng số xác định bán kính.}$$

Ta tạo một hàm trong module `figures` để con rùa cung cấp dịch vụ vẽ hình tham số hóa với tọa độ cực như sau:¹⁷

```

1 https://github.com/vqhBook/python/blob/master/lesson09/figures.py
2 def polar_para_curve(f, g, tS, tE, n):
3     # ...
4     r, phi = f(ti), g(ti)
5     x, y = r*cos(radians(phi)), r*sin(radians(phi))
6     # ...

```

Chương trình sau sử dụng hàm trên để vẽ một hình tròn và hình xoắn ốc Archimedes (Archimedean spiral) với tham số hóa tọa độ cực sau:¹⁸

$$\begin{cases} r = f(t) = 0.15t \\ \phi = g(t) = t \end{cases} \quad \text{với } 0 \leq t \leq 4 \times 360$$

¹⁷Ở đây chỉ nêu “trọng tâm”, bạn mở link tương ứng để có mã nguồn đầy đủ, chi tiết hơn.

¹⁸https://en.wikipedia.org/wiki/Archimedean_spiral

```

1 https://github.com/vqhBook/python/blob/master/lesson09/pp\_curves.py
2 from figures import polar_para_curve as ppcur
3 ppcur(lambda t: 250, lambda t: t, 0, 360,
4       50, line_color="red")
5 ppcur(lambda t: 0.15*t, lambda t: t, 0, 4*360,
6       4*50, line_color="blue")

```

Dùng cách tham số hóa hình với tọa độ cực, vẽ các đường sau:¹⁹

(a) Hyperbolic spiral:²⁰

$$\begin{cases} r = 300 \frac{360}{t} \\ \phi = t \end{cases} \quad \text{với } 360 \leq t \leq 5 \times 360$$

(b) Golden spiral:²¹

$$\begin{cases} r = \left(\frac{1+\sqrt{5}}{2} \right)^{\frac{t}{90}} \\ \phi = t \end{cases} \quad \text{với } 0 \leq t \leq 3 \times 360$$

(c) Rose:²²

$$\begin{cases} r = 200 \cos\left(\frac{\pi}{180} \frac{5}{2} t\right) \\ \phi = t \end{cases} \quad \text{với } 0 \leq t \leq 2 \times 360$$

(d) Hoa hồng:

$$\begin{cases} r = 200 \cos\left(\frac{\pi}{180} \frac{5}{6} t\right) \\ \phi = t \end{cases} \quad \text{với } 0 \leq t \leq 2 \times 360$$

(e) Tô màu hình hoa hồng:

$$\begin{cases} r = 40 + 200 \cos\left(\frac{\pi}{180} 12t\right) \\ \phi = t \end{cases} \quad \text{với } 0 \leq t \leq 360$$

¹⁹Sau khi thử, có thể xem đáp án tại https://github.com/vqhBook/python/blob/master/lesson09/pp_curves_exercise.py.

²⁰https://en.wikipedia.org/wiki/Hyperbolic_spiral

²¹https://en.wikipedia.org/wiki/Golden_spiral

²²[https://en.wikipedia.org/wiki/Rose_\(mathematics\)](https://en.wikipedia.org/wiki/Rose_(mathematics))

Bài 10

Đối tượng và quản lý đối tượng

Trong Phần 4.3 và 8.4, ta đã biết mọi thứ trong Python đều là đối tượng. Hơn nữa, mở rộng khái niệm kiểu dữ liệu, ta có lớp xác định kiểu của đối tượng. Bài này tìm hiểu các vấn đề cơ bản nhất của đối tượng và việc quản lý chúng bằng không gian tên qua các minh họa trên các lớp có sẵn của Python. Trong Bài 14, ta sẽ tự mình định nghĩa lấy các lớp để tạo ra các dạng đối tượng mới. Phần bài tập giới thiệu một môi trường lập trình rất được ưa chuộng là Visual Studio Code và hướng dẫn rèn luyện kỹ năng debug.

10.1 Dữ liệu, thao tác và đối tượng

Ta đã làm việc với nhiều dạng dữ liệu khác nhau. Bảng sau đây tóm lược các kiểu dữ liệu và thao tác xử lý trên chúng mà ta đã học đến giờ:

Kiểu	Tên kiểu	Hằng	Các thao tác	Bài học
Chuỗi	str	"10", 'Hi '	print, format input, str len, +, *, %	Bài 1, Bài tập 3.7 Phần 4.1, Phần 7.7 Phần 11.7
Số nguyên	int	10, 0xa	số học, so sánh int, float	Bài 2, Bài 3
Số thực	float	1.0, 1e-3		
Luận lý	bool	True, False	luận lý, bool	Phần 2.3, Phần 6.4
None	NoneType	None		Phần 4.4, Phần 8.2 Phần 8.8

Hàm `type` cho biết kiểu và hàm `print` xuất ra mọi dữ liệu mà chúng nhận.¹ Các phép toán số học thao tác trên số (nguyên và thực) gồm: +, - (trừ), *, /, //, %, **, - (đổi). Các phép toán so sánh trả về giá trị luận lý gồm: <, <=, >, >=, ==, !=. Các tính toán số học phức tạp hơn được cung cấp qua các hàm dựng sẵn (`abs`, `round`, ...) và các hàm trong module `math` (`factorial`, `sin`, ...). Kiểu `None` mô tả

¹Hàm `print` gọi hàm `str` để chuyển dữ liệu thành chuỗi trước khi xuất.

“không giá trị” nên hầu như không có bất kì thao tác nào ngoại trừ một số thao tác tổng quát trên mọi kiểu (như `type`, `==`).

Ta cũng có thể dùng lẫn lộn dữ liệu của nhiều kiểu như số nguyên, số thực, luận lý (nhớ lại rằng Python hiểu giá trị luận lý theo nghĩa rộng và `True`, `False` có thể hiểu là 1, 0). Các hàm `str`, `int`, `float`, `bool` được dùng để tạo tương ứng dữ liệu của kiểu chuỗi, số nguyên, số thực, luận lý từ các dữ liệu khác.

Ta cũng đã gặp các dạng dữ liệu “lạ” hơn là cặp số (Phần 2.4), bộ số (Bài tập 4.1), số `Decimal` (Bài tập 3.8), số `Fraction` (Bài tập 3.9), font chữ (Bài tập 5.6), thời gian (`time.struct_time`, Bài tập 4.6) và con rùa (`turtle.Turtle`, Bài 5, mà ta sẽ tìm hiểu kĩ ở phần dưới). Ta cũng sẽ gặp nhiều kiểu dữ liệu khác nữa.

Dữ liệu là tất cả mọi thứ mà chương trình thao tác, xử lý. Một cách bình dân, ta có thể xem dữ liệu là cái/con (mô tả bằng danh từ) còn thao tác là hành động (mô tả bằng động từ) tác động lên chúng. Mối quan hệ giữa dữ liệu và thao tác được mô tả bằng phương trình nổi tiếng của Niklaus Wirth:² “*Thuật toán + Cấu trúc dữ liệu = Chương trình*” (“*Algorithms + Data Structures = Programs*”)³. **Thuật toán** (algorithm) là thuật ngữ tổng quát, hình thức cho các thao tác, xử lý còn **cấu trúc dữ liệu** (data structure) là thuật ngữ tổng quát cho cách tổ chức dữ liệu.

Phương trình trên cho thấy quan hệ khăng khít giữa dữ liệu với thao tác. Hơn như vậy, Python xem chúng đều là các **đối tượng** (object) (với kiểu khác nhau). Thật vậy, *mọi thứ trong Python đều là đối tượng!* Thử minh họa sau:⁴

```

1 >>> print(isinstance(1.0, object), isinstance(False, object))
    True True
2 >>> isinstance(float("nan"), object), isinstance(None, object)
    (True, True)
3 >>> import math
4 >>> isinstance(math, object), isinstance(math.sqrt, object)
    (True, True)
5 >>> isinstance(float, object), isinstance(object, object)
    (True, True)
6 >>> print(callable(1.0), callable(None))
    False False
7 >>> print(callable(math.sqrt), callable(float))
    True True
8 >>> import turtle
9 >>> print(callable(turtle.Turtle), callable(turtle))
    True False

```

²Cha đẻ của ngôn ngữ lập trình Pascal. Một ngôn ngữ lập trình khá nổi tiếng nhưng hiện không còn được ưa chuộng nữa. Nhân tiện, các chương trình học Pascal, như chương trình Tin học ở phổ thông ta nên thay bằng Python.

³Đây là tựa một cuốn sách nổi tiếng của Niklaus Wirth xuất bản năm 1976.

⁴Bạn đã biết dấu phẩy giúp tạo “cặp đối tượng”. Tôi dùng nó ở Dòng 2, 4, 5 như là một “thủ đoạn” nữa để tiết kiệm giấy!

Hàm `isinstance` kiểm tra “thứ” gì đó ở đối số thứ nhất có phải là một “thể hiện” hay một “trường hợp” của kiểu nào đó ở đối số thứ 2 không (ta sẽ rõ hơn ở Phần 14.4). Như ta thấy, mọi thứ đều là đối tượng (kiểu `object`) kể cả `None` (mô tả không có giá trị) lẫn “Not a Number” (mô tả giá trị `float` nhưng “không phải số”).⁵ `object` là kiểu chung, mọi kiểu khác đều là **kiểu đặc biệt** (subtype) của nó (như `int`, `float`, ...). Nói theo ngôn ngữ lớp (Bài 14) thì `object` là lớp “tổ tiên”, mọi lớp khác đều là lớp “con cháu”.

Hàm `callable` kiểm tra xem một đối tượng có “gọi được” (callable) hay không, nghĩa là có phải là hàm (hay tương tự như hàm) không. `float` (cũng như `int`, `object`, ...) là “đối tượng kiểu” và có thể gọi được (bằng chứng là ta đã gọi để tạo số thực như lời gọi `float("nan")` ở Dòng 2); cũng như vậy, `turtle.Turtle` là “đối tượng lớp” mà ta sẽ gọi để tạo các chú rùa ở phần sau. Ngược lại, các “đối tượng dữ liệu” thông thường hay “đối tượng module” thì không gọi được.

10.2 Cuộc đua kỳ thú của 4 thiếu niên ninja rùa đột biến

Trong Bài 5, ta đã vẽ bằng một con rùa. Ta không biết tên của con rùa này nhưng rõ ràng nó là một đối tượng theo đúng nghĩa đen (nếu giàu trí tưởng tượng, bạn có thể xem nó giống con rùa thật). Hoàn toàn khác với các con số “vô tri vô giác”, con rùa của ta có *những đặc tính của một sinh vật thật: có trạng thái, có hành vi và có tương tác với môi trường*. Trong phần này, ta tiếp tục dùng con rùa để hiểu rõ về đối tượng trong Python; hơn nữa, ta không chỉ dùng 1 mà là 5 con rùa. Thử chương trình minh họa sau:

https://github.com/vqhBook/python/blob/master/lesson10/turtle_race.py

```

1 import turtle as t
2 from turtle import window_width as ww, window_height as wh
3 from random import randint as rint
4
5 def random_point(r):
6     x = rint(-ww()//2 + r, ww()//2 - r)
7     y = rint(-wh()//2 + r, wh()/2 - r)
8     return (x, y)
9
10 def check_winner():
11     for tur in FOUR_TURTLES:
12         if Pizza.distance(tur) < Pizza.radius/2:
13             print(tur.name + " win the race!")
14             tur.turtlesize(2.5, 2.5)
15             return True
16     return False
17
```

⁵Giá trị `nan` thường được dùng làm kết quả không hợp lệ hay không xác định của các tính toán trên số. Ta đã gặp nó ở Phần 9.4.

```

18 def forward_and_reflex():
19     for tur in FOUR_TURTLES:
20         if abs(tur.xcor()) >= ww()/2 - tur.radius:
21             tur.setheading((180 - tur.heading()) % 360)
22         if abs(tur.ycor()) >= wh()/2 - tur.radius:
23             tur.setheading((360 - tur.heading()) % 360)
24         tur.forward(2)
25
26 def check_collision():
27     for tur in FOUR_TURTLES:
28         for other_tur in FOUR_TURTLES:
29             if (tur is not other_tur and
30                 tur.distance(other_tur) < 2*tur.radius):
31                 tur.setheading(rint(0, 360))
32
33 def run():
34     if check_winner():
35         t.update()
36         return
37     forward_and_reflex(); check_collision()
38     t.update(); t.ontimer(run, 10)
39
40 t.title("Teenage Mutant Ninja Turtles Race for Pizza!")
41 t.addshape("pizza.gif"); t.tracer(False)
42
43 Pizza = t.Turtle()
44 Pizza.shape("pizza.gif"); Pizza.radius = 20
45 Pizza.up(); Pizza.goto(random_point(Pizza.radius))
46 Pizza.ondrag(Pizza.goto)
47
48 Leo, Mike, Don, Raph = t.Turtle(), t.Turtle(), t.Turtle(),
49     t.Turtle()
50 for tur, name, color in [(Leo, "Leonardo", "blue"),
51                         (Mike, "Michelangelo", "orange"),
52                         (Don, "Donatello", "purple"),
53                         (Raph, "Raphael", "red")]:
54     tur.shape("turtle"); tur.radius = 10
55     tur.name = name; tur.color(color, color)
56     tur.setheading(rint(0, 360))
57     tur.up(); tur.goto(random_point(tur.radius)); tur.down()
58 FOUR_TURTLES = [Leo, Mike, Don, Raph]
59 t.onkeypress(t.bye); t.listen()
60 run(); t.mainloop()

```

Chương trình trên mô phỏng cuộc đua giành pizza của 4 thiếu niên ninja rùa đột biến là Leonardo (Leo), Michelangelo (Mike), Donatello (Don) và Raphael (Raph).⁶ Ai “lượm được” miếng pizza trước là người chiến thắng.⁷ Bạn có thể kéo miếng pizza (đặt chuột vào miếng pizza, nhấp giữ, rê chuột và thả) và co giãn cửa sổ để cuộc đua thêm phần kịch tính. Các chú ninja rùa này sẽ được “mổ xẻ”, phân tích ở các phần dưới đây.

10.3 Danh tính, kiểu và giá trị

Trong vũ trụ Python, mọi đối tượng đều có 3 đặc trưng quan trọng: **danh tính** (identity), **kiểu** (type) và **giá trị** (value). Các đối tượng có thể được tham chiếu bằng **tên** (name) hay **định danh** (identifier). Hơn nữa, có thể có nhiều tên tham chiếu đến cùng một đối tượng.

Danh tính giúp phân biệt đối tượng: các đối tượng khác nhau có danh tính khác nhau. Hàm `id` trả về số định danh của đối tượng, đó là một con số nguyên xác định danh tính của đối tượng (ta có thể hình dung là “số chứng minh đối tượng”). Toán tử so sánh `is` giúp kiểm tra xem 2 đối tượng có cùng danh tính không (nghĩa là cùng đối tượng). Toán tử `is not` là “phủ định” của `is` ($x \text{ is not } y$ nghĩa là $\text{not } x \text{ is } y$). Hai toán tử này được gọi chung là toán tử **kiểm tra danh tính** (identity test) và được xếp chung nhóm với các toán tử so sánh (cùng độ ưu tiên).

Kiểu cho biết loại/dạng/nhóm đối tượng. Trường hợp đối tượng là dữ liệu ta gọi nó là kiểu dữ liệu như đã biết; tổng quát hơn, ta gọi kiểu đối tượng là **lớp** (class). Lớp (kiểu đối tượng) xác định khuôn dạng, cách thức tổ chức và xử lý đối tượng. Hàm `type` trả về kiểu (tức là lớp) của đối tượng.

Giá trị của đối tượng là “bản thân đối tượng”. Thuật ngữ này cũng mơ hồ như cách nó hay được dùng trong đời sống thông thường.⁸ Trường hợp **đối tượng đơn** (simple object) như số nguyên, số thực, luận lý, `None` thì điều này khá rõ ràng vì bản thân dữ liệu là toàn bộ đối tượng; chẳng hạn, giá trị của số nguyên **10** chính là số 10, giá trị của số thực **10.0** cũng chính là số 10, ... Thử minh họa sau:

```
1 >>> a = 10; b = 10.0
2 >>> print(a == b, a is b)
True False
3 >>> print(id(a), type(a), id(b), type(b))
1731131456 <class 'int'> 55815968 <class 'float'>
4 >>> a = "10"; print(a, type(a))
```

⁶Đây là 4 nhân vật trong các loạt truyện tranh, phim, game, ... “Teenage Mutant Ninja Turtles” nổi tiếng. Bạn có thể xem thêm thông tin tại https://en.wikipedia.org/wiki/Teenage_Mutant_Ninja_Turtles.

⁷Hình ảnh miếng pizza, tập tin `pizza.gif`, được download tại <https://wanelo.co/p/232589/teenage-mutant-ninja-turtles-pizza-time-clock> và chỉnh transparent. Bạn download tập tin tại <https://github.com/vqhBook/python/blob/master/lesson10/pizza.gif> và đặt tập tin chung thư mục với tập tin mã nguồn.

⁸Thử trả lời câu hỏi: giá trị của bản thân bạn là gì?

```
10 <class 'str'>
```

Dấu == so sánh về ngữ nghĩa (tức giá trị) nên 10 và 10.0 được xem là bằng nhau, tức là cùng giá trị. Điều này hoàn toàn hợp lý như ta đã biết trong Toán. Tuy nhiên, 10 có kiểu và danh tính (hay số định danh) khác với 10.0 như kết xuất của Dòng 3 cho thấy.⁹

Ta phải *phân biệt tên và đối tượng*: thứ mà Python làm việc là đối tượng, tên chỉ giúp truy cập đến đối tượng và có thể được định nghĩa lại để trỏ đến đối tượng khác. Dĩ nhiên, kiểu là của đối tượng chứ không phải tên: câu hỏi `type(a)` là “kiểu của đối tượng mà a trỏ đến?” chứ không phải là “kiểu của tên a?”. Vì lý do này mà Python thường được gọi là ngôn ngữ **kiểu động** (dynamic typing) để phân biệt với các ngôn ngữ **kiểu tĩnh** (static typing) như C/C++.¹⁰

Trường hợp **đối tượng phức** (complex/compound object), tức là đối tượng có các thành phần bên trong (đối tượng chứa các đối tượng khác hay đối tượng ghép từ các đối tượng khác) như chuỗi (gồm nhiều kí tự) hay con rùa (gồm nhiều thông tin như vị trí, hướng, màu sắc, ...) thì giá trị là “kết hợp” giá trị của các thành phần. Chạy chương trình `turtle_race.py` ở trên để có các chú rùa và tương tác tiếp trong phiên làm việc như sau:

```
===== RESTART: D:\Python\lesson10\turtle_race.py =====
Leonardo win the race!

1 >>> print(Leo, type(Leo))
<turtle.Turtle object at 0x03CC8690> <class 'turtle.Turtle'>
2 >>> print(id(Leo), id(Mike))
63735440 63735536
3 >>> print(Leo is Mike, Leo == Mike)
False False
4 >>> type(Pizza)
<class 'turtle.Turtle'>
```

Các chú rùa là đối tượng của lớp (kiểu) `turtle.Turtle`, tức là lớp `Turtle` của module `turtle`. Đây là kết quả của việc tạo đối tượng ở Dòng 48 trong mã `turtle_race.py`. Lưu ý rằng miếng pizza cũng là ... rùa như kết quả của Dòng 4 trên cho thấy. Thật vậy, Pizza được tạo như một con rùa ở Dòng 43, chỉ khác là nó có hình dạng của một miếng pizza (Dòng 44) chứ không phải hình dạng con rùa như 4 chú ninja rùa (Dòng 53). Mặc dù điều này không hợp lý lắm (ta sẽ giải quyết ở Bài 14) nhưng nó giúp chương trình đơn giản hơn (như việc ta có thể vẽ, kéo, kiểm tra vị trí, ... dễ dàng khi Pizza là một con rùa!).

⁹Lưu ý là số định danh chỉ có ý nghĩa lúc Python thực thi và sẽ thay đổi trong các phiên làm việc Python khác nhau. Do đó các số định danh trên (kết quả trả về của hàm `id`) thường không giống với kết quả mà bạn chạy.

¹⁰Trong các ngôn ngữ kiểu tĩnh, tên là “hộp chứa đối tượng” nên ta cần khai báo trước kiểu cho tên và sau đó tên chỉ có thể chứa các đối tượng của kiểu đó (có thể đổi đối tượng nhưng không đổi kiểu). Tương tự như “hộp kẹo” chỉ chứa kẹo hay “chuồng bò” chỉ chứa bò:)

Cũng lưu ý, tên gọi của 4 ninja rùa là khác nhau (Dòng 49-52 với các tên gọi lần lượt là Leonardo, Michelangelo, Donatello, Raphael) nhưng điều này không quyết định đến danh tính “thực sự” của chúng. Chúng thực sự là 4 chú rùa khác nhau dù bạn có đặt chung tên gọi đi nữa.

10.4 Bất biến và khả biến

Danh tính và kiểu là bản chất của đối tượng, nó được gán với đối tượng từ lúc sinh ra và không thể thay đổi. Giá trị là “chính đối tượng”, nó có thể thay đổi trong chu trình sống của đối tượng. Điều này tương tự việc bạn sinh ra đã được gán sẵn một “mã số” (ông Trời quản lý:) và là người. Việc bạn giàu hay nghèo, đẹp hay xấu, nam hay nữ, ... đều có thể thay đổi được do chúng thuộc về giá trị của bạn:) Thử minh họa sau:

```

1 >>> a = 10
2 >>> print(a, type(a), id(a))
  10 <class 'int'> 1694524656
3 >>> a = a + 1
4 >>> print(a, type(a), id(a))
  11 <class 'int'> 1694524672
5 >>> s = "hello"
6 >>> print(s, type(s))
  'hello' <class 'str'>
7 >>> s[0] = "H"
  ...
  TypeError: 'str' object does not support item assignment
8 >>> li = list("hello")
9 >>> print(li, type(li), id(li))
  ['h', 'e', 'l', 'l', 'o'] <class 'list'> 8257784
10 >>> li[0] = "H"
11 >>> print(li, id(li))
  ['H', 'e', 'l', 'l', 'o'] 8257784

```

Phụ thuộc vào kiểu, có những đối tượng không thay đổi được giá trị, gọi là **bất biến** (immutable), và có những đối tượng có thể thay đổi được giá trị, gọi là **khả biến** (mutable).¹¹ Các dữ liệu cơ bản (số nguyên, số thực, chuỗi, luận lý, None) là bất biến. Chẳng hạn, sau Dòng 1 trên thì đối tượng 10 (số nguyên) được tạo ra và được tham chiếu đến bởi biến a. Sau đó, giá trị, kiểu và số định danh của nó (đối tượng 10) được xuất ra ở Dòng 2. Lưu ý là biến (tên) a chỉ giúp ta tham chiếu đến đối tượng 10 chứ không phải là bản thân đối tượng.¹²

¹¹Các thuật ngữ Tiếng Việt này nghe hơi “tiên hiệp” nhưng có lẽ là thuật ngữ phù hợp nhất.

¹²Có sự lạm dụng ngôn “nhẹ” ở đây: ta thường nói là giá trị, kiểu và số định danh của biến nhưng đúng ra phải là của đối tượng đang được biến tham chiếu.

Ở Dòng 3, có một đối tượng mới được tạo ra là số nguyên 11 (kết quả của $10 + 1$) và `a` tham chiếu đến đối tượng mới này như số định danh của nó cho thấy (kết xuất của Dòng 4). Lưu ý rằng kiểu, định danh và giá trị của đối tượng 10 không thay đổi, nhưng `a` đã tham chiếu đến một đối tượng mới có giá trị khác (11), cùng kiểu (`int`) và dĩ nhiên là khác định danh.

Mặc dù chuỗi không là đối tượng đơn nhưng nó cũng bất biến. Sau khi chuỗi được tạo ra, ta không thể thay đổi được các kí tự của nó (tức không thay đổi được giá trị). Chẳng hạn, việc đổi kí tự đầu tiên của chuỗi "hello" (thành kí tự H) sẽ phát sinh lỗi thực thi (như kết xuất của Dòng 7 cho thấy). Bạn cũng lưu ý là chuỗi "hello" bất biến nhưng biến `s` có thể được cho tham chiếu đến chuỗi mới như chuỗi "Hello" (chẳng hạn, bằng lệnh gán `s = "Hello"`).

Trừ các kiểu dữ liệu cơ bản thì các đối tượng của đa số kiểu khác là khả biến. Chẳng hạn, **danh sách** (`list`), là kiểu gồm dãy các phần tử tương tự như chuỗi nhưng khả biến. Như kết xuất của Dòng 9 cho thấy, sau Dòng 8, ta có một danh sách các kí tự `h`, `e`, `l`, `l`, `o` (tương tự dãy kí tự của chuỗi `hello`) được tham chiếu đến bởi biến `li`. Khác với chuỗi, ta có thể đổi kí tự đầu tiên của danh sách này thành kí tự mới `H` ở Dòng 10. Nhận xét rằng, ta chỉ có một đối tượng danh sách nhưng bản thân đối tượng đó đã thay đổi giá trị như kết xuất của Dòng 11 cho thấy.

Các chú rùa của ta, đại diện cho một nhóm đối tượng “cao cấp”, các sinh vật, hiển nhiên là điển hình của “giới” khả biến.¹³ Thật vậy, sau khi được “khai sinh”, Leo và các ninja khác (kể cả miếng pizza) liên tục thay đổi giá trị. Cụ thể, chúng thay đổi vị trí, hướng di chuyển và các thứ khác.

Bạn phải phân biệt rõ ràng ý nghĩa của so sánh `==` với `is` (cũng như `!=` với `is not`), nhất là trên các đối tượng khả biến. Hiển nhiên, khi 2 đối tượng cùng danh tính (`is`) thì chúng chỉ là 1 nên chúng phải bằng nhau về giá trị (`==`). Thử minh họa sau để rõ hơn:

```

1 >>> x = y = [1, 2, 3]; z = [1, 2, 3]
2 >>> print(id(x), id(y), id(z))
62012424 62012424 62012520
3 >>> print(x == y == z, x is y is z)
True False
4 >>> del x[1]
5 >>> print(x, y, z, y is x, y == z)
[1, 3] [1, 3] [1, 2, 3] True False

```

`x`, `y` cùng danh tính (nghĩa là cùng tham chiếu đến 1 đối tượng), `z` khác danh tính (trỏ đến đối tượng khác) nhưng cùng giá trị như `x`, `y` (cùng là danh sách gồm 3 phần tử theo thứ tự là 1, 2, 3). Sau khi xóa phần tử `x[1]` thì `x` thay đổi giá trị (nhưng không thay đổi danh tính), đúng ra là phần tử nó trỏ đến thay đổi giá trị, và

¹³Lại “tiên hiệp” chút xíu: ta có thể nói rằng vũ trụ Python chia ra 2 giới, bất biến và khả biến. Bất biến bao gồm các đối tượng không sống (như số nguyên, số thực) còn khả biến bao gồm các sinh vật (như con rùa).

đó cũng là đối tượng mà *y* trở đến. Khi này, *y* (cũng là *x*) không còn cùng giá trị với *z*.

10.5 Phương thức và trường

Trong mã `turtle_race.py` (Phần 10.2), ta đã dùng một thứ rất quan trọng gọi là phương thức. **Phương thức** (method) là hàm của đối tượng (hàm “gắn” với đối tượng) được gọi với cú pháp:

`<Obj>.<Method>(<Args>)`

Khác với cách gọi hàm thông thường, ta đặt tên đối tượng (`<Obj>`) và toán tử “chấm” (`.`) ngay trước tên phương thức (`<Method>`) để ám chỉ “của đối tượng” (ta sẽ rõ hơn về toán tử này ở Phần 10.8).

Ý định của lời gọi hàm thông thường là “ta làm gì đó trên đối tượng” còn với lời gọi phương thức, “ta yêu cầu đối tượng làm gì đó” hay thậm chí là “ta nhờ đối tượng làm gì đó”. Với dữ liệu thông thường thì hàm là phù hợp, chẳng hạn, “ta bình phương một số nguyên”. Với đối tượng phức tạp thì phương thức phù hợp hơn, chẳng hạn, “ta bấm nút bật để tivi mở lên”: hành động này xuất phát từ ta nhưng kết thúc ở cái tivi, rõ ràng, cuối cùng thì cái tivi tự nó mở (theo cách của nó), ta chỉ bấm nút để yêu cầu hay **“truyền thông điệp”** (message passing) mở mà thôi.

Như đã nói, các chú rùa là đại diện điển hình của các đối tượng.¹⁴ Chúng có trạng thái, hành vi và tương tác với môi trường. Lưu ý, mặc dù đều là rùa nhưng hoạt động cụ thể của từng con là khác nhau: chúng là các **thể hiện** (instance) khác nhau của cùng lớp `turtle.Turtle`. Đến đây bạn đã chấp nhận Pizza là một con rùa chưa nào?!

Rõ ràng, để hoạt động, các chú rùa cần nhớ nhiều thứ như vị trí trên của sổ, màu đường, màu tô, hướng di chuyển, ... Các thứ này được gọi là thuộc tính hay **trường** (field) của đối tượng. Các trường này nhận giá trị khởi động lúc đối tượng mới được tạo ra và thay đổi giá trị trong quá trình đối tượng hoạt động. Giá trị của các trường được gọi là **trạng thái** (state) của đối tượng. Phương thức chính là cách ta tương tác với các đối tượng mà theo đó chúng sẽ thực hiện **hành vi** (behavior) và thay đổi trạng thái tương ứng.

Dạng phương thức đơn giản nhất giúp ta truy vấn (hỏi) trạng thái của đối tượng, chúng trả về giá trị hiện tại của trường và thường được gọi là **getter**.¹⁵ Con rùa có các getter là: `xcor` cho biết tọa độ *x*, `ycor` cho biết tọa độ *y* hay gọn hơn, `position` trả về cặp (*x*, *y*); `pencolor` cho biết màu đường, `fillcolor` cho biết màu tô hay gọn hơn, `color` trả về cặp màu; `heading` cho biết hướng; `isvisible` cho biết trạng thái hiện/ẩn; `isdown` cho biết trạng thái bút đặt/nhấc; ...

Dạng phương thức thứ 2 giúp đặt (thay đổi) giá trị trạng thái của đối tượng, thường được gọi là **setter**.¹⁶ Con rùa có các setter là: `pencolor` đặt màu đường,

¹⁴Các con số và mọi đối tượng khác cũng vậy thôi, chỉ là với con rùa, ta dễ hình dung hơn.

¹⁵Các getter thường được đặt tên theo dạng `get_xyz`, nhưng lớp `turtle.Turtle` không theo qui ước này.

¹⁶Các setter thường đi kèm với getter và được đặt tên theo qui ước `set_xyz`.

`fillcolor` đặt màu tô hay gọn hơn, `color` đặt cặp màu; `setheading` đặt hướng; `showturtle/hideturtle` để hiện/ẩn; `down/up` đặt đặt/nhấc bút; ... Cũng lưu ý, đôi khi getter và setter cùng là một phương thức với qui ước: không đối số là getter (hỏi), có đối số là setter (đặt) như `color`, `pencolor`, `fillcolor`, `width`, `shape`, `speed` của con rùa.¹⁷

Dạng phương thức thức tổng quát yêu cầu đối tượng làm gì đó dựa trên trạng thái hiện tại của mình và cập nhật trạng thái mới. Thường thì thuật ngữ setter dùng để chỉ các phương thức “chỉ thay đổi trạng thái mà không làm gì thêm nữa”, tuy nhiên ranh giới cũng không rõ ràng. Chẳng hạn, phương thức `end_fill` không chỉ tắt chế độ tô mà còn thực hiện việc tô màu các hình trước đó (sau `begin_fill`). Nhiều phương thức của con rùa là ở dạng này như: `forward/backward`, `goto` (hay `setposition`) thay đổi vị trí và vẽ theo đó; `circle` thay đổi cả vị trí, hướng và vẽ theo đó; ...

Cũng có vài dạng phương thức đặc biệt: phương thức “chỉ cập nhật trạng thái” (các setter thường chỉ đặt giá trị mới) như `left/right` thay đổi hướng dựa trên hướng hiện tại; phương thức “chỉ làm mà không thay đổi trạng thái” như `dot` vẽ chấm điểm, `write` vẽ chuỗi; phương thức “tương tác với các đối tượng khác” như `distance` tính khoảng cách, `towards` tính góc đến con rùa khác.

Đặc biệt, phương thức `ondrag` của con rùa cho phép đăng kí trình xử lý sự kiện nhấp-rê chuột trên con rùa đó. Trình xử lý sự kiện này có 2 tham số để nhận tọa độ x, y của chuột. Ở đây, Dòng 46 đăng kí phương thức `goto` của `Pizza` làm trình xử lý sự kiện nhấp-rê chuột trên `Pizza`. Khác với hàm, nhờ là phương thức của đối tượng nên ở đây, ta chỉ có thể nhấp-rê miếng pizza mà khi đó chỉ miếng pizza thay đổi vị trí theo. Các con rùa khác không hề dính líu!

Ngoài ra, trong mã `turtle_race.py`, ta đã tự mình tạo thêm trường cho con rùa để “gắn thêm” dữ liệu cho chúng. Cụ thể, ta đã gắn thêm các trường `radius` và `name` vào các con rùa. Lưu ý, *cũng như phương thức, trường là của riêng đối tượng: dữ liệu gắn trên đối tượng này không ảnh hưởng đến dữ liệu gắn trên đối tượng khác*. Chẳng hạn, cùng tên trường `name` nhưng mỗi con rùa có tên khác nhau. Theo nghĩa này, phương thức có thể được gọi là hàm thể hiện còn trường có thể được gọi là **biến thể hiện** (instance variable) và cả 2 đều là **thành phần** (member) của đối tượng.

Ta đã thấy phương thức cũng là hàm (hàm gắn với đối tượng). Ở Phần 3.1 và Bài tập 3.10 ta “cảm thấy” toán tử cũng là hàm. Đúng hơn, chúng là các phương thức! Thử minh họa sau:

```
1 >>> x = 10; y = "hi"
2 >>> print(x * 5, y * 5)
50 'hihihihihi'
3 >>> print(x.__mul__(5), y.__mul__(5))
50 'hihihihihi'
```

Khi ta dùng toán tử `*` giữa 2 đối tượng, Python gọi phương thức đặc biệt

¹⁷Các phương thức này thường nhận một đối số với giá trị mặc định là `None`.

`__mul__` trên đối tượng là toán hạng thứ nhất. Nói cách khác: $a * b$ chỉ là cách viết đẹp của $a.__mul__(b)$ mà thôi! Ta sẽ học kĩ hơn trong Phần 14.5 về bí ẩn thú vị này.

10.6 Không gian tên

Mọi đối tượng trong Python đều được để trên **bộ nhớ** (memory) của chương trình và được truy cập đến bằng tên. Python quản lý các tên này bằng một hệ thống gọi là **không gian tên** (namespace).¹⁸ Cái gọi là “bộ nhớ” trong Phần 3.4 thực ra là không gian tên mà ta có thể kiểm tra bằng hàm `dir`.¹⁹ Thử minh họa sau:

```

1  >>> "hello"
    'hello'
2  >>> a = 10
3  >>> def b(x): return x*2
4
5  >>> c = lambda x: x**2
6  >>> dir()
    ['__annotations__', '__builtins__', '__doc__', '__loader__',
    '__name__', '__package__', '__spec__', 'a', 'b', 'c']
7  >>> c(b(a))
    400
8  >>> c = b = a
9  >>> print(c, b, a)
    10 10 10
10 >>> del a; a
    ...
    NameError: name 'a' is not defined
11 >>> dir()
    ['__annotations__', '__builtins__', '__doc__', '__loader__',
    '__name__', '__package__', '__spec__', 'b', 'c']

```

Sau đây là các phân tích:

- Dòng 1: Python tạo chuỗi hello (trên bộ nhớ) và xuất giá trị của nó. Sau đó thì đối tượng này không còn được dùng nữa, nó trở thành “**rác**” (garbage).
- Dòng 2: số nguyên 10 được tạo (trên bộ nhớ) và tham chiếu đến nó được đặt vào tên a mà ta hay gọi là biến a. Tên này được tạo và được đặt vào không gian tên.

¹⁸Ý nghĩa tương tự như các loại danh bạ. Thuật ngữ kĩ thuật sát nghĩa hơn là “bảng các tên” hay **bảng danh biểu** (symbol table).

¹⁹Thật ra việc quản lý không gian tên là việc “nội bộ” của Python mà hàm `dir` chỉ cho thấy phần nào.

- Dòng 3: tạo hàm “gấp đôi” (hàm nhận 1 đối số, trả về 2 lần đối số) và cho tên `b` tham chiếu đến nó mà ta hay gọi là hàm `b`. Lưu ý, `def` được dùng để định nghĩa hàm, như vậy, có thể được xem là một lệnh (mà ngữ nghĩa tương tự lệnh gán).
- Dòng 5: tạo hàm “bình phương” (hàm nhận 1 đối số, trả về bình phương đối số) và cho tên `c` tham chiếu đến nó mà ta phân vân không biết gọi `c` là gì?! Gọi là hàm vì mặc dù dùng lệnh gán (với biểu thức `lambda`) nhưng cái mà `c` tham chiếu đến là hàm.
- Dòng 6: cho thấy không gian tên (lúc này) gồm các tên `a`, `b`, `c` (cùng với các tên đặc biệt khác).
- Dòng 7: truy cập (dùng) 3 đối tượng đã tạo trước đó (số nguyên 10, hàm gấp đôi, hàm bình phương) nhờ các tên tham chiếu tương ứng.
- Dòng 8: đặt lại các tên `b`, `c` cùng tham chiếu đến đối tượng mà `a` tham chiếu (số nguyên 10). Quan trọng, sau Dòng 8, hàm gấp đôi và hàm bình phương cũng trở thành rác mà Python sẽ tự động “dọn rác” (tức là hủy) nhờ **“bộ gom rác”** (garbage collector) (tương tự Dòng 1 và tình huống ở Phần 2.2).²⁰
- Dòng 9: cho thấy `a`, `b`, `c` lúc này đang cùng tham chiếu đến đối tượng số nguyên 10.
- Dòng 10: xóa tên `a`, nghĩa là gỡ `a` khỏi không gian tên, và do đó “không còn” tên `a` khi dùng đến. Lưu ý, lệnh `del` chỉ xóa tên chứ không xóa đối tượng.
- Dòng 11: sau Dòng 10 thì `b`, `c` vẫn còn trong không gian tên và cùng tham chiếu đến đối tượng số nguyên 10.

Tóm lại, tương tự như danh bạ điện thoại (quản lý các tên-số điện thoại), Python dùng không gian tên để quản lý tên tham chiếu đến các đối tượng:

- Việc định nghĩa tên (lệnh gán định nghĩa biến, lệnh `def` định nghĩa hàm, lệnh `import` nạp module, ...) sẽ tạo tên mới hoặc cập nhật tên cũ nếu tên đã có (thêm tên-số hoặc sửa số cho tên đã có trong danh bạ).²¹
- Nhiều tên khác nhau có thể cùng tham chiếu đến một đối tượng (nhiều tên cùng số trong danh bạ) và khi một đối tượng không còn được tham chiếu đến thì nó trở thành rác và bị dọn (số không có tên trong danh bạ thì sẽ bị xóa).
- Việc dùng tên (trong biểu thức, gọi hàm, truy cập hàm trong module, ...) sẽ dùng đối tượng được tham chiếu tương ứng nếu có (tra danh bạ) hoặc báo lỗi (`NameError`) nếu không (thông báo tên không có trong danh bạ).²²

²⁰Mặc dù hơi buồn cười nhưng đây là các thuật ngữ chính thống. Bạn cũng nên biết rằng việc tự mình tạo và giải phóng các đối tượng khi không cần dùng là không hề đơn giản. Đây là “nỗi thống khổ” của nhiều lập trình viên trên các ngôn ngữ như C/C++ (bạn cứ thử tưởng tượng việc tự mình dọn rác sẽ rõ). Python (và một số ngôn ngữ khác) đã giải phóng lập trình viên khỏi việc này.

²¹Việc quản lý danh bạ điện thoại có khác vì cho phép nhiều mục cùng tên.

²²Chỗ này hổng thật sự giống danh bạ, vì ta có thể bấm các số không có trong danh bạ, nhưng khá nguy hiểm vì đó là “số lạ”.)

10.7 Phạm vi tên

Mô tả không gian tên ở trên khá đơn giản và dễ hiểu vì ta chỉ đề cập đến 1 không gian tên là **không gian tên toàn cục** (global namespace) chứa các tên được định nghĩa trong chương trình (hay phiên tương tác). Trường hợp có các hàm thì lời gọi hàm sẽ tạo thêm không gian tên gọi là **không gian tên cục bộ** (local namespace) chứa các tên được định nghĩa trong hàm như ta sẽ thấy. Ngoài ra, trước khi chạy chương trình (hay phiên tương tác), Python tạo **không gian tên dựng sẵn** (built-in namespace) chứa tên các hàm dựng sẵn, các hằng, kiểu dựng sẵn, ... như ta đã thấy ở Phần 3.4.

Vì có nhiều không gian tên nên Python phải có qui tắc xác định không gian tên nào được dùng cho tên nào (khi tra, thêm, xóa, sửa tên). Qui tắc này dựa trên **phạm vi** (scope) của tên, tức là “khung mã” chứa tên. Ta dùng chương trình minh họa sau để hiểu rõ hơn (mã ở <https://github.com/vqhBook/python/blob/master/lesson10/scope.py>):

```

1 def func1(d):
2     # global a
3     a = d
4     print(a)
5
6 def func2(d):
7     def func3():
8         # nonlocal a
9         a = 2 * d
10        print(a)
11
12    a = d
13    func3()
14    print(a)
15
16 a = 0
17 func1(1)
18 print(a)
19 func2(2)
20 print(a)
21 # func3()
  
```

Dựa trên mã của chương trình, ta chia nó thành các khung mã phân cấp như sau: khung lớn nhất là toàn bộ chương trình; khung này chứa khung mã của hàm `func1`, `func2`; đặc biệt, khung mã của hàm `func2` chứa khung mã của hàm `func3` bên trong. Ta nói khung ngoài cùng có **mức** (level) là 1, hai khung trong đó có mức là 2 và khung trong cùng có mức là 3. Ta chưa thấy việc định nghĩa “hàm trong hàm”, hay **hàm lồng** (nested function), ở các bài trước nhưng điều này hoàn toàn bình thường trong Python (Bài tập 10.2 cho thấy một ứng dụng của kỹ thuật này).

Theo cấu trúc phân cấp các khung mã, chương trình trên gồm 4 phạm vi “từ trong ra ngoài” là: cục bộ, hàm bao ngoài, toàn cục và dựng sẵn (mức 0).²³ Lưu ý, mỗi module có không gian tên toàn cục riêng nên thuật ngữ đúng hơn cho phạm vi toàn cục là **phạm vi module** (module scope). Nếu hàm lồng nhiều mức thì phạm vi được tính từ trong ra ngoài theo mức lồng.

Mặc dù phạm vi được xác định dựa trên cấu trúc mã (mức phân cấp các khung mã) nhưng các không gian tên được tạo ra lúc chạy: không chạy chẳng có không gian tên nào, chạy chương trình có không gian tên toàn cục (và không gian tên dựng sẵn bên ngoài), chạy vào hàm có thêm không gian tên cục bộ, vào hàm trong nữa có thêm không gian tên cục bộ nữa (mà không gian tên trước đó trở thành không gian tên bao ngoài), khi hàm kết thúc thì không gian tên cục bộ của nó bị gỡ bỏ, ...

Sau đây, tôi đóng vai Python “chạy từng bước” chương trình trên. Bạn theo dõi và đối chiếu kết quả để nắm rõ:²⁴

- Trước khi chạy chương trình, Python tạo không gian tên dựng sẵn B mức 0. Sau đó Python tạo không gian tên toàn cục (module) G mức 1.
- Lệnh `def` ở Dòng 1-4 tạo một hàm và thêm tên `func1` tham chiếu đến hàm đó vào không gian tên hiện tại (G). Lưu ý, lệnh này chỉ định nghĩa hàm (tạo và cho tên tham chiếu đến) chứ không thực thi hàm.
- Tương tự cho tên `func2` với hàm tạo bởi lệnh `def` ở Dòng 6-14.
- Lệnh gán ở Dòng 16 thêm tên `a` tham chiếu đến số nguyên 0 vào G .
- Lời gọi hàm ở Dòng 17 thực thi `func1` với đối số là 1:
 - Trước khi chạy `func1`, Python tạo không gian tên cục bộ L_1 mức 2. Sau đó Python thêm tên tham số `d` vào L_1 và cho tên này tham chiếu đến đối số 1 (như vậy, tham số cũng nằm trong không gian tên cục bộ).
 - Lệnh gán ở Dòng 3 thêm tên `a` tham chiếu đến cùng đối tượng như `d` (số nguyên 1) vào không gian tên hiện tại (L_1). Lưu ý là tên `a` này nằm trong L_1 và cũng có một tên `a` nữa (tham chiếu đến 0) nằm trong G .
 - Để thực hiện lệnh xuất ở Dòng 4, Python cần tra 2 cái tên: `print`, `a`. Để thực hiện quá trình tra tên, Python tìm từ không gian tên hiện tại (L_1) dần theo mức xuống không gian tên cuối cùng (B). Cụ thể, `a` có trong L_1 (mức 2), `print` không có trong L_1 (mức 2), G (mức 1) nhưng có trong B (mức 0), và tên này tham chiếu đến hàm xuất dựng sẵn. Kết quả xuất ra là 1. Nếu bạn gõ nhầm tên `print` thành `pint` thì khi đó sẽ có lỗi `NameError` vì tên `pint` không có trong không gian tên nào cả.
 - Hàm `func1` kết thúc, Python gỡ bỏ không gian tên cục bộ của hàm này (L_1) và đặt lại không gian tên hiện tại là G (mức trước đó, mức 1).
- Python chạy tiếp Lệnh xuất ở Dòng 18 với kết quả xuất ra là 0, là đối tượng tham chiếu bởi tên `a` trong G .
- Tương tự cho lời gọi hàm `func2(2)` ở Dòng 19:

²³4 phạm vi này thường được gọi là **LEGB**, viết tắt của Local, Enclosing, Global và Built-in.

²⁴Có thể nằm coi vì nó khá dài:)

- Lưu ý, khi chạy `func2` thì lệnh `def` ở Dòng 7-10 tạo hàm (không gọi hàm) và thêm tên `func3` tham chiếu đến hàm đó vào L_2 là không gian tên cục bộ của `func2`.
- Tương tự cho lệnh gán ở Dòng 12.
- Lỗi gọi hàm `func3()` thực thi hàm `func3`:
 - * Tương tự, trước khi thực thi hàm, Python tạo không gian tên mới L_3 là không gian tên cục bộ của `func3` (mức 3).
 - * Lệnh gán ở Dòng 9 trước hết tra tên `d` để tính $2 * d$, tên này không có trong L_3 mà có ở L_2 , cụ thể, là tham số `d` của `func2` tham chiếu đến giá trị 2.
 - * Dòng 10 xuất ra 4.
 - * Hàm `func3` kết thúc, Python gỡ bỏ L_3 và đặt lại không gian tên hiện tại là L_2 .
- Dòng 14 xuất ra 2, là giá trị của `a` trong L_2 .
- Kết thúc hàm `func2` tương tự.

- Dòng 20 xuất ra 0 là giá trị của `a` trong G .

Ôi! Việc mô tả từng bước như thế này thực sự ... tốn giấy! Một cách đỡ tốn kém và trực quan hơn là dùng kĩ thuật debug ở Bài tập 10.5. Tóm lại vài lưu ý đặc biệt như sau:

- Các biến định nghĩa trong hàm thường được gọi là **biến cục bộ** (local variable) vì chúng chỉ có thể được dùng bên trong hàm (khi hàm kết thúc thì không gian tên cục bộ của nó cũng bị gỡ bỏ). Ta hoàn toàn yên tâm dùng tên ta muốn làm tên cục bộ mà không sợ trùng các tên khác. Đây là mục đích đầu tiên để Python đưa ra cơ chế quản lý các không gian tên phức tạp như vậy: giải quyết vấn đề đụng độ tên giữa các hàm.
- Tham số là dạng tên cục bộ đặc biệt (tham chiếu đến đối số và được đặt vào không gian tên cục bộ trước khi thân hàm được thực thi).
- Các tên ở mức trong (mức cao) sẽ **che** (hiding) các tên trùng ở mức ngoài (mức thấp). Theo nghĩa đó, các tên cục bộ sẽ có “độ ưu tiên” cao nhất và các tên dựng sẵn là thấp nhất.

Bạn có thấy các ghi chú trong chương trình trên không. Điều gì xảy ra nếu ta bỏ các ghi chú này. Trước hết, **lệnh global** (global statement) với cú pháp:

global <Names>

khai báo cho Python biết sẽ dùng không gian tên toàn cục cho các tên được liệt kê (<Names>) hoặc **lệnh nonlocal** (nonlocal statement) với cú pháp tương tự, khai báo dùng không gian tên cục bộ của **hàm bao ngoài** (enclosing function) hàm chứa nó (không gian tên có mức nhỏ hơn 1 so với không gian tên hiện tại). Rõ ràng các lệnh này ảnh hưởng đến việc tra tên vì nếu không có chúng, Python sẽ tra từ không gian tên hiện tại. Như vậy:

- Nếu bỏ dấu ghi chú ở Dòng 2 (tức là dùng lệnh `global a`) thì biến `a` trong hàm `func1` sẽ trở thành biến toàn cục (được để ở không gian tên toàn cục)

và do đó lệnh gán ở Dòng 3 sẽ định nghĩa lại biến toàn cục `a` (đã được định nghĩa là 0 ở Dòng 16) nên kết quả xuất ra của Dòng 18 là 1 (chứ không phải 0 như trước). Hàm `func2` không thể định nghĩa lại biến toàn cục `a` (vì trong thân nó không có lệnh `global`) nên Dòng 20 cũng xuất ra 1.

- Tiếp theo, với `a` được khai báo là `global` ở Dòng 2 và `a` được khai báo `nonlocal` ở Dòng 8 (xóa dấu ghi chú) thì `a` trong hàm `func3` sẽ được đặt ở không gian tên cục bộ của `func2` chứ không phải `func3` nên `a` cục bộ của `func2` (định nghĩa ở Dòng 12) sẽ được định nghĩa lại ở Dòng 9, do đó kết quả xuất ra của Dòng 14 là 4 (chứ không phải 2 như trước).

Điều gì xảy ra nếu lệnh `nonlocal` ở Dòng 8 được thay bằng `global`? Bạn tự phân tích và chạy thử.

- Kiểu gì thì lời gọi hàm `func3` ở Dòng 21 (nếu bỏ dấu ghi chú) sẽ sinh lỗi `NameError` vì tên `func3` không có trong không gian tên toàn cục (và không gian tên dựng sẵn).

Với kiến thức về phạm vi và không gian tên ở trên, ta điểm lại vài tình huống đã gặp để hiểu rõ hơn:

- Trong mã `turtle_race.py` ở trên, tên (biến) `FOUR_TURTLES` (“danh sách” 4 chú rùa) là toàn cục (cùng với nhiều tên khác). Nó được dùng “bình thường” ở các hàm như `check_winner` vì không có tên cục bộ nào trong các hàm đó che nó. Ta cũng có thể khai báo nó là `global` nhưng không cần thiết. Tương tự với mã `turtle_draw.py` ở Phần 8.6 hay mã `Pisa_tower.py` ở Phần 9.6.
- Trong mã `hello.py` ở Phần 8.1, ta có thể dùng tên `name` thay cho `a_name` (Dòng 10) mà không sợ đụng độ với tên tham số `name` của hàm `sayhello` (Dòng 1). Điều đó được thể hiện trong mã `Newton_sqrt.py` ở Phần 8.2.
- Đặc biệt, trong mã `bloom_square.py` ở Phần 9.6, ta định nghĩa lại biến toàn cục `bloom` (lệnh gán ở Dòng 7) nên phải khai báo `global` cho nó. Tương tự với mã `Pisa_tower.py` ở Phần 9.6.

Ta cũng lưu ý là các tên dùng trong một số cấu trúc lệnh, như tên biến lặp trong vòng lặp `for`, được đặt ở không gian tên hiện tại khi lệnh được thực thi (nếu lệnh `for` đặt trong một hàm thì là không gian tên cục bộ của hàm đó, nếu không thì là không gian tên toàn cục). Do đó, nó sẽ “đề” hoặc che các tên trùng nếu có như minh họa sau:

```

1 >>> i = "Hello"
2 >>> for i in range(6): print(i, end=" ")
0 1 2 3 4 5
3 >>> print(i)
5

```

Cuối cùng, lời gọi `dir()` xuất ra danh sách các tên hiện có trong không gian tên hiện tại (nơi-lúc `dir` được gọi). Chẳng hạn, nếu `dir` được gọi bên trong hàm thì là không gian tên cục bộ của hàm đó, nếu không thì là không gian tên toàn cục.

10.8 Thuộc tính

Mỗi đối tượng sở hữu một không gian tên của riêng nó mà các tên trong đó được gọi là **thuộc tính** (attribute) của đối tượng. Ta có thể truy cập thuộc tính của một đối tượng với cú pháp:

`<Obj>.<Attr>`

Dấu chấm (.) là **toán tử tra cứu thuộc tính** (attribute reference operator), thực hiện thao tác tra tên `<Attr>` trong đối tượng `<Obj>` và trả về tham chiếu đến đối tượng của tên tương ứng nếu có hoặc lỗi thực thi `AttributeError` nếu không có.²⁵ Quá trình tra tên này cũng khá phức tạp (vì nó có thể phải tìm ở nhiều không gian tên như Phần 10.7) mà ta sẽ rõ hơn ở Bài 14.

Ta đã dùng toán tử chấm để truy cập các hàm của module dựng sẵn ngay từ Phần 3.2 (hay module nói chung ở Phần 8.8) vì các module sau khi nạp (`import`) sẽ trở thành **đối tượng module** (module object) và các biến, hàm, ... định nghĩa trong module sẽ trở thành các thuộc tính của đối tượng module đó. Để hiểu rõ hơn, tạo module `amodule` với mã đơn giản như sau:

<https://github.com/vqhBook/python/blob/master/lesson10/amodule.py>

```

1  import math
2
3  def sqrt():
4      return 2 * a
5
6  a = b = 10

```

Giả sử file mã được để ở thư mục `D:\Python\lesson10`, chạy IDLE từ đầu và tương tác như sau:

```

1  >>> import os; os.chdir(r"D:\Python\lesson10")
2  >>> import amodule
3  >>> type(amodule); dir(amodule)
<class 'module'>
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
 '__name__', '__package__', '__spec__', 'a', 'b', 'math', 'sqrt']
4  >>> a = 15; dir()
['__annotations__', '__builtins__', '__doc__', '__loader__',
 '__name__', '__package__', '__spec__', 'a', 'amodule', 'os']
5  >>> print(amodule.a, getattr(amodule, "b"))
10 10
6  >>> amodule.a = 5; setattr(amodule, "b", 20)
7  >>> amodule.sqrt()
10

```

²⁵Phân biệt toán tử “chấm” này với dấu chấm thập phân.

```

8 >>> del amodule.b; hasattr(amodule, "b")
False
9 >>> delattr(amodule, "a"); amodule.sqrt()
...
NameError: name 'a' is not defined
10 >>> print(type(amodule.math), amodule.math.sqrt(2))
<class 'module'> 1.4142135623730951

```

Ta lần theo quá trình thực thi các lệnh tương tác trên của IDLE để hiểu rõ hơn:

- Khi IDLE khởi động Python, nó tạo không gian tên dựng sẵn rồi không gian tên toàn cục (của phiên chạy IDLE).
- Sau đó nó nạp module `os` và đổi thư mục hiện tại thành thư mục có chứa file mã `amodule.py`.
- Lệnh `import amodule` nạp module `amodule`, nghĩa là: nó tạo không gian tên toàn cục cho `amodule` rồi thực thi từng lệnh trong `amodule` mà theo đó các tên `math`, `sqrt`, `a`, `b` tham chiếu đến các đối tượng module, hàm, số nguyên tương ứng được thêm vào không gian tên toàn cục của `amodule`. Sau khi kết thúc thực thi `amodule` thì không gian tên toàn cục này được chuyển thành không gian tên của đối tượng module (với các thuộc tính `math`, `sqrt`, `a`, `b`) do tên `amodule` tham chiếu đến. Tên `amodule` được tạo trong không gian tên toàn cục của phiên chạy IDLE.
- Lệnh `dir(amodule)` trả về danh sách các tên trong không gian tên của đối tượng `amodule`, tức là danh sách các thuộc tính của nó. Như ta thấy danh sách này có chứa `math`, `sqrt`, `a`, `b` và các thuộc tính đặc biệt `__xyz__` khác (ta đã biết `__doc__` ở Phần 8.7, `__name__` ở Bài tập 8.10, ngoài ra `__file__` cho biết đường dẫn file mã, `__annotations__` cho biết thông tin chú giải nếu có (Bài tập ??), ...).
- Lệnh gán tiếp theo tạo tên `a` trong không gian tên toàn cục của phiên làm việc như kết quả của lệnh `dir()` sau đó cho thấy. Lưu ý, có sự đụng độ tên ở đây, tên `a` (và một số tên khác) xuất hiện ở cả không gian tên toàn cục của phiên làm việc và không gian tên của đối tượng `amodule`. Tuy nhiên, như ta đã biết, điều này không thành vấn đề vì chúng nằm trong các không gian tên khác nhau và cách truy cập sẽ chỉ rõ không gian tên nào được dùng.
- 2 lệnh tiếp theo truy cập `a`, `b` trong module `amodule` (thuộc tính của `amodule`) theo cách thông thường hoặc dùng hàm `getattr` cho thuộc tính của đối tượng. Ta thấy giá trị của `a` là 10 (chứ không phải 15) vì đây là `a` trong không gian tên của `amodule`.
- Ta có thể định nghĩa lại tên (tức là cho tên tham chiếu đến đối tượng mới) bằng lệnh gán thông thường hay hàm `setattr` cho thuộc tính của đối tượng.
- Lệnh sau đó thực thi hàm `sqrt` của `amodule`. Lưu ý, như trong mã, lệnh này dùng biến toàn cục `a` nhưng biến này lấy trong không gian tên toàn cục của

module (mà giờ chính là không gian tên của đối tượng `amodule`) chứ không phải không gian tên toàn cục của phiên làm việc. Hơn nữa có sự đụng độ tên `sqrt` trong không gian tên của `amodule` với không gian tên của module `math` (mà `math` giờ là một thuộc tính của `amodule`). Cũng vậy, điều này không thành vấn đề vì chúng nằm trong các không gian tên khác nhau.

- Ta có thể dùng hàm `hasattr` để kiểm tra đối tượng có thuộc tính nào đó không (có tên nào đó trong không gian tên của đối tượng không). Ta có thể xóa tên khỏi không gian tên bằng lệnh `del` hoặc hàm `delattr` với tên trong không gian tên của đối tượng (thuộc tính của đối tượng).

Trường hợp khác của việc dùng toán tử chấm chính là truy cập phương thức hay trường của đối tượng như trong mã `turtle_race.py` và Phần 10.5. Lưu ý, như vậy, các thành phần (phương thức hay trường) đều là thuộc tính của đối tượng.²⁶ Để rõ hơn, chạy file mã `turtle_race.py` ở Phần 10.2, đóng cửa sổ con rùa và tương tác tiếp với Python trên IDLE như sau:

```
1 ===== RESTART: D:\Python\lesson10\turtle_race.py =====
2 >>> dir(Leo)
  [..., '__class__', 'color', 'name', 'radius', ...]
3 >>> print(Leo.__class__, Leo.name, Leo.radius, Leo.color())
  <class 'turtle.Turtle'> Leonardo 10 ('blue', 'blue')
4 >>> x = 1.25; dir(x)
  [..., '__mul__', 'as_integer_ratio', 'is_integer', ...]
5 >>> print(x.__class__, x.is_integer(), x.as_integer_ratio())
  <class 'float'> False (5, 4)
```

Trừ việc ta đã gán (tạo) các trường `name`, `radius` cho các con rùa thì các phương thức (hay các trường khác) của con rùa như `color`, `forward`, ... từ đâu mà có? Từ lớp của nó (`turtle.Turtle`) như ta sẽ thấy khi tự mình viết lớp ở Bài 14. Đặc biệt, lớp của đối tượng có thể được lấy từ thuộc tính `__class__` của đối tượng (hoặc từ hàm `type` như đã biết). Thêm nữa, cách thức tổ chức không gian tên (của đối tượng hay không gian tên toàn cục/cục bộ) sẽ được làm rõ ở Bài tập 12.12.

Tóm tắt

- “Thuật toán + Cấu trúc dữ liệu = Chương trình” (“Algorithms + Data Structures = Programs”).
- Mọi thứ trong Python đều là đối tượng. Lớp xác định kiểu của đối tượng.
- Mọi đối tượng đều có 3 đặc trưng quan trọng là danh tính, kiểu và giá trị.
- Đối tượng có thể thay đổi giá trị được gọi là khả biến; đối tượng không thể thay đổi giá trị được gọi là bất biến. Đối tượng của các kiểu dữ liệu cơ bản đều là bất biến; đa số các đối tượng khác là khả biến.

²⁶Thuật ngữ thuộc tính đôi khi được dùng với nghĩa hẹp để chỉ các trường mà thôi.

- Python quản lý các tên bằng không gian tên và phạm vi tên. Có 4 không gian tên và phạm vi tên là: cục bộ, bao ngoài, toàn cục và dựng sẵn.
- Lệnh `global` khai báo tên có phạm vi toàn cục. Lệnh `nonlocal` khai báo tên có phạm vi bao ngoài.
- Phương thức là hàm gắn với đối tượng. Nó được “bên ngoài” dùng để tương tác với đối tượng.
- Trường là dữ liệu gắn với đối tượng. Nó được đối tượng dùng để lưu giữ trạng thái.
- Thuộc tính là tên nằm trong không gian tên của đối tượng. Thuộc tính được truy cập nhờ toán tử tra cứu thuộc tính (toán tử chấm) hoặc thao tác nhờ các hàm `hasattr`, `getattr`, `setattr`, `delattr`.

Bài tập

10.1 Sửa chương trình đua rùa (mã `turtle_race.py` ở Phần 10.2) để khi rùa ăn được pizza thì sẽ lớn lên một chút và/hoặc bò nhanh hơn một chút. Cuộc đua tiếp diễn cho đến khi 10 miếng pizza được ăn xong. Rùa thắng cuộc là rùa “to” nhất (ăn được nhiều pizza nhất).

Nếu muốn cho cả 10 miếng pizza cùng lúc thay vì lần lượt thì làm sao?

10.2 Bao đóng hàm. Trong Phần 10.7, ta đã thấy minh họa về hàm lồng. Hàm dạng này thường được dùng để tạo **bao đóng hàm** (function closure) mà mục đích là gắn kèm “ngữ cảnh” cho hàm. Cụ thể, hàm bên trong có thể truy cập biến cục bộ của hàm bên ngoài (theo cơ chế được mô tả ở Phần 10.7). Quan trọng, việc truy cập này thậm chí được phép ngay cả khi hàm ngoài đã kết thúc thực thi.

Ta sẽ không đi sâu vào cơ chế nhưng bạn có thể tưởng tượng: tương tự như `import`, không gian tên toàn cục của module được giữ lại và gắn làm không gian tên của đối tượng module; không gian tên cục bộ của hàm bao ngoài sẽ được giữ lại và gắn cho đối tượng hàm. Không gian tên này trở thành ngữ cảnh hay môi trường của hàm. Theo cách này, bao đóng hàm còn được gọi là **hàm sản xuất** (factory function) vì nó giúp tạo ra các hàm “cụ thể khác nhau”. Đúng hơn, cùng một hàm nhưng môi trường khác nhau.

Có thể bạn cảm thấy hơi mơ hồ. Điều này là tự nhiên vì kĩ thuật bao đóng hàm được Python làm ngầm định bên dưới. Ta sẽ làm tường minh việc “gắn môi trường cho hàm” bằng cách viết lớp ở Phần 14.1.

Sau đây là một chương trình minh họa kĩ thuật bao đóng hàm. Chạy chương trình, gõ một phím kí tự (chữ cái hoa/thường) hoặc kí số, con rùa sẽ xuất kí tự đó ra giữa màn hình.

https://github.com/vqhBook/python/blob/master/lesson10/func_cls.py

```

1 import turtle as t
2 import string

```

```

3
4 def write(char):
5     def _write():
6         t.clear()
7         t.write(char, align="center", font=f"Arial {s}")
8         t.update()
9     return _write
10
11 s = 300
12 t.hideturtle(); t.tracer(False); t.color("red")
13 t.up(); t.right(90); t.forward(2*s//3); t.down()
14
15 chars = string.ascii_letters + string.digits
16 for i in range(len(chars)):
17     t.onkey(write(chars[i]), chars[i])
18
19 t.listen()
20 t.exitonclick()

```

Ta cũng đăng kí sự kiện nhấn phím nhưng vì có quá nhiều phím nên ta không thể đăng kí “thủ công” như cách đăng kí 4 phím mũi tên ở mã `turtle_draw.py` Phần 8.6. Ta đăng kí “hàng loạt” cho tất cả các kí tự chữ cái và chữ số bằng vòng lặp `for` ở Dòng 16-17. Trước hết, `string.ascii_letters` và `string.digits` là các chuỗi chứa các kí tự chữ cái và chữ số. Hơn nữa, tên phím mà con rùa dùng để đăng kí cho các phím này chính là kí tự đó luôn. Nếu đăng kí thủ công thì ta sẽ dùng rất nhiều lệnh đại khái như:

```

15 t.onkey(lambda: write("a"), "a")
16 t.onkey(lambda: write("b"), "b")
17 # ...

```

Vấn đề cốt lõi là ta cần tạo được hàng loạt hàm xuất chuỗi cho mỗi kí tự khác nhau và ta đã dùng kĩ thuật bao đóng hàm. Cụ thể, hàm thực sự mà con rùa dùng là `_write` không tham số (để có thể là trình xử lý sự kiện nhấn phím) nhưng `_write` vẫn “nhớ” kí tự nó cần xuất nhờ biến cục bộ của hàm ngoài (tham số `char` của `write`). Lưu ý, cách đặt các tên hàm `write` bên ngoài và `_write` bên trong trông có vẻ bí ẩn nhưng chúng cũng chỉ là các tên bình thường thôi (bạn có thể đặt tên khác nếu muốn).²⁷

Tương tự, dùng kĩ thuật bao đóng hàm, sửa lại mã `turtle_draw.py` ở Phần 8.6 để sau khi chọn hướng bằng các phím mũi tên, người dùng nhấn các phím số 1 đến 9 để di chuyển con rùa theo khoảng cách tăng dần (chẳng hạn, phím

²⁷Việc hàm bên trong dùng cùng tên hàm bao ngoài với tiền tố `_` chỉ là một thói quen của cộng đồng Python. Ta sẽ bàn thêm qui ước đặt tên này sau.

số 1 đi 5 bước, phím số 2 đi 10 bước, ...).²⁸

10.3 Các dạng import. Ta đã dùng lệnh `import` từ sớm (Phần 3.2) để có thể truy cập các đối tượng của một module từ module khác. Lệnh `import` cơ bản thực hiện 2 việc: (1) tìm module, nạp và khởi động nó nếu cần; (2) định nghĩa (các) tên trong không gian tên hiện tại theo phạm vi thực thi lệnh `import`. Thử minh họa sau:

```
1 >>> def func(): import math; print(math.pi)
2
3 >>> func()
3.141592653589793
4 >>> math.pi
...
NameError: name 'math' is not defined
```

Có lỗi `NameError` ở Dòng 4 vì tên `math` (tham chiếu đến đối tượng module `math`) được tạo trong không gian tên cục bộ của hàm `func` mà không gian tên này bị gỡ bỏ khi hàm kết thúc thực thi.

Chạy lại IDLE và thử minh họa sau:

```
1 >>> import math
2 >>> def func(): print(math.pi)
3
4 >>> func()
3.141592653589793
5 >>> math.pi
3.141592653589793
```

Lần này tên `math` được tạo ở không gian toàn cục (của phiên làm việc) nên việc truy cập đến `math` suôn sẻ ở cả trong lẫn ngoài hàm.

Lệnh `import` có một số dạng cơ bản như sau:

- `import <Module>`: nạp module `<Module>` và tạo tên `<Module>` trong không gian tên hiện tại tham chiếu đến đối tượng module. Đây là dạng thông thường nhất mà ta đã gặp từ Phần 3.2.
- `import <Module> as <Name>`: nạp module `<Module>` nhưng kết buộc đối tượng module với tên `<Name>` (thay vì tên `<Module>` như cách trên). Đây là dạng “đặt lại tên” mà ta đã gặp từ Bài tập 3.10 hay từ Bài 5 với con rùa.
- `from <Module> import <Attr>`: nạp module `<Module>` và tạo tên `<Attr>` trong không gian tên hiện tại tham chiếu đến thuộc tính `<Attr>` của đối tượng module. Đây là dạng “nạp đúng cái cần” mà ta đã gặp từ Bài tập 3.8, 3.9.

²⁸Tham khảo đáp án tại https://github.com/vqhBook/python/blob/master/lesson10/turtle_draw.py.

- `from <Module> import <Attr> as <Name>`: tương tự cách trên nhưng kết buộc `<Attr>` với tên `<Name>`. Ta đã gặp dạng này từ Phần 9.5.
- `from <Module> import *`: nạp module `<Module>` và “đưa” tất cả các thuộc tính “công khai” của đối tượng module vào không gian tên hiện tại. Đây có thể xem là dạng “nạp hết cho rồi”.

Dạng `import` này tiện lợi vì giúp ta gõ phím ít hơn. Tuy nhiên, nó đi ngược lại nguyên tắc và lợi ích của module là gom các đối tượng liên quan vào trong các không gian tên để dễ tổ chức và quản lý. Với cách `import` này (nhất là khi ta nạp nhiều module), ta không biết thuộc tính nào là của module nào nên chương trình rất rối rắm và bừa bộn. Bạn cần biết dạng “`import *`” này để có thể đọc mã người khác viết nhưng nhớ là: *tuyệt đối không dùng nó*. Việc lười biếng bây giờ sẽ khiến bạn phải trả giá sau này (khi chương trình của bạn phức tạp lên). Hãy *viết mã thật gọn gàng và ngăn nắp*!

Ta cũng có thể “nạp cùng lúc nhiều thứ” bằng cách dùng dấu phẩy phân cách phù hợp như ta đã làm (chẳng hạn ở Phần 9.5).

Tra cứu lệnh `import` tại https://docs.python.org/3/reference/simple_stmts.html#import và hướng dẫn của PEP 8 cho `import` tại <https://www.python.org/dev/peps/pep-0008/#imports>.

10.4 Visual Studio Code. Đến giờ, năng lực lập trình của bạn đã vững. Bạn nên tạm chia tay IDLE thân thương để chuyển qua thử nghiệm các IDE mạnh mẽ hơn. Phần này giới thiệu một môi trường lập trình rất được yêu thích cho Python (và nhiều ngôn ngữ khác) là Visual Studio Code.²⁹ Bạn vào trang chủ của VS Code (<https://code.visualstudio.com/>), tải file cài đặt và cài bình thường. Sau đó bạn cài thêm mở rộng hỗ trợ Python của VS Code (nhấn nút Install trong trang <https://marketplace.visualstudio.com/items?itemName=ms-python.python>).

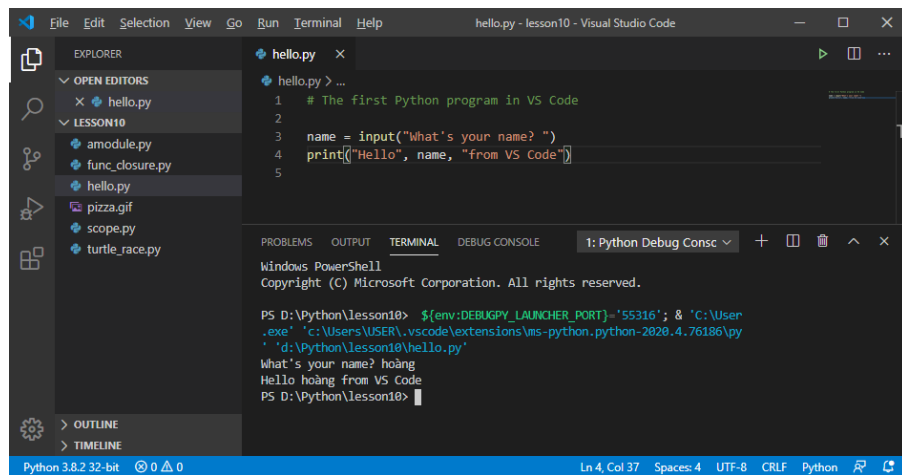
Trong cửa sổ VS Code, bạn mở (hoặc tạo) thư mục `D:\Python\lesson10`, tạo mới một file tên `hello.py` với nội dung như hình sau. Bạn cũng cần chọn Python Interpreter để dùng (nhấp vào nút ở dưới góc trái cửa sổ và chọn Python Interpreter mà ta đã cài từ Bài 1). Nhấn `Ctrl+F5` để chạy chương trình và gõ tên trong cửa sổ TERMINAL.

VS Code cũng hiển thị hộp thoại nhắc bạn cài đặt Linter pylint (ở góc dưới phải). Đây là công cụ kiểm tra mã Python có được viết tốt hay không (theo PEP 8). Bạn nên cài bằng cách nhấp vào nút Install.

Rõ ràng, so với IDLE, VS Code choáng ngợp hơn nhiều nên bạn cần đọc VS Code để quen thuộc hơn.³⁰ Chạy lại các file mã đã gõ trong VS Code. Gõ lại các file mã và dùng các chức năng hỗ trợ soạn mã của VS Code.

²⁹Xếp hạng nhất trong nhóm các công cụ lập trình được yêu thích theo khảo sát của Stack Overflow năm 2019 (<https://insights.stackoverflow.com/survey/2019#development-environments-and-tools>).

³⁰Tin tôi đi, khi bạn đã quen thuộc VS Code thì bạn không còn muốn dùng IDLE nữa đâu.



Bạn tham khảo thêm các link sau:

- https://code.visualstudio.com/docs/python/python-tutorial#_create-a-python-hello-world-source-code-file,
- https://code.visualstudio.com/docs/python/python-tutorial#_run-hello-world,
- <https://code.visualstudio.com/docs/python/editing>,
- <https://code.visualstudio.com/docs/python/linting>.

Chỗ nào “lạ quá” thì tạm thời bỏ qua, ta sẽ quay lại khi thuần thục hơn.

10.5 Debug. Như đã nói từ Phần 2.4, **debug** là việc tìm và xóa lỗi trong chương trình. Các IDE hỗ trợ việc debug bằng các **trình sửa lỗi** (debugger). Hơn nữa, debugger cũng cung cấp nhiều công cụ giúp ta theo dõi, kiểm tra, tìm hiểu, ... hoạt động của chương trình. Có thể nói, debugger là công cụ không thể thiếu với lập trình viên và việc dùng thành thạo nó là kỹ năng quan trọng hàng đầu.

Một trong những lý do mà VS Code được ưa chuộng là các Debugger mạnh mẽ và tiện lợi của nó. Bạn tham khảo link https://code.visualstudio.com/docs/python/python-tutorial#_configure-and-run-the-debugger để biết cách dùng Python Debugger của VS Code.³¹

Chạy lại mã minh họa trong Phần 10.7 với hỗ trợ của Debugger để hiểu rõ hơn (chạy từng bước, quan sát kết quả và đối sánh với mô tả ở Phần 10.7).

³¹ Bạn cũng có thể coi thêm các video hướng dẫn trên mạng.

Bài 11

Danh sách và chuỗi

Ta đã làm việc với dữ liệu đơn lẻ trong các bài trước. Có rất nhiều tình huống mà ta phải làm việc cùng lúc với nhiều dữ liệu/đối tượng như các chú rùa ở Phần 10.2. Đây là lúc ta cần các mô hình hay **cấu trúc dữ liệu** (data structure) phức tạp giúp ta quản lý tập nhiều đối tượng. Bài này tìm hiểu mô hình cơ bản và quan trọng nhất trong số đó, danh sách. Ta cũng tìm hiểu kỹ hơn về chuỗi.

11.1 Duyệt qua nhiều đối tượng với danh sách

Trong Bài tập 3.4, bạn được yêu cầu tính căn của nhiều số (câu (a), (b), ...). Có lẽ, bạn đã làm như sau:

```
1 import math
2
3 print(math.sqrt(0.0196))    # Câu (a)
4 print(math.sqrt(1.21))     # Câu (b)
5 print(math.sqrt(2))        # Câu (c)
6 print(math.sqrt(3))        # Câu (d)
7 print(math.sqrt(4))        # Câu (e)
8 print(math.sqrt(225/256))   # Câu (f)
```

Rõ ràng cách viết này không tốt, ta có công việc được tham số hóa (“tính căn của một số nào đó”) và đáng dấp của lặp nhưng ta buộc lòng phải “sao chép và chỉnh sửa” vì không biết cách dùng `for` hay `while` thế nào. Thật vậy, nếu cần tính căn của các số từ 1 đến 100 (thậm chí còn nhiều số hơn các số ở trên) ta có thể dễ dàng dùng lệnh lặp `for` như sau:

```
for i in range(1, 101): print(math.sqrt(i))
```

Tuy nhiên, trong trường hợp này, ta khó viết được tương tự vì không có cách nào điều khiển biến lặp để được các số cần tính căn trên. Sử dụng phương tiện của

bài này, ta viết ngắn gọn được bằng lập như sau:

https://github.com/vqhBook/python/blob/master/lesson11/sqrt_cal.py

```

1 import math
2
3 # danh sách các số cần tính căn
4 nums = [0.0196, 1.21, 2, 3, 4, 225/256]
5 for i in nums:
6     print(math.sqrt(i))

```

Thay vì để các số “đơn lẻ”, ta cho chúng vào chung “danh sách các số cần tính căn”. Để tính căn nhiều số nữa, ta chỉ cần thêm chúng vào danh sách này, còn mã xử lý chúng thì vẫn không đổi. Như vậy, **danh sách** (list) là một kiểu dữ liệu (thường được gọi là mô hình dữ liệu/cấu trúc dữ liệu) giúp quản lý nhiều đối tượng theo một thứ tự nào đó (ở đây là thứ tự các câu (a), (b), ...). Các đối tượng trong danh sách được gọi là các **phần tử** (element) của danh sách. Để mô tả một danh sách cụ thể, ta liệt kê các phần tử của nó theo thứ tự mong muốn trong cặp dấu ngoặc vuông ([...]) với dấu phẩy (,) phân cách các phần tử. Dĩ nhiên, trường hợp danh sách chỉ có 1 phần tử thì không dùng dấu phẩy và trường hợp danh sách không có phần tử nào, ta viết [], gọi là **danh sách rỗng** (empty list).

Để phân biệt với các **kiểu dữ liệu đơn** (simple data type) ta đã biết, danh sách được nói là thuộc nhóm **kiểu dữ liệu ghép** (compound data type) vì nó bao gồm nhiều dữ liệu thành phần bên trong (chính là các phần tử của danh sách). Cũng vì chứa nhiều phần tử nên danh sách còn được gọi là **bộ chứa** (container) và vì có quản lý thứ tự các phần tử nên được gọi là **bộ chứa tuần tự** (sequential/linear container) hay ngắn gọn là **dãy** (sequence).

Lệnh for ở trên, như vậy, giúp ta “duyet qua lần lượt các phần tử (ở đây là số) trong danh sách”. Ta cũng đã dùng cách này để xử lý 4 chú rùa trong mã `turtle_race.py` ở Phần 10.2. Có gì mới không? Không, nó cũng chính là for thân thương mà ta đã làm nhiều từ Bài 7 đến giờ. Đúng hơn, chỉ có 1 dạng for mà “for với range” ta đã làm trước giờ là một trường hợp đặc biệt.

Cụ thể, hàm range giúp ta tạo danh sách các số nguyên mà sau đó, theo cách tổng quát, lệnh for đặt biến lặp tham chiếu lần lượt các phần tử trong danh sách. Nói cách khác, để tính căn các số từ 1 đến 100 ta chỉ cần tạo danh sách cần tính căn là các số này (thay vì các số trong Bài tập 3.4), nghĩa là, thay lệnh gán ở Dòng 4 trong mã trên bằng lệnh gán:

```

4 nums = list(range(1, 101))

```

Lời gọi `range(1, 101)` giúp “duyet” qua dãy các số nguyên từ 1 đến 100 và hàm `list` giúp “ghép” các số này thành danh sách (ta sẽ rõ hơn ở Phần 14.8).

Nếu hơi bối rối thì ta có thể viết mã gần gũi hơn như sau:

https://github.com/vqhBook/python/blob/master/lesson11/sqrt_cal2.py

```

1 import math
2

```

```

3 # danh sách các số cần tính căn
4 nums = [0.0196, 1.21, 2, 3, 4, 225/256]
5 for i in range(len(nums)):
6     print(math.sqrt(nums[i]))

```

Cách làm này y hệt cách ta duyệt qua các kí tự của chuỗi (như trong Phần 7.7 và Bài tập 10.2) chỉ khác là ta duyệt qua các phần tử của danh sách.¹ Thật vậy, số lượng phần tử của một danh sách được gọi là **chiều dài** (length) của danh sách và ta có thể tính bằng hàm `len` như đã làm với chuỗi. Cũng như kí tự trong chuỗi, ta có thể truy cập các phần tử trong danh sách bằng **chỉ số** (index) của phần tử đó. Nhớ là phần tử đầu tiên có chỉ số 0, phần tử thứ hai có chỉ số 1, ..., và phần tử cuối cùng có chỉ số $l - 1$ với l là chiều dài danh sách.

Ở trên, ta đã dùng danh sách để xử lý nhiều số. Ta cũng có thể dùng danh sách để xử lý nhiều kiểu dữ liệu (đối tượng) khác.² Chẳng hạn, dùng danh sách, kết hợp với biểu thức lambda, ta có cách viết rất đẹp cho Bài tập 3.4 như sau (tiếp tục từ mã `sqrt_cal2.py` trong Phần 8.5):

https://github.com/vqhBook/python/blob/master/lesson11/sqrt_cal3.py

```

1 import math
2 from string import ascii_lowercase as alphabet
3
4 def cal_sqrt(methods, nums):
5     for method, method_name in methods:
6         print(f"Tính căn bằng phương pháp {method_name}:")
7         for i in range(len(nums)):
8             print("%s) Căn của %g là %.9f" %
9                   (alphabet[i], nums[i], method(nums[i])))
10
11 # danh sách các số cần tính căn
12 nums = [0.0196, 1.21, 2, 3, 4, 225/256]
13
14 # danh sách các phương pháp tính căn (và tên)
15 methods = [
16     (math.sqrt, "math's sqrt"),
17     (lambda x: math.pow(x, 0.5), "math's pow"),
18     (lambda x: pow(x, 0.5), "built-in pow"),
19     (lambda x: x ** 0.5, "exponentiation operator")
20 ]
21
22 cal_sqrt(methods, nums)

```

¹Như ta sẽ thấy ở phần dưới, có thể xem chuỗi là danh sách các kí tự.

²Thậm chí trong Python, các phần tử của cùng danh sách có thể có kiểu khác nhau. Tuy nhiên, ta chỉ nên dùng các **danh sách đồng nhất** (homogeneous list), tức là danh sách có các phần tử cùng kiểu.

Chương trình trên minh họa khá nhiều kỹ thuật:

- Danh sách có thể chứa các đối tượng có kiểu bất kỳ, như danh sách `methods` chứa các cặp phương pháp tính căn, mỗi cặp gồm hàm tính căn (hàm nhận một số và trả về căn của số đó) và chuỗi mô tả tên phương pháp.
- Danh sách cũng là dữ liệu nên có thể được dùng như bao dữ liệu khác, chẳng hạn, được truyền cho các hàm như hàm `cal_sqrt`.
- Yêu cầu “khai phương nhiều số bằng nhiều cách” được viết tự nhiên thành vòng lặp lồng với vòng lặp bên ngoài lặp qua các phương pháp khai phương và vòng lặp bên trong lặp qua các số cần khai phương.
- Vì mỗi phần tử của danh sách `methods` là cặp nên ta đã dùng “cặp biến lặp” `method, method_name` để duyệt qua từng phần tử.
- `string.ascii_lowercase` là chuỗi các chữ cái Tiếng Anh thường “`abc...z`” mà ta dùng để “tự động” điền số thứ tự câu (a, b, ...).
- Vì Python phải “khớp” dấu `]` với dấu `[` nên ta có thể viết các phần tử của danh sách trên nhiều dòng (tương tự cách dùng cặp ngoặc tròn để viết biểu thức dài). Việc này thường giúp viết danh sách “đẹp” hơn (như danh sách `methods` trên).³

Thật sự, danh sách rất quan trọng, nó xuất hiện khắp nơi trong Python do khả năng giúp quản lý và xử lý nhiều đối tượng. Chẳng hạn, trong Phần 3.3, ta đã tra cứu danh sách từ khóa của Python, bạn đếm thử có bao nhiêu từ khóa? Dùng module `keyword` ta có thể tính chiều dài danh sách này như sau:

```
1 >>> import keyword
2 >>> keyword.kwlist
  ['False', 'None', ..., 'with', 'yield']
3 >>> print(type(keyword.kwlist), len(keyword.kwlist))
  <class 'list'> 35
```

Hay như trong Phần 3.4 (và Phần 10.6, 10.8), ta đã dùng hàm dựng sẵn `dir` để nghĩa sơ bộ nhớ của Python. Như kết xuất cho thấy, `dir` trả về một danh sách, danh sách các tên đã tạo của chương trình (tường minh hoặc ngầm định do Python tự động làm).

11.2 Lưu trữ dữ liệu với danh sách và xử lý ngoại tuyến

Trong Bài tập 7.8, yêu cầu viết chương trình cho người dùng nhập số lượng số bất kỳ và xuất ra theo thứ tự tăng dần, tôi gợi ý là “hơi khó”. Tôi đã khiêm tốn á ... thật ra là ... “cực khó”! Nếu chỉ dùng các kỹ thuật tới Bài 7 thì tôi cũng chẳng biết làm thế nào (thật á, không khiêm tốn đâu). Tuy nhiên, dùng danh sách, ta làm rất dễ như sau:

³Đây cũng là khuyến cáo của PEP 8.

```

1  https://github.com/vqhBook/python/blob/master/lesson11/ascending.py
2  nums = []
3  while text := input("Nhập số nguyên (Enter để dừng): "):
4      nums.append(int(text))
5
6  if len(nums) > 0:
7      print("Các số đã nhập theo thứ tự tăng dần là:")
8      for num in sorted(nums):
9          print(num, end=" ")

```

Điểm mấu chốt là ta đã lưu trữ lại tất cả các số người dùng nhập bằng danh sách (nums). Ban đầu, nums là danh sách rỗng (Dòng 1), sau đó ta thêm lần lượt các số người dùng nhập vào nums (dùng phương thức append của danh sách ở Dòng 3). Sau khi người dùng nhập xong (nhấn Enter mà không gõ số), ta xuất ra các số trong nums theo thứ tự tăng dần (bằng hàm sorted trên danh sách nums).

Với yêu cầu na ná như vậy, nhưng Bài tập 7.2b, tìm số lớn nhất trong các số người dùng đã nhập, lại dễ hơn nhiều. Ta có thể viết đơn giản như sau mà không cần dùng danh sách:

```

1  https://github.com/vqhBook/python/blob/master/lesson11/max.py
2  max = None
3  while text := input("Nhập số nguyên (Enter để dừng): "):
4      num = int(text)
5      if not max or max < num:
6          max = num
7
8  if max:
9      print("Số lớn nhất đã nhập là", max)

```

Ta dùng biến max để lưu trữ số lớn nhất cho đến “thời điểm” người dùng nhập. Ban đầu, khi người dùng chưa nhập số nào, max được đặt là None với ý “chưa có số lớn nhất”. Như vậy, với cách lượng giá tắt, điều kiện của lệnh if ở Dòng 4 được hiểu là “chưa có số lớn nhất hoặc nếu có mà số đó nhỏ hơn số hiện tại” (và do đó, nếu điều kiện đúng thì cập nhật max thành số mới lớn hơn max “cũ”). Tôi cũng đã dùng cách viết gọn not max và max nhiều “cảm xúc” hơn cách viết max == None và max != None trong các điều kiện của if ở Dòng 4 và Dòng 7.

Điều quan trọng trong chương trình trên là ta đã không dùng danh sách. Đây là điều tốt vì việc dùng danh sách tuy tiện lợi nhưng tốn kém (về bộ nhớ và thời gian thực thi). Cũng với yêu cầu na ná như vậy nhưng là tìm số lớn thứ 2 trong các số người dùng đã nhập (Bài tập 7.2d), bạn có thể làm như trên (không dùng danh sách) không? Được. Bạn suy nghĩ, làm thử rồi đối chiếu với đáp án tại https://github.com/vqhBook/python/blob/master/lesson11/second_max.py.⁴

Yêu cầu tổng quát “tìm số lớn thứ k (do người dùng chọn)” thì lại khác. Ta buộc lòng phải dùng danh sách như sau:

⁴Rất đáng giá để suy ngẫm đó.

https://github.com/vqhBook/python/blob/master/lesson11/k_max.py

```

1 nums = []
2 while text := input("Nhập số nguyên (Enter để dừng): "):
3     nums.append(int(text))
4
5 nums.sort(reverse=True)
6
7 k = int(input("Bạn muốn tìm số lớn thứ mấy? "))
8 if 0 < k <= len(nums):
9     print(f"Số lớn thứ {k} đã nhập là {nums[k - 1]}")

```

Để đơn giản, ta giả sử người dùng nhập các số khác nhau, khi đó, để tìm số lớn thứ k ta chỉ cần nhập tất cả các số vào một danh sách, sắp xếp giảm dần và truy cập phần tử thứ k của danh sách sau khi sắp. Phương thức `sort` của danh sách giúp ta sắp xếp ngay trên danh sách (chứ không phải trả về danh sách mới như hàm `sorted`), hơn nữa, tham số `reverse` xác định có sắp “ngược” (tức là giảm dần) hay không. Cũng lưu ý, theo ngôn ngữ thông thường, số thứ tự được tính từ 1 (1 là đầu tiên); còn trong Python, chỉ số của phần tử được tính từ 0 (0 là đầu tiên).

Các minh họa trên cho thấy rõ 2 cách hay 2 chiến lược xử lý nhiều dữ liệu với các ưu/nhược điểm khác nhau:

- **Trực tuyến** (online): nhận dữ liệu là xử lý liền mà không lưu trữ (hay lưu trữ rất ít). Cách này có lợi là không tốn (hoặc tốn rất ít) bộ nhớ và có ngay kết quả khi nhận xong dữ liệu cuối cùng. Hơn nữa, nó có tính tương tác và tức thời vì có các kết quả xử lý trung gian. Chẳng hạn ta có `max` là số lớn nhất từ đầu cho đến số vừa nhập xong.
- **Ngoại tuyến** (offline): nhận hết dữ liệu, lưu trữ lại rồi mới xử lý toàn bộ. Cách này có nhược điểm là tốn bộ nhớ, không có tính tương tác và có độ trễ do phải đợi xử lý toàn bộ. Bù lại, cách này thường dễ thực hiện hơn do ta có đầy đủ dữ liệu khi xử lý. Với cách này, ta thường dùng danh sách (và các kiểu chứa khác) để lưu trữ dữ liệu.

Nếu không để ý đến hiệu năng (bộ nhớ và thời gian) thì *cách dùng danh sách để lưu trữ dữ liệu và xử lý ngoại tuyến thường mang lại giải pháp đơn giản hơn*.

11.3 Các thao tác trên danh sách

Thử minh họa sau:

```

1 >>> li = list("abcd")
2 >>> print(li, type(li), li[-1], li[len(li) - 1])
['a', 'b', 'c', 'd'] <class 'list'> d d
3 >>> li[len(li)]
...
IndexError: list index out of range

```

```

4 >>> print(li[0:2], li[:2], li[::-1], li[slice(0, -1, 2)])
    ['a', 'b'] ['a', 'b'] ['d', 'c', 'b', 'a'] ['a', 'c']
5 >>> li[0] = "A"; li[-2:] = ["C", "D", "E"]; li
    ['A', 'b', 'C', 'D', 'E']
6 >>> del li[0]; li[-2:] = []; li
    ['b', 'C']
7 >>> li += ["e", "f"] * 2; li
    ['b', 'C', 'e', 'f', 'e', 'f']
8 >>> "C" in li and "c" not in li
    True
9 >>> li.reverse(); li
    ['f', 'e', 'f', 'e', 'C', 'b']
10 >>> li.append("z"); li.insert(0, "a"); li
    ['a', 'f', 'e', 'f', 'e', 'C', 'b', 'z']
11 >>> list(filter(lambda x: x <= "e", li))
    ['a', 'e', 'e', 'C', 'b']

```

Ta đã thực hiện các thao tác cơ bản nhất trên danh sách là: tạo danh sách (`[...]`), truy cập phần tử (toán tử `<chỉ số>`), lấy chiều dài (hàm `len`), duyệt qua các phần tử (lệnh `for`). Minh họa trên mô tả thêm các thao tác hay dùng trên danh sách:

- Tạo danh sách bằng hàm `list` từ các đối tượng phù hợp như chuỗi, “bộ duyệt” (như kết quả trả về của `range`), ...
- Truy cập phần tử bằng chỉ số: chỉ số phần tử (trong Python) có thể là giá trị nguyên âm, với quy ước, -1 là phần tử cuối cùng, -2 là phần tử kế cuối, ... ⁵ Đặc biệt, việc truy cập quá chỉ số sẽ phát sinh lỗi thực thi `IndexError`.
- Truy cập **danh sách con** (sublist) bằng **lát cắt** (slice) với cú pháp:

`<list>[<index1>:<index2>]`

Biểu thức này trả về tham chiếu đến danh sách con gồm liên tiếp các phần tử của danh sách `<list>` từ chỉ số `<index1>` đến trước chỉ số `<index2>` (không bao gồm `<index2>`). Lưu ý, cả 2 chỉ số đều có thể để trống: `<index1>` nếu để trống được hiểu là 0 và `<index2>` nếu để trống được hiểu là chiều dài của danh sách (cũng có nghĩa là -1). Thật ra, lát cắt có thể gồm thêm 1 tham số là “step” với ý nghĩa tương tự như các tham số của hàm `range` (ngoại trừ các số âm mô tả vị trí tính từ cuối). Hơn nữa, ta cũng có thể tạo tường minh đối tượng lát cắt bằng hàm `slice`.

- Sửa phần tử hoặc danh sách con bằng lệnh gán với chỉ số hoặc lát cắt.
- Xóa phần tử bằng lệnh `del` và xóa danh sách con bằng lát cắt.

⁵Đặc trưng này của Python là khá bất thường vì nhiều ngôn ngữ lập trình khác không hỗ trợ nó.

- Nối danh sách bằng toán tử + và sao chép lặp lại nhiều lần bằng toán tử *.
- Kiểm tra một đối tượng có là phần tử của danh sách hay không bằng toán tử **kiểm tra thành viên** (membership test) in và not in.
- Ta cũng có thể thực hiện một số thao tác trên danh sách bằng các phương thức append (thêm phần tử vào cuối danh sách), insert (chèn một phần tử vào vị trí được cho), ... Bạn có thể tra cứu các phương thức này khi cần tại <https://docs.python.org/3/tutorial/datastructures.html#more-on-lists>.
- Lọc (chọn) ra các phần tử của danh sách bằng hàm filter. Hàm này nhận đối số đầu tiên là một hàm kiểm tra mà sẽ được gọi trên từng phần tử của danh sách ở đối số thứ 2 để chọn ra các phần tử thỏa hàm kiểm tra (tức là hàm kiểm tra trả về đúng trên phần tử đó). Kết quả trả về của filter là một “bộ duyệt” (tương tự kết quả trả về của range).

Điều quan trọng cần nhớ, *danh sách là cấu trúc khả biến* (xem lại Phần 10.4) như minh họa trên và dưới đây cho thấy:

```

1 https://github.com/vqhBook/python/blob/master/lesson11/list\_remove\_dup.py
2 def remove_duplicates(li):
3     i = 0
4     while i < len(li):
5         j = i + 1
6         while j < len(li):
7             if li[j] == li[i]:
8                 del li[j]
9             else:
10                j += 1
11        i += 1

```

Hàm `remove_duplicates` xóa các phần tử trùng (giữ lại phần tử ở vị trí xuất hiện đầu tiên) trong danh sách `li` mà nó nhận vào. Vì hàm sửa trực tiếp trên `li` nên ta dùng lệnh lặp `while` thay cho `for`.⁶ Ý tưởng là: với mỗi phần tử trong danh sách (có chỉ số `i`) ta duyệt qua các phần tử từ sau đó (có chỉ số `j`) nếu nó trùng thì ta xóa nó (vẫn giữ `j` vì sau khi xóa, các phần tử sau sẽ “dồn lên”), nếu không thì ta giữ (không xóa mà tăng `j`). Chạy module trên để có định nghĩa hàm và tương tác trong phiên làm việc đó như sau:

```

===== RESTART: D:\Python\lesson11\list_remove_dup.py =====
1 >>> a = [1, 2, 0, 9, 2, 0, 1, 9]; remove_duplicates(a); a
[1, 2, 0, 9]

```

Như bạn thấy, sau lời gọi hàm `remove_duplicates(a)`, danh sách `a` bị xóa các phần tử trùng. Ta làm rõ lời gọi hàm để hiểu kĩ hơn tại sao danh sách `a` thay đổi: danh sách ban đầu tham chiếu bởi `a` được truyền cho tham số `li`, như vậy trong

⁶Mặc dù vẫn có thể dùng `for`, nhưng nếu phải lặp trên một danh sách mà trong thân lặp có thêm/xóa phần tử thì ta không nên dùng `for`.

hàm, li tham chiếu đến cùng danh sách như a; vì thân hàm xóa trực tiếp trên danh sách li tham chiếu, nên kết quả, sau khi hàm kết thúc, ta có danh sách ban đầu bị sửa đổi nội dung. Cách làm này, do đó, được gọi là làm **tại chỗ** (in-place) hay “ngay trên” danh sách.

11.4 Các danh sách song hành và duyệt với zip

Bạn đã làm Bài tập 6.4 chưa? Nếu chưa thì làm liền bây giờ. Bạn có lẽ dùng cấu trúc if tầng để viết. Khá đơn giản nhưng hơi dài dòng. Ta có thể viết gọn hơn và linh hoạt hơn như sau:

<https://github.com/vqhBook/python/blob/master/lesson11/score.py>

```

1 ds_mốc_điểm = [9, 8, 7, 6, 5, 4, 0]
2 ds_đạt = [True, True, True, True, True, False, False]
3 ds_học_lực = ["Xuất sắc", "Giỏi", "Khá", "Trung bình khá",
4              "Trung bình", "Yếu", "Kém"]
5
6 điểm = float(input("Nhập điểm: "))
7 for i in range(len(ds_mốc_điểm)):
8     if điểm >= ds_mốc_điểm[i]:
9         print("Bạn " + ("đạt" if ds_đạt[i] else "không đạt"))
10        print("Học lực " + ds_học_lực[i])
11        break

```

Không có gì mới ngoài việc ta cố ý sắp các phần tử của 3 danh sách “mốc điểm”, “đạt?”, “học lực” **song hành** hay **đóng hàng** nhau, nghĩa là các phần tử cùng chỉ số ở các danh sách sẽ “đi kèm” với nhau (mà trực quan là chúng cùng một dòng trong bảng “xếp loại” ở Bài tập 6.4).

Ngoài việc gọn hơn thì chương trình trên rất linh động. Chẳng hạn, khi có thay đổi yêu cầu như bỏ loại học lực nào đó; với if tầng, ta phải sửa lại khung chương trình; với cách viết trên, ta giữ nguyên khung chương trình, chỉ đổi danh sách. Khung chương trình được xem là “phần cứng” còn dữ liệu được xem là “phần mềm” của chương trình. Ta đã *tránh việc “code cứng” dữ liệu vào khung chương trình* bằng cách cô lập dữ liệu vào trong danh sách và cập nhật nó khi cần, nhờ đó, ta có một chương trình bền vững và linh hoạt hơn.

Vì các danh sách song hành được dùng nhiều nên Python hỗ trợ hàm zip giúp duyệt qua chúng một cách tự nhiên hơn.⁷ Với zip, ta có thể viết lại vòng for trên đẹp hơn như sau:

```

7 for mốc_điểm, đạt, học_lực in zip(ds_mốc_điểm, ds_đạt,
8   ↪ ds_học_lực):
9     if điểm >= mốc_điểm:

```

⁷Nếu tính ý, bạn sẽ thấy, về ý nghĩa, ta đã dùng các danh sách song hành trong mã sqrt_cal3.py ở Phần 11.1. Ta sẽ nói rõ hơn trong bài của nó, các bộ (Bài 12).

```

9         print("Bạn " + ("đạt" if đạt else "không đạt"))
10        print("Học lực " + học_lực)
11        break

```

Ta đã không phải dùng chỉ số một cách tường minh mà dùng tên gọi nhớ cho các “thành phần” tương ứng của cấu trúc song hành.

11.5 Danh sách lồng

Trong Bài tập 7.9, yêu cầu xuất tam giác Pascal n dòng, tôi cũng đã gợi ý là “hơi khó”. Tôi đã nói thật á, nó ... “hơi khó” thật! Tuy nhiên, bằng cách dùng danh sách, ta có thể làm khá dễ và trực quan như sau:

https://github.com/vqhBook/python/blob/master/lesson11/Pascal_triangle.py

```

1  n = int(input("Nhập số dòng n: "))
2  a = [] # Pascal triangle
3
4  for i in range(n):
5      a.append([1] * (i + 1))
6
7  for i in range(1, n):
8      for j in range(1, i):
9          a[i][j] = a[i-1][j-1] + a[i-1][j]
10
11 for i in range(n):
12     for j in range(i + 1):
13         print("%5d" % a[i][j], end="")
14     print()

```

Chạy thử chương trình với $n = 5$ như sau:

```

===== RESTART: D:\Python\lesson11\Pascal_triangle.py =====
Nhập số dòng n: 5
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1

1 >>> print(a)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
2 >>> temp = a[3]; print((a[3])[1], temp[1], a[3][1])
3 3 3

```

Tôi đã cố ý xuất ra biến `a` trong chương trình. Như kết quả cho thấy, `a` chính là

... tam giác Pascal.⁸ Thật vậy, danh sách *a* có thể được xuất ra đẹp hơn như trong Bài tập 12.13 mà khi đó ta thấy rõ nó là “tam giác”. Thật ra, *a* chỉ là danh sách như bao danh sách khác, chỉ là, các phần tử của *a* cũng là danh sách. Ta, như vậy, gọi *a* là **danh sách lồng** (nested list).⁹ Rất tự nhiên khi ta dùng danh sách dạng này để lưu trữ tam giác Pascal vì tam giác Pascal gồm nhiều dòng, mỗi dòng gồm nhiều số. Có vài kĩ thuật cần lưu ý trong chương trình trên:

- Đầu tiên, ta tạo danh sách *a* theo “cấu trúc” tam giác Pascal *n* dòng. Cụ thể, xuất phát từ danh sách rỗng, sau đó với các dòng *i* từ 0 đến *n* – 1 (nhớ là chỉ số đánh từ 0), ta tạo (và thêm vào *a*) danh sách gồm *i* + 1 số 1. Nhớ rằng cách viết [1] kí hiệu cho danh sách chỉ có một phần tử là 1.
- Sau khi đã có “cấu trúc” tam giác và các số 1 ở lề trái và đường chéo, ta lần lượt “đi qua” các ô của tam giác và điền số là tổng của “ô trên trái” và “ô ngay trên”. Cách viết *a*[*i*][*j*] trông khá lạ nhưng không có gì mới hết, viết rõ hơn là (*a*[*i*])[*j*] vì toán tử chỉ số có tính kết hợp trái. Cách viết này cho thấy rõ là Python lượng giá *a*[*i*] trước để được “dòng *i*” là danh sách các số, sau đó danh sách này lại được chỉ số để lấy về số thứ *j*.
- Sau khi đã điền các số cho tam giác Pascal, ta xuất ra nó bằng vòng lặp lồng ở Dòng 11-14. Chuỗi đặc tả định dạng %5d yêu cầu xuất số nguyên canh phải rộng 5 kí tự (điền kí tự trắng vào trái cho đủ 5 nếu cần).
- Vòng lặp lồng (2 cấp) thường được dùng để xử lý danh sách lồng (2 cấp), tương tự như vòng lặp đơn thường được dùng để xử lý danh sách đơn. Có những trường hợp ta cần dùng danh sách lồng nhiều cấp hơn và khi đó ta sẽ dùng vòng lặp lồng nhiều cấp hơn để xử lý nó.

Cũng như các vòng lặp lồng, các danh sách lồng không có gì mới về ngôn ngữ hay tổ chức vật lý nhưng chúng mang lại sức mạnh lớn về mặt ứng dụng. *Sự phức tạp đến từ lồng ghép.*

11.6 Thao tác trên từng phần tử và bộ tạo danh sách

Một thao tác trên danh sách được gọi là **thao tác trên từng phần tử** (elementwise operation) nếu đầu ra của nó là một danh sách chứa kết quả của thao tác trên từng phần tử theo thứ tự của danh sách đầu vào. Chẳng hạn với danh sách [0, 1, 2, ..., 10], kết quả của thao tác tính giai thừa trên từng phần tử của nó là danh sách [1, 1, 2, 6, ..., 3628800]. Ta luôn có thể tạo “thủ công” danh sách đầu ra bằng cách tính và thêm dần các phần tử từ danh sách rỗng như các phần trên. Tuy nhiên, Python hỗ trợ 2 cách thực hiện ngắn gọn hơn như minh họa sau:

```
1 >>> print(li := list(range(0, 11)))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

⁸Biến *a* nên được đặt tên là *Pascal_triangle* để gợi nhớ, tuy nhiên, tên này hơi dài và tên *a* (viết tắt của array) cũng hay được dùng cho các biến chứa nhiều dữ liệu bên trong.

⁹Ta đã gặp chữ “lồng” này trong if lồng hay vòng lặp lồng.

```

2 >>> import math
3 >>> list(map(math.factorial, li))
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]
4 >>> [math.factorial(i) for i in li]
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]

```

Cách thứ nhất dùng hàm `map` để tạo một “bộ duyệt” “áp” một thao tác lên các phần tử của một danh sách mà khi dùng làm đối số của hàm `list` sẽ được danh sách kết quả (tương tự như `range` và `list`). Cách thứ hai, đơn giản và dễ nhìn hơn, dùng **bộ tạo danh sách** (list comprehension). Nhân tiện, tôi đã dùng toán tử `:=` để “gán và xuất” danh sách `li` mà không phải tốn thêm 1 lệnh nữa.

Ta cũng có thể dùng `map` và/hoặc bộ tạo danh sách để tạo các danh sách lồng. Chẳng hạn, việc tạo cấu trúc của tam giác Pascal có thể được viết gọn là (thay cho các Dòng 2-5 ở mã `Pascal_triangle.py`):

```
a = [[1 for _ in range(i+1)] for i in range(n)]
```

Bạn có thể hơi ngợp với cách viết này, nhưng đó chỉ là vấn đề tâm lý thôi, không có gì mới cả. Ta tạo mảng `a` với các phần tử là kết quả của biểu thức `[1 for _ in range(i+1)]` với `i` chạy từ 0 đến $n - 1$ (`range(n)`). Điều này ứng với việc `a` gồm n dòng mà dòng thứ i ($0 \leq i < n$) là kết quả của biểu thức `[1 for _ in range(i+1)]`. Đi vào biểu thức này (tức là dòng thứ i) thì nó là một danh sách (dòng thứ i là dãy nhiều số) mà danh sách đó lại được tạo bằng bộ tạo. Bộ tạo này, `[1 for _ in range(i+1)]`, thì dễ hiểu rồi, nó tạo danh sách gồm $i + 1$ số 1. Cũng như danh sách lồng (hay if lồng, vòng lặp lồng), bạn sẽ quen thuộc hơn khi luyện tập nhiều.

Bộ tạo danh sách cũng cho phép lọc ra (chọn lựa) các phần tử trong danh sách với phần `if` như minh họa sau:

```

1 >>> print(li := range(0, 10), [e for e in li if e % 2 == 0])
range(0, 10) [0, 2, 4, 6, 8]

```

11.7 Chuỗi

Qua hoạt động và ý nghĩa, tôi đã nói rằng có thể xem chuỗi là danh sách đặc biệt, danh sách các kí tự. Thật ra không phải vậy, chúng là anh em (và là anh em họ, chứ không phải anh em ruột:)). Chúng giống nhau rất nhiều vì đều là dãy (xem Bài tập 12.11), tuy nhiên, chúng khác nhau ở một điều rất quan trọng: *danh sách là khả biến còn chuỗi là bất biến*. Như vậy, hầu như mọi thao tác trên danh sách đều có thể dùng trên chuỗi (đúng hơn là mọi thao tác trên dãy đều có thể dùng trên danh sách và chuỗi), ngoại trừ các thao tác “sửa đổi”, chuỗi cấm tuyệt các thao tác này (đó chính là ý nghĩa của bất biến như ta đã biết trong Phần 10.4).

```

1 >>> s = "hello python"
2 >>> del s[3]
...
TypeError: 'str' object doesn't support item deletion
3 >>> print(li := list(s))
['h', 'e', 'l', 'l', 'o', ' ', 'p', 'y', 't', 'h', 'o', 'n']
4 >>> remove_duplicates(li); "".join(li)
'helo pytn'
5 >>> print(s.title(), ", ", s)
Hello Python , hello python

```

Các “thao tác trên” chuỗi thực ra là tạo chuỗi mới chứ không thay đổi chuỗi ban đầu vì nó là bất biến. Bạn hãy nhìn lại để thấy rằng chuỗi là một dạng dãy bất biến (với các phần tử là các kí tự, mà Python xem là chuỗi đặc biệt).¹⁰ Dĩ nhiên, chuỗi có các đặc trưng riêng của nó như được Python hỗ trợ trong cách mô tả hằng chuỗi, một số thao tác như % định dạng chuỗi, hàm `str`, cách kiểm tra chuỗi con với toán tử `in`, `not in`. Xem thêm Bài tập 11.11 để nhuần nhuyễn hơn về chuỗi.

Tóm tắt

- Danh sách giúp ta quản lý cùng lúc nhiều đối tượng theo tứ tự.
- Lệnh `for` giúp duyệt qua các phần tử của một dãy, như danh sách hay chuỗi.
- Việc dùng danh sách để chứa toàn bộ dữ liệu rồi mới xử lý được gọi là xử lý ngoại tuyến. Kỹ thuật này thường dễ dàng, tiện lợi nhưng kém hiệu quả hơn kỹ thuật xử lý trực tuyến, là cách xử lý ngay khi nhận dữ liệu mà không lưu trữ lại.
- Python hỗ trợ nhiều thao tác trên danh sách qua các lệnh, toán tử, hàm hay phương thức.
- Các danh sách song hành liên kết các “thành phần” của đối tượng theo chỉ số. Hàm `zip` giúp “ghép” các thành phần này.
- Danh sách lồng, tức là danh sách chứa danh sách, hay được dùng để quản lý các “bảng” dữ liệu.
- Hàm `map` và bộ tạo danh sách giúp tạo danh sách là kết quả thực hiện các thao tác trên từng phần tử của danh sách khác.
- Chuỗi và danh sách đều là dãy nên có nhiều thao tác giống nhau. Tuy nhiên, chuỗi là bất biến còn danh sách là khả biến.

¹⁰Nhân tiện, bạn hãy sửa lại mã trong phần duyệt qua các kí tự của chuỗi ở Phần 7.7 và Bài tập 10.2 để mã tự nhiên hơn (không dùng chỉ số mà dùng bản thân kí tự). Đáp án của Bài tập 10.2 cũng được viết lại hay hơn tại https://github.com/vqhBook/python/blob/master/lesson11/turtle_draw.py.

Bài tập

11.1 Tương tự mã `sqrt_cal3.py` trong Phần 11.1, làm lại Bài tập 3.5, 3.6, 3.7.

11.2 Thao tác thu danh sách. Bên cạnh nhóm thao tác trên từng phần tử, một nhóm thao tác cũng hay gặp khác trên danh sách (hay dãy nói chung) là “**thu**” (reduce) danh sách, tức là tính toán một kết quả ra từ các phần tử của danh sách. Thử minh họa sau:

```
1 >>> print(li := list(range(1, 11)))
  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2 >>> import math
3 >>> print(max(li), min(li), sum(li), math.prod(li))
  10 1 55 3628800
4 >>> print(ti := [i <= 10 for i in li])
  [True, True, True, True, True, True, True, True, True, True]
5 >>> print(all(ti), any([4 < i < 5 for i in li]))
  True False
```

Các thao tác trên có chung đặc điểm là tính một giá trị “tích lũy” từ các phần tử của danh sách: `sum` tính tổng các phần tử (tích lũy thao tác +), `math.prod` tính tích (*), `max` tính số lớn nhất (>=), `min` tính số nhỏ nhất (<=), `all` kiểm tra tất cả đều đúng (and) và `any` kiểm tra có phần tử nào đúng (or).

- (a) Làm lại các câu của Bài tập 7.2 bằng kỹ thuật ngoại tuyến, tức là dùng danh sách để lưu trữ các số người dùng nhập, sau đó mới tính toán trên danh sách.

Gợi ý: Tất cả các thao tác này đều là thao tác thu danh sách.

- (b) Dùng bộ tạo danh sách và hàm `all/any` kiểm tra rằng $55^{n+1} - 55^n$ chia hết cho 54 với mọi số tự nhiên n không quá 10000. (Xem Bài tập 2.4.)
- (c) Tự viết lấy hàm thay cho các hàm dựng sẵn hỗ trợ các thao tác thu danh sách: `all`, `any`, `max`, `min`, `sum`.

11.3 Dùng danh sách song hành, làm lại Bài tập 2.2b tại nhiều cặp giá trị: $x = 18$ và $y = 4$, $x = 4$ và $y = 18$, $x = 1$ và $y = 0$, $x = 2$ và $y = 0$, $x = 0$ và $y = 0$, $x = 2$ và $y = 1$, $x = 4$ và $y = 2$.

11.4 Biểu diễn đa thức bằng danh sách các hệ số. Đơn thức một biến x là biểu thức có dạng ax^k , với a là một số thực gọi là **hệ số** (coefficient), k là một số nguyên không âm gọi là số mũ. **Đa thức** (polynomial) một biến x là tổng các đơn thức một biến x , thường được viết theo số mũ tăng dần, và số mũ cao nhất được gọi là **bậc** (degree) của đa thức. Nói cách khác, đa thức P bậc n một biến x , thường được kí hiệu $P_n(x)$ là biểu thức có dạng:

$$P_n(x) = \sum_{k=0}^n a_k x^k = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$$

Rất tự nhiên khi ta dùng danh sách để lưu trữ các hệ số của đa thức, hơn nữa, với qui ước, phần tử ở vị trí (chỉ số) k của danh sách là hệ số của đơn thức có số mũ là k thì một đa thức được hoàn toàn xác định bởi một danh sách các số thực (các hệ số) và ngược lại.¹¹ Chẳng hạn đa thức $P_5(x) = x + 2x^3 - 3x^5$ được biểu diễn bởi danh sách các hệ số là $[0, 1, 0, 2, 0, -3]$ và danh sách $[-1, 0, 0, 0, 0, 0, 0, 0, 0, 1]$ biểu diễn cho đa thức $P_{10}(x) = x^{10} - 1$.¹²

Bằng cách dùng danh sách để biểu diễn cho đa thức, ta có thể viết chương trình Python thao tác trên các đa thức, chẳng hạn, hàm sau tính giá trị của một đa thức tại giá trị được cho của biến:

```

1 https://github.com/vqhBook/python/blob/master/lesson11/polynomial.py
2 def eval_polynomial(P, x):
3     s = 0
4     for k, a in enumerate(P):
5         s += a * x**k
6     return s
7
8 # Tính giá trị của đa thức  $x + 2x^3 - 3x^5$  tại  $x = 2$ 
9 P = [0, 1, 0, 2, 0, -3]
10 print(eval_polynomial(P, 2))

```

Chạy chương trình sẽ được -78 là kết quả của đa thức $x + 2x^3 - 3x^5$ tại $x = 2$. Ví dụ trên cũng minh họa cách dùng hàm dựng sẵn `enumerate` để duyệt qua danh sách, cụ thể, hàm này cho phép lấy chỉ số “đi kèm” với phần tử khi duyệt. Nếu không dùng `enumerate` thì lệnh `for` trên phải viết xấu hơn như sau:

```

3 for k in range(len(P)):
4     a = P[k]
5     s += a * x**k

```

- Viết hàm xuất ra đa thức sắp xếp theo số mũ tăng dần.
- Viết hàm xuất ra đa thức sắp xếp theo số mũ giảm dần.
- Viết hàm tính giá trị của đa thức tại giá trị được cho của biến bằng phương pháp Horner (xem Bài tập 2.6).
- Viết hàm cộng 2 đa thức (trả về đa thức tổng, tức là danh sách các hệ số).

11.5 Biểu diễn số nguyên bằng danh sách các chữ số. Tương tự như chuỗi là danh sách các kí tự (hoặc đa thức là danh sách các hệ số), ta có thể biểu diễn một số nguyên không âm bằng danh sách các chữ số. Cũng như đa thức, cách thuận tiện nhất là lưu trữ các chữ số theo thứ tự tăng dần của hàng với qui ước

¹¹Đây cũng chính là lí do ta chọn cách sắp xếp các đơn thức theo số mũ tăng dần. Một cách sắp xếp khác cũng hay được dùng là theo thứ tự giảm dần số mũ, tuy nhiên, cách này không thuận lợi khi dùng danh sách để biểu diễn.

¹²Nếu chưa rõ thì bạn cần viết lại $P_5(x)$ theo “dạng chuẩn” là $P_5(x) = x + 2x^3 - 3x^5 = 0 + 1x + 0x^2 + 2x^3 + 0x^4 + (-3)x^5$ khi đó sẽ có danh sách các hệ số tương ứng.

hàng đơn vị là hàng 0 lưu ở vị trí 0, hàng chục là hàng 1 lưu ở vị trí 1, hàng trăm là hàng 2 lưu ở vị trí 2, ...¹³ Chẳng hạn, số 2019 được biểu diễn bởi danh sách [9, 1, 0, 2] còn danh sách [2, 0, 1, 9] biểu diễn cho số 9102.

Hiển nhiên, số 0100 được biểu diễn bởi danh sách [0, 0, 1], tức là chỉ lưu trữ từ hàng cao nhất có chữ số khác 0 (đó cũng có thể gọi là bậc của số, tương tự bậc của đa thức). Ta cũng qui ước, số 0 có biểu diễn duy nhất là [0].¹⁴

- (a) Viết hàm nhận một số nguyên không âm và trả về danh sách các chữ số biểu diễn cho nó theo cách trên.
- (b) Khi đã có biểu diễn bằng danh sách của số nguyên không âm thì ta có thể thực hiện một số thao tác trên đó rất đơn giản như đếm số chữ số (dùng hàm `len`) hay tính tổng các chữ số (Bài tập 7.2) (dùng hàm `sum`). Viết lại Bài tập 9.1 bằng cách dùng hàm ở Câu (a) để có danh sách biểu diễn, sau đó thao tác trên danh sách biểu diễn để có kết quả xuất ra như yêu cầu.
- (c) Theo cách biểu diễn này thì số nguyên (không âm) có quan hệ rất gần gũi với đa thức. Thật vậy, bạn hãy suy nghĩ một chút và viết hàm tính lại số nguyên từ danh sách biểu diễn của nó bằng hàm tính giá trị của đa thức (hàm `eval_polynomial`) ở Bài tập 11.4 hoặc hàm tính nhanh hơn ở Bài tập 11.4c.
- (d) Viết hàm cộng 2 số nguyên không âm (trả về tổng). (Tất cả đều dùng danh sách để biểu diễn.)

11.6 Viết lại Bài tập 6.5 bằng cách dùng danh sách song hành thay cho `if` tăng.

11.7 Mở rộng Bài tập 5.7 và Bài tập 5.8 bằng cách viết hàm `bar_chart` (`pie_chart`) vẽ biểu đồ thanh (biểu đồ quạt) từ cặp danh sách song hành là số lượng (tỉ lệ) với nhãn tương ứng. Viết chương trình minh họa việc dùng các hàm này (tương tự yêu cầu của Bài tập 5.8 nhưng không nhất thiết là học lực).

11.8 Dùng bộ tạo danh sách tạo các danh sách:

- (a) Các số chính phương nhỏ hơn 1000 theo thứ tự tăng dần.
- (b) Các số chính phương nhỏ hơn 1000 theo thứ tự giảm dần.
- (c) Các số chia hết cho cả 3, 5 và 7 không quá 1000 theo thứ tự tăng dần.

11.9 Vẽ nổi điểm (plot). Thêm vào module `figures` hàm `plot` nhận tham số `x`, `y` là cặp danh sách song hành mô tả hoành độ và tung độ tương ứng của các điểm. Hàm này vẽ nối các đoạn giữa 2 điểm liên tiếp. Hàm cũng nên có tham số cho lựa chọn vẽ nổi điểm đầu với cuối và chọn màu đường, màu tô.

11.10 Số liệu và thống kê. Phân tích số liệu (Data Analysis) hay **Thống kê** (Statistics) là một ngành khoa học (và nghệ thuật) rất quan trọng.¹⁵ Công việc nghiên cứu bắt đầu bằng việc xác định một hay nhiều **biến** (variable), còn gọi

¹³Cách lưu trữ “hàng thấp trước” này được gọi là **little-endian**, ngược lại với cách lưu trữ “hàng cao trước” gọi là **big-endian**.

¹⁴Một lựa chọn khác, thậm chí hợp lý hơn, là dùng danh sách rỗng `[]` để biểu diễn số 0.

¹⁵Thống kê được đưa vào dạy và học từ rất sớm ở hầu hết các nước. Bạn có thể đọc lại thống kê trong Toán Lớp 7 Tập 2.

là dấu hiệu hay đặc tính (characteristic, feature), quan tâm nào đó trên một **tổng thể** (population) các đối tượng. **Biến định lượng** (quantitative variable) là các đại lượng số như số lượng, chiều cao, cân nặng, ...; còn **biến định tính** (categorical variable) là các tính chất như giới tính (nam/nữ) hay phân loại như học lực (giỏi/khá/trung bình/...), ...

Để khảo sát các biến này, ta tiến hành thu thập số liệu từ một **mẫu** (sample) đại diện cho tổng thể. Sau đó, ta tóm tắt và trình bày số liệu, phân tích và suy luận từ số liệu để đưa ra các nhận định, quyết định. Bước đầu được gọi là **thống kê mô tả** (descriptive statistics) và bước sau là **thống kê suy diễn** (inferential statistics). Như vậy, công việc chủ yếu của thống kê mô tả là tóm tắt dữ liệu bằng các con số và mô tả trực quan dữ liệu bằng các loại biểu đồ (như biểu đồ thanh, biểu đồ quạt ở Bài tập 5.7).

Các con số tóm tắt từ số liệu còn được gọi là các **thống kê** (statistics) của số liệu. Giả sử biến định lượng X có các số liệu là x_1, x_2, \dots, x_n , các thống kê hay dùng cho nó là:

- **Trung bình** (mean):

$$\bar{X} = \frac{\sum_{i=1}^n x_i}{n}$$

- **Trung vị** (median): giá trị nằm giữa các số liệu sau khi sắp.
- **Mode**: giá trị xuất hiện nhiều lần nhất trong các số liệu.
- **Phạm vi** (range): khoảng cách giữa giá trị lớn nhất đến giá trị nhỏ nhất của số liệu.
- **Phương sai** (variance):

$$S_X^2 = \frac{\sum_{i=1}^n (x_i - \bar{X})^2}{n}$$

- **Độ lệch chuẩn** (standard deviation): $s_X = \sqrt{S_X^2}$.

Các thống kê này cho ta một hình dung chung về số liệu. Trung bình, trung vị, mode cho biết giá trị trung tâm mà các số liệu của biến xoay quanh đó còn phạm vi, phương sai, độ lệch chuẩn cho thấy mức độ phân tán của các số liệu so với giá trị trung tâm (và so với nhau).

Số liệu thường được tổ chức thành dạng bảng, gọi là **bảng dữ liệu** (dataset, data table). Trong Python, ta có nhiều cách để lưu trữ và xử lý bảng số liệu. Cách đơn giản là dùng các danh sách song hành cho các biến mà trường hợp 1 biến sẽ là 1 danh sách. Khi đó, việc tóm tắt dữ liệu là việc tính toán từ danh sách này theo mô hình ngoại tuyến. Hơn nữa, thống kê (con số “tổng kết” từ dãy số liệu) chính là kết quả của thao tác thu danh sách số liệu (Bài tập 11.2).

- Tạo module `statistics` cung cấp các hàm tính các thống kê trên từ danh sách chứa mẫu số liệu.

- (b) Viết chương trình minh họa cho người dùng nhập số liệu và dùng module `statistics` để xuất ra các thống kê thông dụng trên.
- (c) Thư viện chuẩn Python có module `statistics` cung cấp hàm tính các thống kê trên (và các thống kê khác). Tra cứu module này (<https://docs.python.org/3/library/statistics.html>) và dùng nó để đối chiếu với module đã viết ở Câu (a).

11.11 Xử lý chuỗi. Ta đã thao tác/xử lý chuỗi rất nhiều cho đến giờ. Tuy nhiên, vì tầm quan trọng đặc biệt của nó nên bài tập này minh họa và yêu cầu bạn thực hành nhiều hơn trên chuỗi.

Chuỗi có khá nhiều thao tác/vấn đề, gồm: mô tả hằng chuỗi (Bài 1), chuỗi Unicode (Phần 11.7), các toán tử/hàm dựng sẵn (Phần 11.7, Phần 11.3), duyệt chuỗi và kiểm tra chuỗi con (Phần 7.7, Bài tập 10.2), chuyển đổi/định dạng (Bài tập 3.7), các phương thức của chuỗi (Phần 11.7) và module `string` (Bài tập 10.2). Ngoài ra một công cụ mạnh để xử lý chuỗi là biểu thức chính qui sẽ được tìm hiểu ở Phần ??.

Vì các thao tác xử lý chuỗi rất phong phú nên bạn chỉ cần nắm bản chất của chuỗi (Phần 11.7) và tra cứu khi cần. Nguồn chính thống là:

- <https://docs.python.org/3/tutorial/introduction.html#strings>,
- <https://docs.python.org/3/library/stdtypes.html#string-methods>,
- <https://docs.python.org/3/library/string.html>,
- <https://docs.python.org/3/library/string.html#formatspec>.

Minh họa sau dùng 2 phương thức của chuỗi là `startswith` và `format`:

```

1 >>> a = 10; b = 20
2 >>> def f(x): return x**2
3
4 >>> [att for att in dir() if not att.startswith("__")]
  ['a', 'b', 'f']
5 >>> print("a = {0}, {2}({1}) = {r}".format(a, b,
  ↪ f.__name__, r=f(b)))
a = 10, f(20) = 400

```

Như vậy, ta có một cách định dạng nữa là dùng phương thức `format` của chuỗi (thay cho toán tử định dạng `%`, f-string và hàm dựng sẵn `format` đã biết, Phần 4.5 và Bài tập 3.7). Tra cứu “sơ lược” các tài liệu trên và thử nghiệm (chạy tương tác với Python) các ví dụ minh họa trong đó.

11.12 Sơ lược lập trình hàm. Python là một ngôn ngữ lập trình đa mô thức (multi-paradigm), nghĩa là nó hỗ trợ nhiều cách thức lập trình khác nhau. Mô thức chủ đạo mà ta dùng tới giờ là **lập trình mệnh lệnh** (imperative programming), nhấn mạnh các bước giải quyết bài toán (how) qua các lệnh (như lệnh gán, lệnh chọn, lệnh lặp, ...). Một mô thức khác cũng được Python hỗ trợ, mô thức **lập trình hàm** (functional programming), thường được xem là “hay ho” hơn và “biểu cảm” hơn do nhấn mạnh đến bài toán cần giải quyết (what).

Chẳng hạn, hàm `eval_polynomial` viết theo mô thức lập trình mệnh lệnh ở Bài tập 11.4 có thể được viết lại theo mô thức lập trình hàm như sau:

```

1 https://github.com/vqhBook/python/blob/master/lesson11/polynomial2.py
2 def eval_polynomial(P, x):
3     return sum([a * x**k for k, a in enumerate(P)])
4
5 # Tính giá trị của đa thức  $x + 2x^3 - 3x^5$  tại  $x = 2$ 
6 print(eval_polynomial([0, 1, 0, 2, 0, -3], 2))

```

Không chỉ bỏ đi lệnh lặp `for`, mã này còn cô đọng và biểu cảm hơn (Dòng 2 gọi nhớ đến kí pháp sigma trong định nghĩa của đa thức, xem Bài tập 11.4). Về mặt kĩ thuật, ta đã dùng thao tác thu danh sách (`sum`) và bộ tạo danh sách.

Mặc dù mỗi mô thức lập trình có ưu điểm và nhược điểm riêng, không thể nói rằng lập trình hàm là hay hơn lập trình mệnh lệnh,¹⁶ nhưng vì Python là ngôn ngữ lập trình **mức cao** (high-level) nên lập trình hàm được ưa chuộng hơn.¹⁷ Ta sẽ tìm hiểu kĩ lập trình hàm ở Phần ??, trước mắt, để lập trình hàm, ta cần dùng tối đa các đặc trưng “cao cấp” trên danh sách (như bộ tạo danh sách, thao tác trên từng phần tử với `map`, các thao tác thu danh sách, lọc danh sách với `filter`) và hạn chế (một cách hợp lý) các lệnh lặp, lệnh chọn.

Viết lại các mã của bài này bằng mô thức “lập trình hàm” (một cách hợp lý).¹⁸

11.13 Đối số dòng lệnh. Khi chạy chương trình bằng cửa sổ lệnh, người dùng có thể cung cấp thông tin cho chương trình qua các **đối số dòng lệnh** (command line argument). Chương trình có thể lấy về các đối số này và dùng nó như trong mã `cmd_args.py` minh họa sau:

```

1 https://github.com/vqhBook/python/blob/master/lesson11/cmd\_args.py
2 import sys
3 print(len(sys.argv), sys.argv)

```

Giả sử file mã được để ở thư mục `D:\Python\lesson11`, chạy Command Prompt hoặc PowerShell từ thư mục đó (xem Bài tập 4.11) và gõ lệnh như hình dưới. Danh sách `sys.argv` gồm chuỗi thứ nhất là tên (hoặc đường dẫn đầy đủ) của file mã và các chuỗi còn lại là các đối số dòng lệnh.

Các chương trình thiên về xử lý (ít tương tác với người dùng) thường nhận dữ liệu từ đối số dòng lệnh và thực thi chương trình mà không đòi nhập dữ liệu thêm từ người dùng. Chẳng hạn, ta có thể sửa file mã `hello.py` ở Phần 4.1 như sau để nó có thể nhận tên từ đối số dòng lệnh:

```

1 https://github.com/vqhBook/python/blob/master/lesson11/hello.py
2 import sys

```

¹⁶Cũng tương tự như kungfu và boxing vậy.

¹⁷Thuần thực lập trình hàm cũng là một tiêu chí đánh giá “lập trình viên Python chân chính” (Pythonistas), một thuật ngữ có phần “tự sướng”.)

¹⁸Rõ ràng, bài tập này quá ... mơ hồ!)

```

3 if len(sys.argv) < 2:
4     name = input("What's your name? ")
5 else:
6     name = " ".join(sys.argv[1:])
7 print("Hello", name)

```

```

Windows PowerShell
PS D:\Python\lesson11> python cmd_args.py
1 ['cmd_args.py']
PS D:\Python\lesson11> python cmd_args.py hi there
3 ['cmd_args.py', 'hi', 'there']
PS D:\Python\lesson11> python cmd_args.py "hi there"
2 ['cmd_args.py', 'hi there']
PS D:\Python\lesson11> python -m cmd_args hi there
3 ['D:\Python\lesson11\cmd_args.py', 'hi', 'there']
PS D:\Python\lesson11>

```

Giả sử file mã trên được để ở thư mục D:\Python\lesson11, chạy Command Prompt hoặc PowerShell từ thư mục đó và gõ lệnh như minh họa sau:

```

Windows PowerShell
PS D:\Python\lesson11> python hello.py
What's your name? Guido van Rossum
Hello Guido van Rossum
PS D:\Python\lesson11> python hello.py Guido van Rossum
Hello Guido van Rossum
PS D:\Python\lesson11> python hello.py "Guido van Rossum"
Hello Guido van Rossum
PS D:\Python\lesson11>

```

Trong IDLE, ta có thể chạy một file mã với đối số dòng lệnh bằng Run → Run... Customized (Shift+F5) và gõ đối số dòng lệnh trong hộp thoại hiện ra (không gõ tên module). Trong VS Code, ta dùng cửa sổ TERMINAL tương tự như PowerShell.

- (a) Module `sys` cung cấp các thông số, cấu hình và thiết lập hệ thống. Tra cứu để rõ hơn về module `sys`: `sys.argv`, `sys.exit`, `sys.modules`, `sys.path`, `sys.platform`, `sys.ps1`, `sys.ps2`, `sys.version`.¹⁹
- (b) Viết lại các chương trình minh họa ở Bài 4 để dùng đối số dòng lệnh và chạy thử bằng Command Prompt/PowerShell, IDLE và TERMINAL của VS Code.

¹⁹Có thể dùng <https://docs.python.org/3/library/sys.html>.

Bài 12

Bộ, tập hợp và từ điển

Bài này tìm hiểu các cấu trúc dữ liệu hay dùng khác trong Python (sau chuỗi và danh sách) là bộ, tập hợp và từ điển. Bộ cũng là dãy, hơn nữa, là dãy bất biến (điều này làm cho bộ gần gũi với chuỗi hơn là danh sách). Tập hợp thì không phải dãy do không có tính thứ tự. Từ điển là tập hợp đặc biệt, cho phép tổ chức phần tử theo cặp “khóa - giá trị”, giúp tra cứu và mở rộng việc quản lý, truy cập không chỉ dựa trên chỉ số (nguyên).

12.1 Bộ

Bộ (tuple) là cấu trúc dữ liệu gồm dãy cố định các phần tử hay **thành phần** (component) có thứ tự. Một bộ gồm n thành phần thường được gọi là **bộ n** (n -tuple) mà trường hợp điển hình là bộ 2 hay **cặp** (pair, couple) và bộ 3 (triple) mà ta đã gặp trong các bài trước. Vì bộ cũng là dãy nên, tương tự như danh sách và chuỗi, mọi thao tác trên dãy đều có thể dùng cho bộ như minh họa sau (xem thêm Bài tập [12.11](#)):

```
1 >>> a = 2, 5
2 >>> print(a, type(a), a[0], a[1])
(2, 5) <class 'tuple'> 2 5
3 >>> print(b := ("a", a, [1]))
('a', (2, 5), [1])
4 >>> x, (y, z), u = b; [v] = u
5 >>> x, y, z, v
('a', 2, 5, 1)
6 >>> b[2] = [1, 2]
...
TypeError: 'tuple' object does not support item assignment
7 >>> b[2][0] = 2; print(b, u, v)
('a', (2, 5), [2]) [2] 1
```

Cách viết “chuẩn” của bộ là liệt kê các thành phần (phân cách bởi dấu phẩy) trong cặp ngoặc tròn. Tuy nhiên, trừ những trường hợp gây nhầm lẫn (như trong lời gọi hàm) thì có thể không dùng cặp ngoặc tròn (vẫn phải dùng dấu phẩy) như ở Dòng 1 hay Dòng 5. Các thành phần của bộ không nhất thiết cùng kiểu (thường là khác kiểu) và có thể có kiểu là bộ (tức là các “bộ lồng”) như ở Dòng 3.

Khác với danh sách, và giống như chuỗi, *bộ là dãy bất biến* (như kết quả của Dòng 6 cho thấy). Tuy nhiên, tính bất biến này là ở bản thân bộ còn các phần tử của nó lại là chuyện khác (Dòng 7). Cụ thể, tính bất biến của bộ (hay các container bất biến khác) có nghĩa là không thể thêm/xóa/sửa các phần tử (thành phần) còn bản thân phần tử có thể thay đổi giá trị nếu nó là khả biến.

Sở dĩ ta không cần dùng cặp ngoặc tròn (để mô tả bộ) trong Dòng 1 và Dòng 5 vì Python tự động thực hiện thao tác **đóng gói bộ** (tuple packing), tức là gom dãy giá trị lại thành bộ, khi phù hợp. Ngược lại, ở Dòng 4, ta đã thực hiện các phép “gán thành phần” hay “tách thành phần”, nhờ thao tác **mở gói dãy** (sequence unpacking). Khi cần thiết (như trường hợp bên trái dấu gán có nhiều biến), Python sẽ “mở” các cấu trúc dữ liệu để lấy các thành phần bên trong (và thực hiện lần lượt các phép gán “đơn” tương ứng). Thử minh họa sau để hiểu rõ hơn:

```

1 >>> a, b = "hi"
2 >>> print(a, b)
  h i
3 >>> a, b = "hello"
  ...
  ValueError: too many values to unpack (expected 2)
4 >>> a, *b = "hello"
5 >>> print(a, b)
  h ['e', 'l', 'l', 'o']
6 >>> a, *b, c = "hello"
7 >>> a + "".join(b) + c
  'hello'

```

Dòng 3 có lỗi thực thi vì, như thông báo lỗi cho thấy, chuỗi bên phải dấu gán có 5 phần tử (5 ký tự) nhiều hơn 2 biến bên trái dấu gán. Đặc biệt, ta dùng ký hiệu * để yêu cầu Python “gom” nhiều phần tử như minh họa ở Dòng 4 và Dòng 6. Dĩ nhiên, ta có thể dùng toán tử truy cập theo chỉ số để “lấy” phần tử của dãy và lát cắt để “lấy nhiều” phần tử.

Thao tác đóng/mở gói tự động chính là bí mật đằng sau phép “gán kép”, tổng quát là **“gán bội”** (multiple assignment), trong mã nguồn `Fibonacci_spiral.py` ở Phần 7.3. Tới giờ, bạn cũng phải hiểu rõ hơn kỹ thuật duyệt qua các danh sách song hành bằng hàm `zip` ở Phần 11.4 như minh họa sau phối bày:

```

1 >>> a = [1, 2, 3]; b = "hello"
2 >>> list(zip(a, b))
  [(1, 'h'), (2, 'e'), (3, 'l')]

```

Lưu ý, zip không yêu cầu các danh sách phải có cùng chiều dài, zip “ghép bộ song hành các phần tử cho đến hết danh sách ngắn nhất”. Cũng vậy, bạn phải hiểu rõ hơn cách hàm enumerate hoạt động trong Bài tập 11.4 như minh họa sau:

```
1 >>> list(enumerate("hello"))
[(0, 'h'), (1, 'e'), (2, 'l'), (3, 'l'), (4, 'o')]
```

Danh sách các bộ cũng có thể được tạo “tường minh” bằng bộ tạo danh sách. Chẳng hạn, ta có thể thay vòng lặp lồng trong mã nguồn `chicken_dog.py` ở Phần 7.4 bằng vòng lặp đơn như sau:

```
1 https://github.com/vqhBook/python/blob/master/lesson12/chicken\_dog.py
2 for x, y in [(i, j) for i in range(37) for j in range(37)]:
3     if x + y == 36 and 2*x + 4*y == 100:
        print(f"Số gà là: {x}, số chó là: {y}.")
```

12.2 Xử lý bảng dữ liệu với danh sách và bộ

Bộ thường được dùng như danh sách cố định (không thêm/xóa/sửa) các phần tử **thuần nhất** (homogeneous), tức cùng kiểu. Khi đó, các thành phần thường được gọi là phần tử (như cách gọi của danh sách) và bộ mô tả nhiều phần tử có thứ tự và cố định. Đây là cách dùng của cặp số ở Phần 2.4, cả 2 phần tử đều là số (chỉ khác là ta phân biệt số thứ nhất, số thứ hai như hoành độ và tung độ của một điểm).¹

Bộ cũng thường gồm các thành phần **hỗn tạp** (heterogeneous), tức không cùng kiểu. Khi đó, bộ thường mô tả một đối tượng với các mặt (đặc trưng, thuộc tính, thành phần) khác nhau được mô tả tương ứng bởi các thành phần (thuật ngữ thành phần cũng thường được dùng trong trường hợp này). Đây là cách dùng của bộ 3 mô tả font chữ cho tham số font của hàm `turtle.write` ở Bài tập 5.6 gồm: tên font, kích thước, kiểu dáng; chẳng hạn, ta đã dùng font ("`Arial`", 30, "`italic`").

Ngoài ra, bộ cũng có thể được dùng theo các cách khác (tùy trường hợp và sự sáng tạo của lập trình viên). Chẳng hạn, ở Bài tập 4.1 ta đã dùng bộ 3 sau toán tử định dạng chuỗi. Trường hợp này, cũng khó nói rằng ta dùng bộ theo cách thứ 2 nói trên. Cách dùng các cấu trúc dữ liệu (nhất là kết hợp chúng với nhau) thật sự rất đa dạng và việc chọn lựa cách tổ chức tốt phụ thuộc vào kinh nghiệm và sự sáng tạo của lập trình viên. Chẳng hạn, bạn dùng cách nào để mô tả một gia đình 2 thế hệ gồm vợ, chồng và con cái? Một cách có thể là dùng bộ ba (vợ, chồng, các con) với các con là danh sách các con (theo “thứ tự sinh”, trường hợp không có con thì là danh sách rỗng).

Một trường hợp vận dụng kết hợp các cấu trúc dữ liệu (danh sách, bộ và các cấu trúc khác) điển hình là khi xử lý các **bảng dữ liệu** (data table) (xem Bài tập 11.10). Mỗi bảng dữ liệu gồm nhiều **dòng** (row) và **cột** (column); mỗi dòng mô tả dữ liệu của một đối tượng; mỗi cột mô tả một biến, còn được gọi là **đặc trưng** (feature)

¹Nhân tiện, hàm `turtle.goto` có thể nhận 2 số x, y hoặc nhận 1 cặp số (x, y) xác định vị trí (hoành độ, tung độ) con rùa di chuyển đến.

hay **thuộc tính** (attribute), quan tâm trên các đối tượng. Chẳng hạn, bảng dữ liệu sau mô tả mã số, tên và điểm (các cột) của các sinh viên (các dòng):²

Mã số	Tên	Điểm
19001	SV A	6.0
19005	SV B	7.5
19004	SV C	5.0
19010	SV D	8.5
19009	SV E	10.0

Một cách đơn giản để tổ chức bảng dữ liệu này là dùng các danh sách song hành (như trong Phần 11.4), mỗi cột là một danh sách và các danh sách (cột) được đóng hàng theo chỉ số (cùng đối tượng). Trường hợp ta chỉ “truy vấn” (tức chỉ đọc) hoặc chỉ xử lý riêng từng cột (chẳng hạn xử lý riêng điểm như tính trung bình điểm, tìm điểm cao nhất) thì cách tổ chức này khá thuận lợi như trong Bài tập 11.10. Trường hợp ta cần thay đổi bảng dữ liệu trên nhiều cột (chẳng hạn, ta cần sắp xếp lại bảng theo thứ tự mã số tăng dần) thì cách tổ chức trên gặp nhiều khó khăn.

Cách tổ chức bảng dữ liệu bằng danh sách song hành có thể nói là “hướng cột”, ngược lại, cách tổ chức “hướng dòng” sẽ dùng một danh sách các đối tượng, mỗi đối tượng là một dòng và được mô tả bằng một bộ với các thành phần là các cột. Cách tổ chức này thường chặt chẽ, rõ ràng và thuận tiện hơn. Đây chính là cách ta đã lựa chọn cho mã `sqrt_cal3.py` ở Phần 11.1. Minh họa sau dùng cách tổ chức này cho bảng dữ liệu trên:

```

1 >>> sv = [(19001, "SV A", 6.0),
2           (19005, "SV B", 7.5),
3           (19004, "SV C", 5.0),
4           (19010, "SV D", 8.5),
5           (19009, "SV E", 10.0)]
6 >>> sv.sort(key=lambda sinh_vien: sinh_vien[0])
7 >>> sv
[(19001, 'SV A', 6.0), (19004, 'SV C', 5.0), (19005, 'SV B',
8           7.5), (19009, 'SV E', 10.0), (19010, 'SV D', 8.5)]
>>> max(sv, key=lambda sinh_vien: sinh_vien[2])
(19009, 'SV E', 10.0)

```

Như kết quả cho thấy, ta đã dễ dàng sắp xếp lại các dòng (sinh viên) theo mã số sinh viên (Dòng 6) hay tìm sinh viên có điểm cao nhất (Dòng 8). Lưu ý, phương thức `sort` của danh sách hay hàm `max` có tham số `key` nhận một hàm 1 tham số mà hàm này sẽ được gọi trên từng dòng (với đối số là dòng đó) để trả về **khóa** (key) so sánh, tức là giá trị dùng để so sánh các dòng với nhau. Ở Dòng 6 ta đã dùng khóa so sánh là mã số (cột 0) để sắp xếp (tăng dần) theo mã số và ở Dòng 8 ta đã dùng khóa so sánh là điểm số (cột 2) để tìm dòng “lớn nhất”. Hơn nữa, ta dùng hàm `lambda` để tránh công kênh (xem lại Phần 8.5).

²Rõ ràng, các tên chỉ có ý nghĩa minh họa.

Thật ra, cột mã số rất đặc biệt, nó giúp định danh sinh viên vì mỗi mã số tương ứng với một và chỉ một sinh viên (lưu ý rằng các sinh viên khác nhau có thể trùng tên). Cột này thường được gọi là **khóa chính** (primary key) của bảng dữ liệu và thường được tách riêng khỏi các cột khác. Do đó, cách tổ chức tốt hơn cho bảng dữ liệu trên là danh sách các bộ có dạng (Mã số, (Tên, Điểm)) như minh họa sau:

```
1 >>> sv = [(19001, ("SV A", 6.0)),
2           (19005, ("SV B", 7.5)),
3           (19004, ("SV C", 5.0)),
4           (19010, ("SV D", 0.5)),
5           (19009, ("SV E", 10.0))]
6 >>> sv
[(19001, ('SV A', 6.0)), ..., (19009, ('SV E', 10.0))]
```

12.3 Hàm có số lượng đối số tùy ý

Ta đã dùng hàm `print` từ rất sớm và biết rằng `print` có khả năng nhận số lượng đối số bất kỳ như trong Phần 1.2, Phần 3.1. Thật tự nhiên khi cho phép `print` nhận bao nhiêu đối số cũng được (kể cả không có) và xuất ra tất cả các đối số (theo thứ tự nhận). Câu hỏi là: tại sao `print` làm được điều này? Tra cứu `print` trong Python Docs (<https://docs.python.org/3/library/functions.html#print>), ta thấy dòng tiêu đề của nó được mô tả như sau:

```
print(*objects, sep=' ', end='\n', file=sys.stdout,
     ↪ flush=False)
```

Ngoại trừ các tham số có tên (và nhận các giá trị mặc định) như ta đã biết, đặc biệt, hàm `print` có khai báo tham số `objects` với dấu sao (*) đằng trước. Điều này có ý yêu cầu Python: khi `print` được gọi thì đóng gói tất cả các đối số (không truyền bằng tên) vào trong một bộ và gán bộ đó cho tham số `objects` trước khi thực thi thân hàm. Như vậy, tham số `objects` là tham số “ảo”, nó không được truyền đối số tương minh như thông thường (không dùng tên được) mà được truyền bằng cơ chế nói trên.

Dĩ nhiên, Python cũng cho phép ta viết các hàm dùng đặc trưng này. Chẳng hạn, ta có thể viết lại hàm `sayhello` trong Phần 8.1 để cho phép “chào cả lô” nhiều người như sau:

```
1 https://github.com/vqhBook/python/blob/master/lesson12/hello.py
2 def sayhello(*names):
3     for name in names:
4         hello_string = " Chào " + name + " "
5         print("=" * (len(hello_string) + 2))
6         print("|" + hello_string + "|")
7         print("=" * (len(hello_string) + 2))
```

```

7 sayhello("Python", "Vũ Quốc Hoàng", "Guido van Rossum")
8

```

Python giúp ta gom các đối số vào bộ names mà trong thân hàm, ta chỉ cần duyệt qua các tên trong bộ names để xuất khung chào cho từng tên. Lưu ý, ta cũng có thể gọi sayhello như trước đây, chẳng hạn sayhello("Python"), khi đó bộ names chỉ có một phần tử (mà hiệu ứng tương đương với tên được dùng trực tiếp như phiên bản cũ của sayhello). Trường hợp không có đối số, lời gọi sayhello(), thì bộ names là rỗng (mà hiệu ứng là “không có gì xảy ra”). Tương tự, bạn hãy viết lại mã hello2.py ở Phần 8.3.³

Về logic, đặc trưng này cũng có thể được hiện thực bằng danh sách (tức là thay vì dùng bộ thì Python dùng danh sách để gom các đối số). Tuy nhiên, triết lý “sâu xa” là danh sách các đối số được gom này là cố định (chẳng hạn, mặc dù có thể không dùng đối số nào đó nhưng hàm không thể bỏ bớt hay thêm đối số mà “nơi gọi” đưa cho: dùng hay không là chuyện của mình, nhưng đưa gì là chuyện của người khác) hơn nữa chúng có thứ tự (theo thứ tự nhận đối số) nên một danh sách bất biến như bộ là hợp lý nhất.

Điều trở trêu của đặc trưng này là khi ta đã có sẵn bộ các đối số và muốn truyền trực tiếp nó cho tham số bộ “ảo” thì Python vẫn máy móc gom nó vào bộ (được bộ 1 phần tử là bộ các đối số) do đó tham số không còn được truyền đối số đúng ý nữa (bộ các đối số). Chẳng hạn, với hàm sayhello trên thì lời gọi hàm sau sẽ bị lỗi (lỗi logic mà trở thành lỗi thực thi trong trường hợp này):

```
sayhello(("Python", "Hoàng", "Guido"))
```

Để giải quyết vấn đề này, Python cung cấp cách gọi hàm đặc biệt sau:

```
sayhello(*("Python", "Hoàng", "Guido"))
```

hay:

```
sayhello(*["Python", "Hoàng", "Guido"])
```

giúp mở gói đối số là dãy thành dãy các đối số. Đặc trưng này không nhất thiết chỉ dùng cho các hàm có số lượng đối số bất kì mà có thể dùng cho mọi hàm. Thử mình họa sau:

```

1 >>> def func(a, b, c, d): return a + b + c + d
2
3 >>> func(1, 2, 3, 4)

```

³Tham số language xác định ngôn ngữ cho tất cả các tên. Trường hợp các tên truyền vào là rỗng thì vẫn giữ giá trị mặc định là "World". Bạn suy nghĩ và làm, trước khi xem đáp án tại <https://github.com/vqhBook/python/blob/master/lesson12/hello2.py>. Bạn cũng suy nghĩ phương án và làm cho trường hợp từng tên được xác định ngôn ngữ riêng, xem đáp án tại <https://github.com/vqhBook/python/blob/master/lesson12/hello3.py>.

```

10
4 >>> print(func(*[1, 2, 3, 4]), func(1, 2, *[3, 4]))
10 10
5 >>> print(func(*[1, 2], 3, 4), func(*[1, 2], *(3, 4)))
10 10
6 >>> func([1, 2, 3, 4])
...
TypeError: func() missing 3 required positional arguments:
      'b', 'c', and 'd'
7 >>> func(*[1, 2, 3])
...
TypeError: func() missing 1 required positional argument: 'd'

```

Ta cũng đã gặp hàm dựng sẵn có khả năng nhận nhiều đối số là `max`, `min` (Phần 3.1). Tuy nhiên, logic nhận đối số của `max/min` phức tạp hơn `print` vì chúng không chỉ muốn tìm `max/min` các phần tử của một dãy (1 đối số là dãy) mà còn muốn tìm `max/min` của dãy các đối số (nhiều đối số với số lượng tùy ý). Ngoài ra, các hàm dựng sẵn khác là `zip` (Phần 11.4) và `map` (Phần 11.6) cũng có khả năng nhận nhiều đối số. Bạn tra cứu các hàm này trong Python Docs (đặc biệt là dòng tiêu đề của chúng) và suy ngẫm thêm.

12.4 Tập hợp

Khác với danh sách, **tập hợp** (set) là **bộ sưu tập** (collection) các phần tử không để ý đến thứ tự và không có các phần tử giống nhau. Cũng như danh sách, *tập hợp là khả biến*, nó có thể được thêm/xóa các phần tử. Các thao tác chính trên tập hợp là kiểm tra thuộc (`in`, `not in`), loại phần tử trùng (giống nhau) và các thao tác Toán (kiểm tra tập con, giao, hợp, hiệu, ...). Lưu ý, vì không có thứ tự nên tập hợp không phải là dãy, do đó không dùng được các thao tác dãy trên tập hợp. Thử minh họa sau:

```

1 >>> a = {1, 2, 1, 3, 1}; print(a, type(a), len(a))
{1, 2, 3} <class 'set'> 3
2 >>> print(1 in a, 2 not in a, {1, 3} <= a)
True False True
3 >>> b = a - {1, 3}; b.add(3); b
{2, 3}
4 >>> for e in b: print(e, end=" ")
2 3
5 >>> b[0]
...
TypeError: 'set' object is not subscriptable

```

Chương trình sau đếm số lần xuất hiện các kí tự trong một chuỗi:

```

1  https://github.com/vqhBook/python/blob/master/lesson12/char\_freq.py
2  def char_freq(text):
3      chars = set(text)
4      print(f"Có {len(chars)} kí tự trong chuỗi {text}")
5      for c in sorted(chars):
6          print(f"{c} xuất hiện {text.count(c)} lần")
7      if chars:
8          most = max([(c, text.count(c)) for c in chars],
9                      key=lambda pair: pair[1])
10         print("Nhiều nhất là %s xuất hiện %d lần" % most)
11 char_freq(input("Nhập chuỗi nào đó: "))

```

Mấu chốt là việc dùng tập hợp (chars) để biết kí tự nào có trong chuỗi (text).

Bạn hãy xem lại chương trình tìm số lớn thứ k trong các số mà người dùng nhập (mã nguồn `k_max.py` trong Phần 11.2). Ta đã giả sử là người dùng nhập các số khác nhau vì trường hợp có số giống nhau sẽ làm phức tạp bài toán rất nhiều, chẳng hạn, nếu người dùng nhập tất cả các số giống nhau thì tất cả các số đều đồng hạng nhất mà không có số lớn thứ 2. Nếu ta vẫn muốn giải quyết trường hợp phức tạp này thì sao (tức là bỏ đi giả định và được chương trình mạnh hơn, tổng quát hơn)? Mấu chốt là việc loại các phần tử trùng khỏi danh sách. Khá phức tạp nếu làm như cách ở mã nguồn `list_remove_dup.py` (Phần 11.3) nhưng rất đơn giản nếu ta dùng tập hợp. Cụ thể, ta chỉ cần loại các phần tử trùng nhau trước khi sắp xếp danh sách bằng lệnh: `nums = list(set(nums))`. Bạn thử liền cho nóng!

Cũng như danh sách, ta có thể dùng **bộ tạo tập hợp** (set comprehension) để tạo tập hợp như minh họa sau:

```

1  >>> set("hello") - set("hi")
   {'l', 'e', 'o'}
2  >>> {x for x in "hello" if x not in "hi"}
   {'l', 'e', 'o'}

```

Cả 2 cách đều giúp ta tạo tập các kí tự có trong chuỗi `hello` mà không có trong chuỗi `hi`. Bạn có thể tra cứu tại <https://docs.python.org/3/library/stdtypes.html#set> để biết thêm các thao tác trên tập hợp.

12.5 Từ điển

Bạn xem lại cuộc bỏ trốn của con rùa với mã nguồn `turtle_escape.py` ở Phần 6.5. Ta có thể viết lại mã này đẹp hơn (và hay hơn) như sau:

```

1  https://github.com/vqhBook/python/blob/master/lesson12/turtle\_escape.py
2  import turtle as t
   import random

```

```

3
4 t.shape("turtle")
5 d = 20
6 actions = {"L": 180, "R": 0, "U": 90, "D": 270}
7 while (abs(t.xcor()) < t.window_width()/2 and
8         abs(t.ycor()) < t.window_height()/2):
9     direction = random.choice("LRUD")
10    t.setheading(actions[direction])
11    t.forward(d)
12 print("Congratulations!")

```

Ở đây, ta dùng một từ điển để mô tả các lựa chọn hành động (actions) của con rùa. **Từ điển** (dictionary) cũng là bộ sưu tập các phần tử, mỗi phần tử là một cặp **khóa: giá trị** (key: value). Từ điển có thể được xem là tập đặc biệt (do đó cách dùng cặp ngoặc nhọn (`{...}`) là hợp lý, hơn nữa, `{}` mô tả một từ điển rỗng chứ không phải tập rỗng) vì nó yêu cầu các khóa là **duy nhất** (unique), nghĩa là không trùng nhau giữa các phần tử. Từ điển cũng có thể được xem là danh sách với khả năng truy cập không chỉ bằng chỉ số mà bằng khóa.

Về logic, mỗi phần tử của từ điển có một đặc trưng đặc biệt nhận diện hay định danh phần tử (như mã số sinh viên, số chứng minh nhân dân). Đặc trưng đặc biệt này không nhất thiết là số mà có thể là chuỗi hay bất kì dữ liệu bất biến nào khác và nó chính là khóa xác định duy nhất phần tử.⁴ Các đặc trưng (thành phần) còn lại của phần tử tạo thành giá trị (value) của phần tử đó. Trường hợp đơn giản, giá trị có thể là dữ liệu đơn (như số nguyên trong mã trên), trường hợp phức tạp nó có thể là dữ liệu bất kì mà thường là bộ. Thật ra, hay hơn nữa, ta thay Dòng 9 trên thành:

```

9 direction = random.choice(list(actions))

```

Lệnh này cũng cho thấy từ điển là tập và ta cần chuyển nó thành dãy để có thể dùng hàm `random.choice` chọn một phần tử ngẫu nhiên.⁵ Tương tự, bạn hãy viết lại mã nguồn `turtle_draw.py` ở Phần 7.7 đẹp hơn với từ điển (và các cấu trúc dữ liệu, kĩ thuật đã học tới giờ). Gợi ý, bạn có thể kiểm tra một khóa có trong từ điển hay không bằng toán tử `in`.⁶ Cũng vậy cho mã nguồn `turtle_draw.py` ở Phần 8.6 (gợi ý, xem lại Bài tập 10.2).⁷

Để hiểu rõ từ điển hơn, bạn xem lại cách tổ chức dữ liệu cho bảng ở Phần 12.2. Ta đã đưa ra 3 lựa chọn, và bây giờ, có một lựa chọn tốt hơn nữa, là dùng từ điển với key là mã số và value là bộ gồm các đặc trưng còn lại. Về cốt lõi thì đây chính

⁴Về mặt kĩ thuật, đối tượng dùng làm khóa phải là bất biến để nó có thể “**băm được**” (hashable) vì Python dùng **bảng băm** (hash table) để tổ chức từ điển (giúp “tra” nhanh các phần tử).

⁵Bí mật bên trong của `random.choice` là phát sinh một chỉ số ngẫu nhiên và chọn phần tử tại chỉ số đó nên ta cần dãy để có thể truy cập theo chỉ số.

⁶Đáp án: https://github.com/vqhBook/python/blob/master/lesson12/turtle_draw.py.

⁷Đáp án: https://github.com/vqhBook/python/blob/master/lesson12/turtle_draw2.py. Mã này có dùng kĩ thuật bao đóng hàm ở Bài tập 10.2.

là ý tưởng của cách thứ 3. Thực vậy, ta có thể dùng hàm `dict` để chuyển danh sách trên thành từ điển như minh họa sau (tiếp tục từ Phần 12.2):

```
1 >>> sv = [(19001, ("SV A", 6.0)),
2           (19005, ("SV B", 7.5)),
3           (19004, ("SV C", 5.0)),
4           (19010, ("SV D", 0.5)),
5           (19009, ("SV E", 10.0))]
6 >>> dict(sv)
{19001: ('SV A', 6.0), 19005: ('SV B', 7.5), 19004:
('SV C', 5.0), 19010: ('SV D', 0.5), 19009: ('SV E', 10.0)}
```

Có một cách tổ chức bảng dữ liệu khác là dùng **từ điển lồng** (nested dictionary) dạng `{<Mã số>: {"Tên": <Tên>, "Điểm": <Điểm>}}` như minh họa sau:

```
1 >>> mã_số = [19001, 19005, 19004, 19010, 19009]
2 >>> tên = ["SV A", "SV B", "SV C", "SV D", "SV E"]
3 >>> điểm = [6.0, 7.5, 5.0, 0.5, 10.0]
4 >>> sv = {m: {"Tên": t, "Điểm": d} for m, t, d in zip(mã_số,
5               ↪ tên, điểm)}; sv
{19001: {'Tên': 'SV A', 'Điểm': 6.0}, 19005: {'Tên': 'SV B',
'Điểm': 7.5}, 19004: {'Tên': 'SV C', 'Điểm': 5.0}, 19010: {'Tên':
'SV D', 'Điểm': 0.5}, 19009: {'Tên': 'SV E', 'Điểm': 10.0}}
6 >>> for m in sorted(sv): print(m, sv[m])
19001 {'Tên': 'SV A', 'Điểm': 6.0}
19004 {'Tên': 'SV C', 'Điểm': 5.0}
19005 {'Tên': 'SV B', 'Điểm': 7.5}
19009 {'Tên': 'SV E', 'Điểm': 10.0}
19010 {'Tên': 'SV D', 'Điểm': 0.5}
7 >>> ms_max = max(sv, key=lambda k:sv[k]["Điểm"])
>>> print(ms_max, sv[ms_max])
19009 {'Tên': 'SV E', 'Điểm': 10.0}
```

Các lệnh trên minh họa một số kỹ thuật trên từ điển:

- Dòng 4 dùng **bộ tạo từ điển** (dictionary comprehension) để tạo từ điển (ở đây là từ các danh sách song hành). Cách này thường được dùng trong một số trường hợp đơn giản. Linh hoạt hơn, ta có thể tạo từ điển bằng cách thêm/xóa/sửa trên từ điển với các lệnh, hàm, toán tử, phương thức tương ứng (bạn tra cứu thêm tại <https://docs.python.org/3/library/stdtypes.html#typesmapping>).
- Dòng 5 duyệt qua từ điển theo thứ tự khóa (ở đây là mã số sinh viên) tăng dần với hàm `sorted` và truy cập value bằng key (ở đây là truy cập tên, điểm qua mã số sinh viên).
- Dòng 6 tìm key (ở đây là mã số sinh viên) của value “lớn nhất” (ở đây là điểm cao nhất).

Các minh họa trên cho thấy *sức mạnh của sự kết hợp*. Danh sách có thể chứa các phần tử là danh sách, tức là danh sách lồng. Tương tự, ta có bộ lồng, từ điển lồng hay danh sách lồng từ điển, từ điển lồng danh sách với số mức lồng và cách lồng tùy ý. Tóm lại là có muôn vàn kiểu, biến hóa khôn lường! Vấn đề là ta cần chọn được cách phù hợp nhất để tổ chức và xử lý dữ liệu cho bài toán của ta. Đây là nghệ thuật mà sự thành công được quyết định bởi kinh nghiệm, sức sáng tạo và cá tính của lập trình viên.

12.6 Đối số từ khóa

Bạn đã thấy rằng hàm có thể nhận số lượng đối số bất kì truyền theo vị trí bằng cách dùng tham số ảo 1 dấu *. Python cũng cho phép hàm nhận số lượng đối số bất kì truyền theo tên bằng cách dùng tham số ảo 2 dấu * (**). Thử minh họa sau:

```

1 >>> def f(*a, b=None, **c): print(a, b, c)
2
3 >>> f()
  () None {}
4 >>> f(1)
  (1,) None {}
5 >>> f(1, 2, b=3)
  (1, 2) 3 {}
6 >>> f(b=3, 1, 2)
  SyntaxError: positional argument follows keyword argument
7 >>> f(b=3, a=1)
  () 3 {'a': 1}
8 >>> f(b=3, a=1, c=2, d="hello", e=1.5, f=(4, 5))
  () 3 {'a': 1, 'c': 2, 'd': 'hello', 'e': 1.5, 'f': (4, 5)}

```

Bạn nghiền ngẫm và thử nghiệm thêm để hiểu rõ. Lưu ý rằng các tham số a, c là “ảo” vì ta không trực tiếp truyền đối số cho nó được. Python tự xử lý các đối số và truyền cho a, c. Cụ thể, Python đóng gói tất cả các đối số truyền theo vị trí (không phải cho các tham số thông thường) vào một bộ và truyền cho a; Python cũng đóng gói tất cả các đối số truyền theo tên (không phải cho các tham số thông thường) vào một từ điển và truyền cho b.

Như vậy, cặp <name>=<value> trong danh sách đối số sẽ được Python truyền <value> cho tham số (thông thường) có tên <name> hoặc nếu không có tham số nào tên <name> thì Python sẽ đóng gói vào từ điển với khóa là chuỗi <name> và giá trị là <value> (hoặc báo lỗi nếu không có tham số ảo từ điển nào được khai báo). Đây là lí do ta có thuật ngữ **đối số từ khóa** (keyword argument) vì khóa là một chuỗi (từ). Tối đây, ta chính thức khôi phục danh nghĩa cho thuật ngữ này!⁸

⁸Tôi đã nói thuật ngữ này không ra gì ở các bài trước. Nhưng nó là thuật ngữ tốt ... nếu bạn hiểu rõ!

Ở Phần 12.3, ta đã thấy cách mở gói đối số là dãy thành dãy các đối số truyền theo vị trí trong lời gọi hàm với toán tử *. Tương tự như vậy, Python hỗ trợ việc mở gói đối số là từ điển thành các đối số truyền theo tên trong lời gọi hàm với toán tử **. Thử minh họa sau:

```

1 >>> def func(a, b, c, d):
2     return a + b + c + d
3
4 >>> func(a=1, b=2, c=3, d=4)
10
5 >>> func(**{"a": 1, "b": 2, "c": 3, "d": 4})
10
6 >>> func(1, 2, **{"c": 3, "d": 4})
10
7 >>> func(**{"a": 1, "b": 2}, c=3, d=4)
10
8 >>> print(*[1, 2, 3, 4], **{"sep": " ", "end": "\n"})
1234

```

Hàm `print` nhận dãy tùy ý các đối số vị trí và 2 đối số từ khóa là `sep` và `end`. Phương thức `format` của chuỗi là trường hợp điển hình dùng cả các đối số vị trí và từ khóa với số lượng tùy ý (xem lại Bài tập 11.11 và tra cứu phương thức `format`). Cũng tìm hiểu và dùng thử phương thức `pen` của con rùa.

Tóm tắt

- Bộ là dãy bất biến.
- Python tự động thực hiện việc đóng gói bộ, tức là gom dãy đối tượng vào bộ, khi cần thiết.
- Mở gói dãy (danh sách, bộ, chuỗi) là việc tách riêng từng phần tử của dãy như trong cách gán kép.
- Bảng dữ liệu có thể được biểu diễn bằng các danh sách song hành cho từng cột, hay bằng danh sách các bộ cho từng dòng, hay danh sách các từ điển.
- Python hỗ trợ hàm nhận nhiều đối số vị trí bằng cách đóng gói bộ các đối số này vào “tham số bộ ảo 1 dấu sao”. Dấu * khi dùng với đối số dãy trong lời gọi hàm sẽ mở dãy này thành dãy các đối số.
- Tập hợp là bộ sưu tập các phần tử không để ý đến thứ tự và không có các phần tử giống nhau.
- Từ điển là bộ sưu tập các cặp khóa: giá trị (key: value).
- Python hỗ trợ hàm nhận nhiều đối số từ khóa bằng cách đóng gói các đối số này vào “tham số từ điển ảo 2 dấu sao”. Dấu ** khi dùng với đối số từ điển trong lời gọi hàm sẽ mở từ điển này thành các đối số truyền theo tên.

- Ta có thể dùng các bộ tạo để tạo tập hợp và từ điển, tương tự như bộ tạo danh sách.

Bài tập

12.1 Hàm trả về nhiều giá trị. Với các cấu trúc dữ liệu đã học, ta dễ dàng viết các hàm trả về nhiều giá trị. Một cách tự nhiên, ta có thể dùng danh sách để chứa tất cả các giá trị cần trả về và trả về nó làm giá trị trả về (duy nhất) của hàm. Tuy nhiên, như đã bàn trong Phần 12.3 về việc Python dùng bộ mà không phải danh sách để gom các đối số vị trí vào “tham số ảo 1 dấu *”, kết quả trả về của hàm là bất biến ở nơi nhận. Như vậy, một cách triết lý hơn, ta nên dùng bộ để chứa các giá trị trả về của hàm như cách Python hay làm:

```
1 >>> divmod(10, 3)
(3, 1)
2 >>> (1.25).as_integer_ratio()
(5, 4)
3 >>> import turtle; turtle.forward(100); turtle.position()
(100.00, 0.00)
```

Viết hàm nhận các hệ số của một phương trình bậc 2 và trả về các nghiệm của phương trình đó (xem lại Phần 6.6 để biết thuật toán). Minh họa việc dùng hàm này để giải một phương trình bậc 2 do người dùng nhập. Lưu ý cách viết bộ rỗng () và bộ chỉ có 1 thành phần (xxx,).

12.2 Vector 2D. Cặp số (x, y) thường được gọi là **vector 2 chiều** (2-dimensional vector). Chúng thường được dùng để mô tả tọa độ điểm trên mặt phẳng như tọa độ (vị trí) của con rùa. Ta hay gọi thành phần thứ nhất (x) là hoành độ còn thành phần thứ hai (y) là tung độ.⁹

Cho $v = (x, y)$, $v' = (x', y')$ là 2 vector và k, θ là số, các phép toán sau thường được dùng trên các vector:

- Cộng: $v + v' = (x + x', y + y')$
- Trừ: $v - v' = (x - x', y - y')$
- Nhân với số k : $k \times v = v \times k = (kx, ky)$
- Nhân vector: $v \times v' = xx' + yy'$
- Tính độ dài: $|v| = \sqrt{x^2 + y^2}$
- Quay góc θ : $\text{rot}_\theta v = (x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)$

Viết module `vec2D` cung cấp các hàm với tham số và giá trị trả về thích hợp cài đặt các thao tác trên. Sau đó, viết chương trình minh họa việc dùng module này.

⁹Thật vậy, bạn có thể gọi `turtle.goto(10, 20)` hoặc `turtle.goto((10, 20))` để di chuyển con rùa đến điểm (10, 20). Sau đó, lời gọi `turtle.postion()` sẽ trả về vector (cặp số) (10, 20) là vị trí hiện tại của con rùa.

12.3 Đa giác. Đa giác (polygon) là hình “kín” với “biên” là các đoạn thẳng. Các biên còn được gọi là **cạnh** (edge) và điểm giao giữa các cạnh được gọi là **đỉnh** (vertex) hay **góc** (corner) của đa giác. Một đa giác có n đỉnh ($n \geq 3$) được gọi là n -giác. Chẳng hạn, tam giác hay tứ giác là các đa giác có 3 đỉnh, 4 đỉnh.

- (a) Tiếp tục Bài tập 8.8, bổ sung vào module `figures` hàm `polygon` vẽ và tô đa giác với danh sách đỉnh được cho, mỗi đỉnh là một cặp số (tức là vector 2 chiều).
- (b) Dùng hàm `polygon` trên, vẽ các Hình (b), (c) ở Bài tập 5.4.

12.4 Tự viết lấy các thao tác đã được Python hỗ trợ sau:

- (a) Hàm `enumerate` cho phép “điểm danh” các phần tử của một dãy (xem lại Phần 12.1). Viết hàm `my_enumerate` trả về danh sách các bộ tương tự như `enumerate`.
- (b) Hàm `zip` cho phép tạo danh sách các bộ từ việc “ghép các danh sách song hành” (xem lại Phần 12.1). Viết hàm `my_zip` trả về danh sách các bộ tương tự như `zip`.
- (c) Ta đã dùng bộ tạo danh sách các bộ trong mã `chicken_dog.py` ở Phần 12.1. Viết hàm `my_tuple_com`, tương tự, trả về danh sách các bộ.

12.5 Viết lại hàm ở Bài tập 5.8 với tham số nhận vào là từ điển với các cặp key: value là nhân: số lượng.

12.6 Trong Bài tập 4.1, ta đã biết cách dùng hàm `eval` để tính giá trị của một biểu thức chứa biến theo giá trị được cho của các biến. Để mô tả giá trị được cho của các biến ta có thể dùng từ điển với cấu trúc `{"biên_1": giá_trị_1, "biên_2": giá_trị_2, ...}`. Viết hàm `eval_expression` lượng giá một biểu thức chứa biến (được cho bằng chuỗi) theo giá trị của các biến (được cho bằng cấu trúc từ điển như đã nói). Ví dụ, lời gọi:

```
eval_expression("(10 + x)*y", {"x": 0, "y": 1})
```

sẽ trả về giá trị 10.

12.7 Mở rộng hàm ở Bài tập 8.1d để nhận danh sách:

- (a) Các bộ có dạng (`<Romeo>`, `<Juliet>`)
- (b) Các từ điển có dạng `{"Romeo": <tên>, "Juliet": <tên>}`

và xuất ra lần lượt đoạn trích “Roméo và Juliet” cho từng cặp tên tương ứng.

12.8 Dùng đặc trưng hàm nhận số lượng đối số bất kì:

- (a) Viết lại hàm trong các Bài tập 8.1c và 8.1d để “xuất cả lô”.
- (b) Viết lại hàm trong Bài tập 9.5 để tìm ước số chung lớn nhất của nhiều số nguyên dương.¹⁰
- (c) Mở rộng các câu của Bài tập 12.4 cho nhiều danh sách.

¹⁰`math.gcd` cũng dự định cung cấp khả năng này (nhận nhiều đối số) ở phiên bản 3.9.

(d) Tự viết lấy hàm thay cho các hàm dựng sẵn `max` và `min`.

12.9 Biểu diễn đa thức bằng từ điển. Trong Bài tập 11.4 ta đã dùng danh sách (các hệ số) để biểu diễn và xử lý đa thức. Một cách biểu diễn đa thức khác là dùng từ điển các cặp key: value là số mũ; hệ số. Chẳng hạn đa thức $P_5(x) = x + 2x^3 - 3x^5$ được biểu diễn bởi từ điển `{1: 1, 3: 2, 5: -3}` và từ điển `{0: -1, 10: 1}` biểu diễn cho đa thức $P_{10}(x) = x^{10} - 1$. Rõ ràng cách biểu diễn này tốt hơn cách dùng danh sách các hệ số cho trường hợp **đa thức “thưa”** (sparse polynomial) tức là đa thức có nhiều hệ số bằng 0 (mà trong cách viết thông thường ta bỏ qua không viết số hạng tương ứng).

Làm lại Bài tập 11.4 theo cách biểu diễn mới này của đa thức.

12.10 Thống kê 1 biến định tính. Trong bài tập 11.10, ta đã dùng danh sách các số để lưu trữ dữ liệu của 1 biến định lượng và viết hàm tính toán các thống kê (con số tổng kết từ dãy số liệu) trên danh sách số liệu này. Ta cũng dùng danh sách để lưu trữ dữ liệu của 1 biến định tính nhưng có nhiều lựa chọn hơn cho kiểu của phần tử, chẳng hạn:

- Luận lý cho các biến có 2 giá trị như giới tính (Nam/Nữ) với `True` là Nam, `False` là Nữ.
- Chuỗi cho các biến có nhiều giá trị như nhóm máu (A/B/AB/O).
- Các số nguyên “nhỏ” cho các biến có giá trị “sắp thứ tự” như cấp học (1/2/3).

Dĩ nhiên, có thể có nhiều lựa chọn khác nhau như có thể dùng chuỗi cho giới tính ("Nam" cho Nam, "Nữ" cho Nữ) hoặc dùng số (1 là Nam, 0 là Nữ); dùng chuỗi cho xếp loại học lực (Xuất sắc/Giỏi/Khá/...) hoặc dùng số (1 là Xuất sắc, 2 là Giỏi, ...).

Viết hàm tính các “thống kê” sau đây cho dữ liệu của 1 biến định tính (nghĩa là hàm nhận danh sách mô tả dữ liệu của 1 biến định tính được chọn phù hợp như mô tả trên, tính toán và trả về kết quả phù hợp):

- Số mức (level):** số lượng các giá trị khác nhau của biến. (trả về số đếm)
- Các mức:** các giá trị khác nhau của biến. (trả về danh sách)
- Yếu vị (mode):** giá trị xuất hiện nhiều lần nhất. (trả về cặp gồm giá trị và tần số, tức là số lần xuất hiện giá trị đó trong dữ liệu).
- Bảng tần số (frequency table):** các giá trị khác nhau cùng với tần số. (trả về danh sách các cặp (giá trị, tần số) sắp giảm dần theo tần số).

12.11 Kiểu dãy. Thuật ngữ **dãy** (sequence) ám chỉ tập các đối tượng **có thứ tự** (ordered).¹¹ Tính chất này cũng dẫn đến việc một phần tử có thể xuất hiện nhiều lần trong dãy. Python hỗ trợ các **kiểu dãy** (sequence type) dựng sẵn là danh sách (list), chuỗi (str), bộ (tuple) mà ta đã học và **kiểu range** sẽ biết rõ hơn ở Phần 14.8.¹² Các kiểu dãy được phân loại tiếp dựa trên tính bất biến/khả biến như ta đã biết.

¹¹Điển hình trong Toán là dãy số như dãy Fibonacci ở Phần 5.3.

¹²Ngoài ra còn có các **kiểu dãy byte** (binary sequence type) mà ta không tìm hiểu trong tài liệu này.

Tất cả các kiểu dãy đều hỗ trợ một tập chung các thao tác trên dãy và các thao tác đặc thù riêng của từng kiểu cụ thể. Thao tác quan trọng nhất, dĩ nhiên, là truy cập phần tử với chỉ số hay lát cắt ([...]), sau đó là các thao tác kiểm tra thành viên (`in/not in`), lấy chiều dài (`len`), ... Lưu ý, thao tác kiểm tra thành viên đã được “mở rộng” để kiểm tra **chuỗi con** (substring) trên chuỗi. Sau đây là vài minh họa:

```

1 >>> s = "hello"; print(min(s), s.count("l"))
e 2
2 >>> print("e" in s, "ell" in s, "elo" in s)
True True False
3 >>> print(r := range(1, 6), type(r), list(r))
range(1, 6) <class 'range'> [1, 2, 3, 4, 5]
4 >>> print(r[0], r[-2:], 6 in r)
1 range(4, 6) False
5 >>> print(bool(r), bool(""), bool([False]), bool(range(0)))
True False True False

```

Đặc biệt, các **dãy rỗng** (empty sequence), tức là dãy không có phần tử, được xem là `False` (ngược lại, dãy không rỗng được xem là `True`). Bạn có thể tra cứu đầy đủ về kiểu dãy tại <https://docs.python.org/3/library/stdtypes.html#typeseq>.

- (a) Tổng kết, hệ thống lại các kiểu dãy.
- (b) Thử nghiệm kiểu `range` để thấy rằng nó cũng là dãy như bộ (nó cũng là bất biến).

12.12 Tổ chức không gian tên bằng từ điển. Ta đã học về không gian tên ở Phần 10.6. Rõ ràng, cách tự nhiên nhất để tổ chức chúng là dùng từ điển, tập các cặp `key: value` với `key` là tên và `value` là đối tượng mà tên tham chiếu đến (name: object). Chẳng hạn:

- thao tác định nghĩa tên (lệnh `gán`, `import`, `def`) sẽ tạo mới phần tử `name: object` hoặc đổi giá trị `object` mới cho `name` đã có.
- thao tác dùng tên là tra `object` của `name` hoặc phát sinh lỗi nếu chưa có `name`. Thao tác này được thực hiện rất nhanh nhờ kỹ thuật tra khóa của từ điển (kỹ thuật băm).

Quả thật, Python dùng từ điển để tổ chức các không gian tên. Ta có thể dùng hàm `globals` để lấy về không gian tên toàn cục, hàm `locals` lấy về không gian tên cục bộ hiện tại (có thể là không gian tên toàn cục), hàm `vars` (hay thuộc tính `__dict__`) lấy về không gian tên (các thuộc tính) gắn với đối tượng.

Dùng lại file mã `amodule.py` của Phần 10.8, giả sử nó được đặt ở thư mục `D:\Python\lesson12`, chạy IDLE từ đầu và tương tác như sau:

```

1 >>> import os; os.chdir(r"D:\Python\lesson12")
2 >>> import amodule

```

```

3 >>> [k for k in vars(amodule) if not k.startswith("__")]
  ['math', 'sqrt', 'a', 'b']
4 >>> amodule.__dict__["a"] = 0; amodule.a
  0
5 >>> del os
6 >>> def func(a):
7     b = 100
8     print(locals())
9     print(globals())
10
11 >>> func(50)
  {'a': 50, 'b': 100}
  {'__name__': '__main__', ... }
12 >>> globals()["a"] = 200; a
  200
13 >>> dict(filter(lambda k: not k[0].startswith("__"),
  ↪   globals().items()))
  {'amodule': ..., 'func': ..., 'a': 200}

```

Tra cứu và thử nghiệm các hàm `globals`, `locals`, `vars` và thuộc tính `__dict__` để hiểu rõ hơn.

12.13 Xuất đẹp các cấu trúc dữ liệu. Các cấu trúc dữ liệu phức tạp như danh sách, bộ, từ điển, tập hợp thường gồm nhiều phần tử lồng chứa nhiều mức nên chuỗi biểu diễn chúng thường khó đọc. Python cung cấp module `pprint` hỗ trợ việc “xuất đẹp” (pretty-print) các dữ liệu này như minh họa sau:

```

1 >>> import keyword
2 >>> print(keyword.kwlist)
  ['False', 'None', 'True', ..., 'while', 'with', 'yield']
3 >>> import pprint
4 >>> pprint.pprint(keyword.kwlist, width=50, compact=True)
  ['False', 'None', 'True', 'and', 'as', 'assert',
   ...,
   'while', 'with', 'yield']
5 >>> a = [[1 for j in range(i+1)] for i in range(10)]
6 >>> pprint.pprint(a)
  [[1],
   [1, 1],
   ...,
   [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]

```

Ta cũng có thể dùng hàm `pprint.pformat` để lấy về chuỗi định dạng của dữ liệu thay vì xuất nó ra.

(a) Tìm hiểu module `pprint` và các hàm `pprint.pprint`, `pprint.pformat`.

(b) Dùng pprint để xuất đẹp các dữ liệu phức tạp trong Bài 11 và Bài 12.

12.14 Cấu trúc dữ liệu túi. Túi (bag) là bộ sưu tập các đối tượng không để ý đến thứ tự các phần tử (như tập hợp) nhưng cho phép các phần tử xuất hiện nhiều lần (như dãy). Túi có thể được xem là mở rộng của tập hợp để cho phép các phần tử xuất hiện nhiều lần nên nó còn được gọi là **multiset**. Túi cũng có thể được xem là một từ điển với key là phần tử và value là số lần xuất hiện của phần tử trong tập (các phần tử không xuất hiện thì không có trong từ điển). Đây cũng là cách Python cung cấp túi qua kiểu collections.Counter. Thử minh họa sau:

```
1 >>> from collections import Counter
2 >>> print(b := Counter("hello"))
Counter({'l': 2, 'h': 1, 'e': 1, 'o': 1})
3 >>> print(b["l"], b["i"], b.most_common(1))
2 0 [('l', 2)]
4 >>> print(list(b.elements()), set(b))
['h', 'e', 'l', 'l', 'o'] {'o', 'e', 'l', 'h'}
```

Kiểu Counter giúp ta giải quyết nhiều bài toán liên quan đến đếm (như tên gọi của nó) một cách đơn giản. Chẳng hạn, mã char_frequency.py ở Phần 12.4 có thể được viết lại đơn giản hơn như sau:

https://github.com/vqhBook/python/blob/master/lesson12/char_freq2.py

```
1 from collections import Counter
2
3 def char_freq(text):
4     chars = Counter(text)
5     print(f"Có {len(chars)} kí tự trong chuỗi {text}")
6     for c in sorted(chars):
7         print(f"{c} xuất hiện {chars[c]} lần")
8     if chars:
9         most = chars.most_common(1)[0]
10        print("Nhiều nhất là %s xuất hiện %d lần" % most)
11
12 char_freq(input("Nhập chuỗi nào đó: "))
```

(a) Tìm hiểu kiểu collections.Counter tại <https://docs.python.org/3/library/collections.html#collections.Counter>.

(b) Dùng Counter làm lại Bài tập 12.10.

Bài 13

Case study 3: Ngẫu nhiên, mô phỏng và trực quan hóa

Cuộc sống đầy rẫy ngẫu nhiên. Đó là lí do mà người ta hay chúc nhau may mắn. Thật khó lí giải tại sao một người mua vé số hoài không trúng, trong khi có người chỉ mua 1 lần đã trúng độc đắc. Do ăn ở chẳng?! Cũng thật khó hiểu khi trong số hàng triệu người, ta lại gặp được người mà ta mong đợi. Do duyên số chẳng?! Do gì đi nữa thì *ngẫu nhiên là bản chất và cũng là hương vị của cuộc sống*. Bài này tìm hiểu ngẫu nhiên và cách mô phỏng nó trên máy (Python). Như là một phương tiện để trực quan hóa việc mô phỏng, thư viện Matplotlib và Pillow cũng được tìm hiểu.

13.1 Vẽ biểu đồ với Matplotlib

Matplotlib là một **thư viện vẽ biểu đồ** (plotting library) hay rộng hơn là **thư viện trực quan hóa** (visualization library) miễn phí và phổ biến trên Python. Khác với `turtle` là module nằm trong thư viện chuẩn của Python, Matplotlib là thư viện ngoài, cần cài đặt trước khi dùng (xem lại Phần 3.2). Trang chủ của Matplotlib được để ở địa chỉ <https://matplotlib.org/>. Bạn hãy lướt sơ qua đó, chẳng hạn, thưởng lãm các biểu đồ minh họa được vẽ bằng Matplotlib (<https://matplotlib.org/gallery/index.html>).

Với máy Windows, sau khi cài đặt Python, ta có thể cài Matplotlib bằng cách mở cửa sổ lệnh (Command Prompt) và gõ lệnh:¹

```
py -m pip install -U matplotlib
```

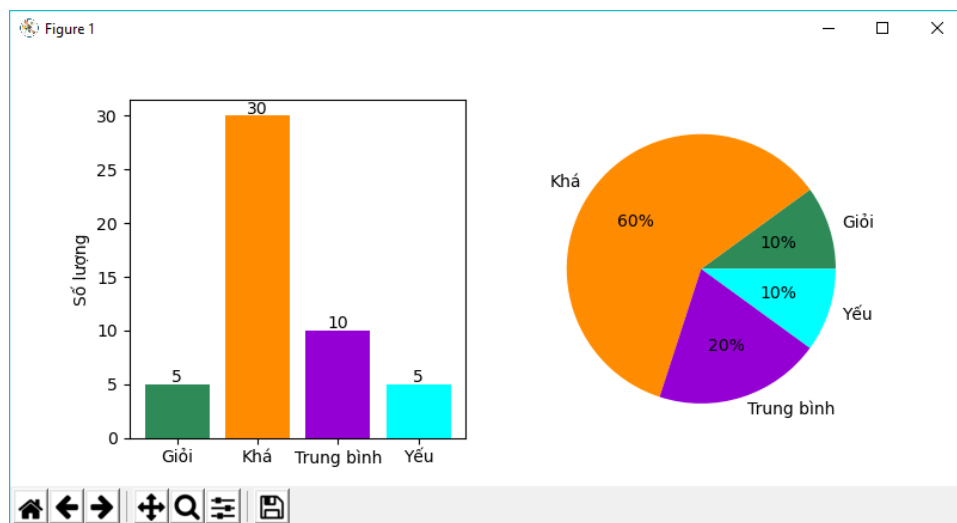
như hình minh họa sau:

¹Để có thể dùng lệnh này, lúc cài Python bạn phải đánh dấu chọn “Add Python ... to PATH” như đã nhấn mạnh ở Phần 1.1. Nếu không bạn có thể cài lại Python và đánh dấu chọn. Để mở Command Prompt, bạn có thể nhấn nút Search ở thanh Taskbar và gõ Command Prompt.

218 BÀI 13. CASE STUDY 3: NGẪU NHIÊN, MÔ PHỎNG VÀ TRỰC QUAN HÓA

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

C:\Users\USER>py -m pip install -U matplotlib
Collecting matplotlib
  Downloading matplotlib-3.2.1-cp38-cp38-win32.whl (9.0 MB)
    | 9.0 MB 3.3 MB/s
Collecting numpy>=1.11
  Downloading numpy-1.18.3-cp38-cp38-win32.whl (10.8 MB)
    | 10.8 MB 103 kB/s
Collecting cycler>=0.10
```



Ta chỉ cần cài đặt Matplotlib một lần duy nhất và sau đó có thể dùng nó như các module trong thư viện chuẩn. Chương trình sau đây minh họa việc dùng Matplotlib để vẽ biểu đồ thanh và biểu đồ quạt ở Bài tập 5.7:

```
https://github.com/vqhBook/python/blob/master/lesson13/chart.py
1 import matplotlib.pyplot as plt
2
3 labels = ["Giỏi", "Khá", "Trung bình", "Yếu"]
4 freqs = [5, 30, 10, 5]
5 clrs = ["seagreen", "darkorange", "darkviolet", "cyan"]
6
7 fig, (ax1, ax2) = plt.subplots(1, 2)
8
9 ax1.bar(labels, freqs, color=clrs)
10 for i, f in enumerate(freqs):
11     ax1.text(i, f+0.2, str(f), horizontalalignment='center')
12 ax1.set_ylabel("Số lượng")
13
```



```

14 ax2.pie(freqs, labels=labels, autopct="%.0f%%", colors=clrs)
15
16 plt.show()

```

Hình trên cho thấy kết xuất của chương trình. Sau đây là vài lưu ý:

- Dòng 1 nạp module `pyplot` của `matplotlib` và đặt lại tên là `plt`.²
- Dòng 3-5 dùng 3 danh sách song hành (xem lại Phần 11.4) để tổ chức dữ liệu gồm tên loại học lực, số lượng học sinh và màu của phần biểu đồ tương ứng. Các tên màu này được đặt theo qui định của Matplotlib (xem tại https://matplotlib.org/3.1.0/gallery/color/named_colors.html#references). Ta cũng có thể dùng chuỗi mã màu hay bộ 3 số thực màu RGB như trong con rùa (xem Bài tập 5.3).
- Dòng 7 gọi hàm `subplots` (của `matplotlib.pyplot`) để tạo 2 vùng biểu đồ bên trong cửa sổ với cách bố trí 1 dòng (đối số đầu), 2 cột (đối số thứ hai). Hàm này trả về cặp đối tượng với đối tượng đầu (`fig`) tham chiếu đến cả hình và đối tượng sau là dãy các biểu đồ con tương ứng. Ta đã dùng kĩ thuật mở gói dãy (xem lại Phần 12.1) để lấy các đối tượng `ax1`, `ax2` tương ứng với biểu đồ con trái và phải.
- Dòng 9-12 vẽ biểu đồ thanh lên biểu đồ con trái:
 - Phương thức `bar` vẽ biểu đồ thanh với tham số thứ nhất là nhãn các thanh, tham số thứ 2 là chiều cao các thanh và tham số tên `color` xác định màu tương ứng của các thanh.
 - Dòng 10-11 duyệt qua từng thanh và xuất chuỗi số lượng cho từng loại học lực tương ứng bằng phương thức `text` với tham số thứ nhất là chỉ số thanh, thứ 2 là vị trí đặt chuỗi, thứ 3 là chuỗi cần xuất và tham số tên `horizontalalignment` xác định cách canh lề ngang.
 - Phương thức `set_ylabel` đặt tiêu đề cho trục đứng.
- Dòng 14 vẽ biểu đồ quạt lên biểu đồ con bên phải: phương thức `pie` vẽ biểu đồ quạt với tham số thứ nhất là giá trị các vùng (sẽ được tự động tính tỉ lệ), tham số `labels` xác định nhãn các vùng, tham số `autopct` xác định chuỗi định dạng cho tỉ lệ xuất ra bên trong vùng (ở đây là xuất ra dạng phần trăm làm tròn đến chữ số đơn vị) và tham số `colors` xác định màu.
- Dòng 16 gọi hàm `show` để hiển thị cửa sổ cùng với các biểu đồ kết quả. Hàm này cũng thực hiện vòng lặp thông điệp mà khi cửa sổ được đóng lại thì hàm kết thúc và chương trình kết thúc theo. Hoạt động này tương tự như hoạt động của cửa sổ con rùa mà ta đã biết ở Phần 8.6.

Cửa sổ biểu đồ của Matplotlib (cửa sổ có tiêu đề “Figure 1” ở trên) có các nút lệnh mà ta có thể nhấn vào để thao tác thêm với biểu đồ như các nút thu phóng, tinh chỉnh biểu đồ, ... và đặc biệt là nút Save cho phép ta lưu biểu đồ lại thành tập tin ảnh với nhiều lựa chọn định dạng phổ biến. Bạn thử thao tác với các nút này.

²Tên viết tắt này là qui ước hay dùng của cộng đồng Python.

13.2 Đồ họa tương tác với Matplotlib

Nếu như chương trình trên minh họa kỹ thuật đồ họa **tĩnh** (static) thì chương trình sau đây minh họa kỹ thuật đồ họa **tương tác** (interactive) của Matplotlib:

https://github.com/vqhBook/python/blob/master/lesson13/interactive_plot.py

```

1 import matplotlib.pyplot as plt
2 from matplotlib.widgets import Slider
3
4 def update(val):
5     t = val = int(val)
6     if val == 1:
7         tex = "y = x"
8     elif val > 1:
9         tex = "y = $x^{%d}$" % val
10    else:
11        t = 1/(-val + 2)
12        tex = r"y = $x^{\frac{1}{%d}}$" % (-val + 2)
13
14    ys = [x**t for x in xs]
15    ax.clear()
16    ax.plot(xs, ys, color="r")
17    ax.set_title(tex)
18
19 fig, ax = plt.subplots()
20 plt.subplots_adjust(left=0.1, right=0.9, bottom=0.2)
21 slider_ax = plt.axes([0.1, 0.1, 0.8, 0.03])
22 slider = Slider(slider_ax, "val", -14, 15, valinit=1,
23               ↪ valstep=1, valfmt='%d')
24 slider.on_changed(update)
25
26 xs = [i/1000 for i in range(1001)]
27 ax.set_ylim(0, 1)
28 ax.set_xlim(0, 1)
29 update(1)
30 plt.show()

```

Chương trình trên vẽ đồ thị của hàm số: $y = x^t$ với t nhận các giá trị nguyên 1, 2, 3, ... hoặc phân số $\frac{1}{2}$, $\frac{1}{3}$, ... Lưu ý $x^{\frac{1}{2}} = \sqrt{x}$, $x^{\frac{1}{3}} = \sqrt[3]{x}$, ... Trường hợp $t = 1$ ta có đồ thị của hàm số $y = x$ là một đường thẳng. Đặc biệt, trong cửa sổ biểu đồ, ta có thể kéo thanh trượt (vạch màu đỏ trong thanh val) để điều chỉnh giá trị của t mà từ đó đồ thị sẽ thay đổi theo. Sau đây là giải thích thêm cho mã:

- Dòng 19 tạo đối tượng hình và biểu đồ như ví dụ trước (lời gọi subplots không đổi số tạo 1 biểu đồ con).

- Dòng 20 điều chỉnh vùng vẽ của biểu đồ để chứa thêm thanh trượt bên dưới. Các tham số được cho tính theo tỉ lệ của hình. Ví dụ `bottom=0.2` là để biên dưới của biểu đồ cách cạnh dưới cửa sổ là 20% chiều cao cửa sổ.
- Dòng 21 tạo thêm một vùng biểu đồ để chứa thanh trượt.
- Dòng 22 tạo thanh trượt (để người dùng kéo chọn giá trị cho t) bằng hàm `Slider` trong module `matplotlib.widgets` (được nạp ở Dòng 2).
- Đặc biệt, theo mô hình lập trình hướng sự kiện (xem lại Phần 8.6), Dòng 23 đăng kí thủ tục `update` làm trình xử lý sự kiện người dùng kéo thanh trượt với tham số `val` xác định giá trị tương ứng của thanh trượt.
- Trong hàm `update` ta thay đổi biểu đồ theo giá trị `val` của thanh trượt. Cụ thể ta tính t từ `val` với qui các số dương xác định mũ như thông thường còn các số 0, -1, -2, ... xác định tương ứng các giá trị của t là $\frac{1}{2}$ (căn bậc 2), $\frac{1}{3}$ (căn bậc 3), $\frac{1}{4}$ (căn bậc 4), ...
- Dòng 15 xóa kết quả của đồ thị cũ và Dòng 16 vẽ đồ thị mới ứng với giá trị của t bằng phương thức `plot`. Phương thức này nhận 2 danh sách song hành, danh sách đầu là hoành độ và danh sách sau là tung độ của các điểm trên đường cần vẽ. Matplotlib nối các điểm này thành đường (tương tự kĩ thuật vẽ đường bất kì ở Phần 9.5 và Bài tập 11.9).
- Tiêu đề biểu đồ cho biết phương trình đường đang vẽ. Đặc biệt, ta có thể xuất công thức Toán trong tiêu đề (hay văn bản nói chung) bằng cách dùng LaTeX (tham khảo <https://matplotlib.org/tutorials/text/mathtext.html>).

Các chương trình trên là khung chương trình minh họa các bước cơ bản để vẽ biểu đồ với Matplotlib. Thật sự, *Matplotlib cung cấp khả năng vẽ biểu đồ rất đẹp và phong phú*. Bạn chắc chắn phải tra cứu thêm khi cần (lục trong trang chủ của Matplotlib hoặc search Google với tiền tố Matplotlib đằng trước).

13.3 Tung đồng xu

Nếu hay xem bóng đá, bạn sẽ biết tầm quan trọng của đồng xu: nó được trọng tài dùng để quyết định cầu môn của 2 đội khi bắt đầu trận đấu.³ Cơ bản, số phận của 2 đội (cầu môn) được quyết định bởi kết quả đồng xu ra ngửa hay sấp. **Tung đồng xu** (coin flipping, coin tossing) thường được dùng để phân xử các tình huống như thế này là vì 2 kết quả có **khả năng** hay **cơ hội** (chance) như nhau. Nói một cách bình dân thì kết quả ngửa/sấp là “5 ăn 5 thua”. Nói theo kiểu Toán, cụ thể là **lý thuyết xác suất** (probability theory), thì **xác suất** (probability) của **biến cố** (event) được ngửa hay sấp đều là 50% (tức là 0.5) trong **thí nghiệm ngẫu nhiên** (random experiment) tung đồng xu.

Có chắc là khả năng ra ngửa/sấp là “5 ăn 5 thua” (xác suất được ngửa hay sấp đều là 50%)? Tương truyền, vì tò mò nên Buffon và Pearson (các nhà Toán học xác

³Đồng xu còn được dùng để chọn đội đá penalty trước hay thậm chí chọn đội thắng khi đá luân lưu như việc Ý thắng Liên Xô nhờ may mắn trong kết quả tung đồng xu ở giải Châu Âu năm 1968.

suất nổi tiếng) đã thử kiểm chứng giả thuyết này bằng cách tung đồng xu rất nhiều lần và đếm số lần được ngửa và sấp. Nếu số lần ngửa và sấp “xêm xêm nhau” thì “có vẻ” giả thuyết này đúng. Lưu ý, ta không mong đợi số lần ngửa đúng bằng số lần sấp vì nếu vậy việc tung đồng xu không còn ngẫu nhiên (chẳng hạn, nếu lần đầu ra ngửa thì đoán được lần thứ hai ra sấp).

Ý nghĩa thực sự của trực giác trên là: nếu ta tung đồng xu nhiều lần thì tỉ lệ ra ngửa (số lần được ngửa chia cho tổng số lần tung) sẽ gần 50% và càng gần hơn khi tung nhiều lần hơn. Nếu ta cũng tò mò như Buffon hay Pearson thì sao, ta cũng thử tung đồng xu nhiều lần như vậy nhưng đỡ nhọc công hơn nhiều nhờ Python với chương trình sau:

— https://github.com/vqhBook/python/blob/master/lesson13/coin_toss.py —

```

1 import random
2
3 N = int(input("Bạn muốn tung bao nhiêu lần: "))
4 coins = [random.randint(0, 1) for i in range(N)]
5 N_ngua = sum(coins)
6 N_sap = N - N_ngua
7 print(f"Số lần ngửa là {N_ngua}/{N} ({N_ngua/N*100:.2f}%)")
8 print(f"Số lần sấp là {N_sap}/{N} ({N_sap/N*100:.2f}%)")

```

Kỹ thuật trên được gọi là **mô phỏng** (simulation) vì ta “tái hiện” một quá trình thực tế nào đó trên máy (ở đây là việc tung đồng xu). Cụ thể hơn, nó được gọi là **mô phỏng ngẫu nhiên** (stochastic simulation) vì quá trình được mô phỏng là ngẫu nhiên. Việc lặp lại nhiều lần không có gì khó với máy, quan trọng là việc mô phỏng một lần tung. Ta đã chọn “ngẫu nhiên” số 1 (mà ta xem là ngửa) hoặc 0 (mà ta xem là sấp) bằng hàm `random.randint` (xem thêm Bài tập 13.18).

Chạy thử chương trình trên bạn sẽ thấy: với số lần tung (N) ít thì kết quả biến động nhiều (thay đổi trong các lần chạy khác nhau), với số lần tung nhiều (từ vài chục ngàn trở lên) thì kết quả sẽ ổn định hơn. Để thấy rõ sự “hội tụ” của tần suất (tỉ lệ) đến xác suất khi số lần tung lớn, ta viết lại chương trình trên dùng kỹ thuật **hoạt họa** (animation) của Matplotlib như sau:

— https://github.com/vqhBook/python/blob/master/lesson13/coin_toss2.py —

```

1 import random
2 import matplotlib.pyplot as plt
3 from matplotlib.animation import FuncAnimation
4
5 def update(frame):
6     global N, freqs
7
8     N += 1
9     freqs[random.randint(0, 1)] += 1
10    fr = [f/N for f in freqs]
11
12    ax.clear()

```

```

13     ax.set_ylim(0, 1)
14     ax.bar(["Sấp", "Ngửa"], fr, color=["red", "blue"])
15     ax.text(0, fr[0] + 0.01, f"{fr[0]*100:.2f}%",
16            horizontalalignment='center')
17     ax.text(1, fr[1] + 0.01, f"{fr[1]*100:.2f}%",
18            horizontalalignment='center')
19     ax.set_title(f"Lần tung thứ {N}")
20
21     N = 0
22     freqs = [0, 0] # Số lượng sấp, ngửa
23
24     fig, ax = plt.subplots()
25     ani = FuncAnimation(fig, update, interval=1000//24)
26     plt.show()

```

Hàm `FuncAnimation` giúp đều đặn gọi một hàm để vẽ lại khung hình mới. Ý tưởng và hoạt động tương tự như kỹ thuật hoạt họa ta đã bàn ở Phần 9.6 (bạn xem lại phần này và tra cứu thêm Matplotlib).

Thật ra, chuyện không đơn giản như vậy! Câu hỏi về khả năng ra ngửa/sấp phải hỏi cho đồng xu cụ thể nào đó (thậm chí là người tung, địa điểm tung, thời điểm tung, ... cụ thể). Với “đồng xu của Python” ta đã thấy “5 ăn 5 thua” nhưng đồng xu của Buffon hay Pearson thì không biết!⁴

13.4 Trò chơi may rủi

Chắc hẳn ở quê ai cũng đã từng chơi “Bầu Cua” mỗi khi Tết đến. Ngoại trừ chuyện biến tướng thành cờ bạc thì đây có lẽ là một nét văn hóa Tết của người Việt giống như việc đốt pháo, lì xì,... Trò chơi có bàn cờ gồm 6 ô ứng với 6 “linh vật” thân quen của người Việt là Tôm, Cua, Bầu, Cá, Gà và Nai. Sau khi người chơi “đặt” (cược) tiền vào các ô này, nhà cái sẽ rung 3 viên xúc xắc có 6 mặt ứng với 6 linh vật trên. Nếu trong 3 viên xúc xắc “có” (xuất hiện) linh vật mà người chơi đã cược, họ sẽ lấy lại tiền cược và nhà cái phải “chung” (trả) số tiền bằng số lần xuất hiện linh vật đó nhân với số tiền cược. Nếu linh vật người chơi đặt không xuất hiện thì nhà cái sẽ “ăn” (lấy) số tiền cược cho linh vật đó. Chẳng hạn nếu nhà cái rung ra 3 Bầu (cả 3 xúc xắc đều ra mặt Bầu) thì nhà cái sẽ ăn tất cả số tiền trong 5 linh vật còn lại và người chơi đặt Bầu sẽ nhận lại số tiền đã đặt cho Bầu cùng với số tiền chung của cái là 3 lần số tiền đã đặt cho Bầu. Bạn đọc thêm bài viết “Lắc bầu cua” trên trang Wikipedia Việt để biết thêm.⁵

Nếu đã từng chơi Bầu Cua, bạn chắc hẳn thắc mắc câu hỏi sau: nếu có 3 tờ 1 đồng và bạn muốn đặt cả 3 tờ thì nên chọn phương án nào trong 3 phương án sau:

⁴Rất nhiều “kĩ xảo gian lận” có thể được tiến hành mà dẫn đến khả năng ra ngửa/sấp không còn như nhau mà có lợi hơn cho người tung như trong cờ bạc gian lận!

⁵https://vi.wikipedia.org/wiki/Lắc_bầu_cua

224 BÀI 13. CASE STUDY 3: NGẪU NHIÊN, MÔ PHỎNG VÀ TRỰC QUAN HÓA

- Đặt cả 3 tờ vào cùng 1 ô,
- Đặt 2 tờ cùng ô và 1 tờ ô khác,
- Đặt 3 tờ vào 3 ô khác nhau.

Rõ ràng, nếu không có gian lận thì 6 ô như nhau nên câu hỏi trên cũng là: nên đặt hết 3 tờ ô Tôm, hay 2 Tôm 1 Cua, hay 1 Tôm 1 Cua 1 Bầu?

Bạn sẽ học cách trả lời bằng Toán (với lý thuyết xác suất) sau nhưng hiện giờ ta có thể dùng Python để trả lời. Rất đơn giản, ta sẽ chơi nhiều lần và xem thử số tiền có được khi chơi theo 1 trong 3 cách trên thì cách nào nhiều hơn. Sau đây là module Python mô phỏng.

```
1 import random
2
3 def chung_tien(ds_cược, mặt_ra):
4     cược = sum(ds_cược.values())
5     for lv in set(LINH_VẬT) - set(mặt_ra):
6         cược -= ds_cược.get(lv, 0)
7     for lv in mặt_ra:
8         cược += ds_cược.get(lv, 0)
9     return cược
10
11 def bầu_cua():
12     mặt_ra = random.choices(LINH_VẬT, k=3)
13     t1 = chung_tien({"TÔM": 3}, mặt_ra)
14     t2 = chung_tien({"TÔM": 2, "CUA": 1}, mặt_ra)
15     t3 = chung_tien({"TÔM": 1, "CUA": 1, "BẦU": 1}, mặt_ra)
16     return t1, t2, t3
17
18 def trung_bình(N):
19     k_quả = [bầu_cua() for _ in range(N)]
20     return [sum([k[c] for k in k_quả])/N for c in [0, 1, 2]]
21
22 LINH_VẬT = ["TÔM", "CUA", "BẦU", "CÁ", "GÀ", "NAI"]
```

Chạy module để có định nghĩa các hàm và tương tác tiếp như sau:

```
===== RESTART: D:\Python\lesson13\bau_cua.py =====
1 >>> trung_bình(500_000)
  [2.764992, 2.763612, 2.762578]
2 >>> trung_bình(500_000)
  [2.75832, 2.758818, 2.763098]
3 >>> trung_bình(500_000)
  [2.761044, 2.762686, 2.76386]
```

Kết quả cho thấy, trung bình trong 1 lần chơi, mỗi phương án cho số tiền khoảng 2.76 đồng. Rất khó nói cách nào cho số tiền cao hơn vì kết quả vẫn còn biến động (lúc cái nào cao, lúc cái kia cao). Dĩ nhiên, ta có thể tăng số lần chơi thử (tham số `N` của hàm `trung_bình`) để có kết quả mô phỏng chính xác hơn, nhưng bù lại, ta phải đợi lâu hơn (với số lần thử 500,000 thì thời gian mô phỏng trên máy của tôi cũng đã hơi lâu).

Thật ra, kết quả phân tích chính xác bằng lý thuyết xác suất cho thấy, trung bình một lần chơi, cả 3 cách đều cho số tiền như nhau là $\frac{199}{72} \approx 2.764$ đồng.⁶ Như vậy, ta không cần phân vân giữa 3 phương án chơi vì phương án nào cũng như nhau. Hơn nữa, ta không nên chơi vì nếu không chơi thì ta còn 3 đồng mà chơi (dù theo cách nào) thì chỉ còn lại (trung bình) khoảng 2.76 đồng. Mà như vậy, sau khoảng 13 lần chơi là ta sạch vốn (3 đồng ban đầu).

Sau đây là giải thích thêm cho mã trên:

- Hàm `bầu_cua` mô phỏng 1 lần chơi: Dòng 12 mô phỏng việc tung 3 con xúc xắc bằng hàm `random.choices`. Dòng 13, 14, 15 lần lượt tính số tiền còn lại khi đặt theo 3 phương án trên. Hàm này trả về bộ 3 số tiền đã tính.
- Hàm `chung_tiền` tính số tiền người chơi có được khi đặt cược theo `ds_cược` và 3 mặt ra của 3 xúc xắc là danh sách `mặt_ra`. Vì số tiền đặt “thừa” nên ta dùng từ điển để mô tả số tiền cược trên 6 ô với các phần tử dạng linh vật: số tiền cược. Dòng 4 tính tổng số tiền cược trên 6 ô. Dòng 5-6 tương ứng việc cái ăn tiền ở các ô không có mặt ra. Dòng 7-8 tương ứng việc cái chung tiền cho các ô có mặt ra.
- Hàm `trung_bình` tính trung bình số tiền có được theo 3 phương án trong `N` lần chơi: Dòng 19 mô phỏng `N` lần chơi và ghi nhận số tiền kết quả trong danh sách `k_quả`. Dòng 20 tính trung bình số tiền có được theo mỗi phương án bằng cách tính tổng số tiền có được theo mỗi phương án và chia cho `N`. Tôi đã dùng kỹ thuật “lập trình hàm” để viết ngắn gọn hàm này. Bạn xem lại Phần 11.12 và nghiền ngẫm để hiểu rõ.

13.5 Ngẫu nhiên và trực giác

Thật ra, nếu có “trực giác tốt”, ta có thể nhận ra rằng 3 phương án đặt cược ở trên đều “cho số tiền như nhau” vì qui tắc chung tiền của nhà cái là tính theo mỗi tờ tiền (nếu ra ô tờ tiền đó đặt thì được chung thêm 1 tờ còn nếu không thì mất tờ vốn) mà không phụ thuộc các tờ tiền khác (bao nhiêu tờ và đặt ô nào). Như vậy, không cần phải mô phỏng hay tính toán lý thuyết, ta có thể kết luận là 3 tờ đặt 3 ô nào không quan trọng.

Tuy nhiên, *trực giác của ta thường bị sai khi đối mặt với các vấn đề liên quan đến ngẫu nhiên*. Ta rất dễ bối rối, mơ hồ hay thậm chí bị trực giác đánh lừa trong

⁶Việc phân tích xác suất để cho ra kết quả trên cũng không phải khó lắm nhưng vượt quá phạm vi Toán Lớp 9 nên tôi không bàn thêm. Ta có thể gặp nhau trong một tài liệu khác khi bạn đã vững Toán hơn. Bạn cũng nên học kỹ xác suất ở Toán Lớp 11 vì nó rất quan trọng.

nhiều tình huống như tình huống sau đây, thường được gọi là “Monty Hall problem”:⁷

- Có 3 cửa #1, #2, #3; 1 chiếc xe và 2 con dê được để ngẫu nhiên vào 3 cửa.
- Người chơi chọn 1 cửa (chẳng hạn #1).
- Người dẫn chương trình biết cửa nào có gì. Người dẫn chọn và mở cửa có dê trong 2 cửa còn lại (chẳng hạn #3).
- Người dẫn hỏi người chơi có muốn đổi lựa chọn không (giữ #1 hay chọn #2).

Người chơi nên giữ hay đổi để khả năng được xe cao hơn? Có người cho rằng nên đổi, có người cho rằng giữ hay đổi không quan trọng. Bạn thì sao?

Cho đến bây giờ, câu hỏi này vẫn còn gây nhiều tranh cãi. Thậm chí nhiều người sau khi đọc lời giải đúng bằng Toán vẫn cảm thấy không thuyết phục vì nó không “khớp” với trực giác. Nếu như lời giải bằng Toán không thuyết phục được bạn thì chương trình sau lại làm được điều đó vì nó hoàn toàn dựa vào trực giác.

https://github.com/vqhBook/python/blob/master/lesson13/Monty_Hall.py

```

1 from random import choice
2 import matplotlib.pyplot as plt
3 from matplotlib.animation import FuncAnimation
4
5 def Monty_Hall(doors={"#1", "#2", "#3"}):
6     car_door = choice(list(doors))
7     choice_door = choice(list(doors))
8     open_door = choice(list(doors - {choice_door, car_door}))
9     op_door = choice(list(doors - {choice_door, open_door}))
10
11     return car_door == choice_door, car_door == op_door
12
13 def update(_):
14     global N, freqs
15
16     N += K
17     results = [Monty_Hall() for _ in range(K)]
18     freqs[0] += sum([v for v, _ in results])
19     freqs[1] += sum([v for _, v in results])
20     fr = [f/N for f in freqs]
21
22     ax.clear()
23     ax.set_ylim(0, 1)
24     ax.bar(["Giữ", "Đổi"], fr, color=["red", "blue"])
25     ax.text(0, fr[0], f"{fr[0]*100:.2f}%",
26            horizontalalignment='center')
```

⁷Tình huống này được đặt ra dựa trên một game show truyền hình nổi tiếng của Mỹ là “Let’s Make a Deal” do Monty Hall dẫn chương trình. Bạn có thể đọc thêm tại https://en.wikipedia.org/wiki/Monty_Hall_problem.


```

27     ax.text(1, fr[1], f"{fr[1]*100:.2f}%",
28             horizontalalignment='center')
29     ax.set_title(f"Lần chơi thứ {N}")
30
31 K = 1000 # số lần chơi mỗi lần update
32 N = 0
33 freqs = [0, 0] # Số lượng thắng khi Giữ, Đổi
34
35 fig, ax = plt.subplots()
36 ani = FuncAnimation(fig, update, interval=1000//24)
37 plt.show()

```

Chương trình trên rất dễ thuyết phục vì nó tái hiện lại các bước của game show (các gạch đầu dòng trong mô tả trên). Cụ thể:

- Dòng 6: ban tổ chức chọn ngẫu nhiên cửa đặt xe (các cửa còn lại đặt dê).
- Dòng 7: người chơi chọn ngẫu nhiên 1 cửa.
- Dòng 8: người dẫn mở ngẫu nhiên 1 cửa trong số các cửa không phải cửa người chơi đã chọn và cửa chứa xe.
- Dòng 9: nếu lựa chọn đổi cửa thì người chơi sẽ chọn ngẫu nhiên 1 cửa trong số các cửa còn lại (không phải cửa chọn ban đầu và cửa người dẫn đã mở).
- Dòng 11: nếu cửa mở ban đầu là cửa chứa xe thì lựa chọn giữ được xe, nếu cửa đổi là cửa chứa xe thì lựa chọn đổi được xe.

Mã trên cũng cho phép tình huống mở rộng với nhiều cửa bằng cách dùng tham số các cửa `doors`. Như kết quả mô phỏng cho thấy, trường hợp 3 cửa thì xác suất giữ được xe chỉ khoảng 1/3 và đổi được xe là 2/3 (1/3 và 2/3 là các giá trị chính xác khi tính toán bằng lý thuyết). Rõ ràng như “thực tế” cho thấy, đổi cửa sẽ có khả năng được xe cao hơn (gấp đôi) giữ cửa.

13.6 Vẽ ngẫu nhiên

Trong phần này, ta sẽ dùng ngẫu nhiên để vẽ nên các bức hình rất đẹp. Trước hết, ta học cách dùng Pillow, một **thư viện xử lý ảnh** (image processing library) miễn phí và phổ biến trên Python. Cũng như Matplotlib, Pillow là thư viện ngoài, cần cài đặt trước khi dùng. Trang chủ của Pillow được để ở địa chỉ <https://python-pillow.org/> và tài liệu tra cứu được để ở <https://pillow.readthedocs.io/en/stable/>.

Tương tự Matplotlib, ta có thể cài Pillow bằng lệnh sau trong Command Prompt:

```
py -m pip install -U Pillow
```

Sau khi cài đặt, giả sử có file ảnh `image.jpeg` đặt trong thư mục `D:\Python\lesson13` (bạn có thể dùng bất kỳ file ảnh nào hoặc dùng file ảnh `image.jpeg` tại <https://>

github.com/vqhBook/python/blob/master/lesson13/image.jpeg)⁸, ta có thể dùng thử Pillow như sau:

```

1 >>> import os; os.chdir(r"D:\Python\lesson13")
2 >>> from PIL import Image
3 >>> im = Image.open("image.jpeg")
4 >>> print(im.format, im.size, im.mode)
    JPEG (640, 640) RGB
5 >>> im = im.rotate(180)
6 >>> im.show()

```

Dòng 2 nạp module Image từ thư viện Pillow. Dòng 3 dùng hàm Image.open để mở file ảnh có tên được cho với kết quả trả về là đối tượng ảnh tương ứng. Lưu ý, nếu không mở được file (chẳng hạn tên file không đúng) thì lỗi thực thi sẽ được phát sinh. Dòng 4 dùng các trường format, size và mode của đối tượng ảnh để lấy các thông tin tương ứng là định dạng, kích thước (rộng, cao) và chế độ màu của ảnh. Dòng 5 dùng phương thức rotate để xoay ảnh và Dòng 6 hiển thị ảnh nhờ phương thức show.

Có một “thủ pháp” hội họa rất đơn giản nhưng cho hiệu ứng nghệ thuật rất mạnh là **vẽ chấm điểm** (dotwork, pointillism). Xuất phát từ nền trắng, ta cứ chấm từng điểm đen (hoặc màu) để tạo nên hình từ mật độ thưa dày của các điểm. Đối với họa sĩ, đây là một loại hình nghệ thuật “khổ sai” vì thường phải chấm rất nhiều điểm. Tuy nhiên, với máy, loại hình này lại rất phù hợp vì công việc rất đơn giản, thật sự đơn giản: lặp lại việc chấm các điểm. Dĩ nhiên câu hỏi tối quan trọng của thủ pháp này là: chấm chỗ nào? (và chấm bao nhiêu điểm?) Chương trình sau dùng ngẫu nhiên để hiện thực thủ pháp trên:⁹

https://github.com/vqhBook/python/blob/master/lesson13/random_dots.py

```

1 import random
2 from PIL import Image
3 import matplotlib.pyplot as plt
4 from matplotlib.animation import FuncAnimation
5
6 def update(_):
7     xy_choices = random.choices(xys, weights, k=K)
8     for xy in xy_choices:
9         out_im.putpixel(xy, (0, 0, 0))
10
11     ax.clear()
12     ax.imshow(out_im)
13
14 t = 4.0 # tham số điều khiển mật độ

```

⁸Bức ảnh “gây xao xuyến” này được “lượm” đâu đó trên mạng. Sẽ ghi nguồn khi truy được nguồn.

⁹Bạn có thể xem qua một tác phẩm kết quả ở https://github.com/vqhBook/python/blob/master/lesson13/image_dot.png.

```

15 K = 200 # số điểm trong một lần update
16
17 im = Image.open("image.jpeg").convert("L")
18 xys = [(x, y) for x in range(im.width) for y in
19         range(im.height)]
19 weights = [((255 - im.getpixel(xy))/255)**t for xy in xys]
20 out_im = Image.new("RGB", im.size, color=(255, 255, 255))
21
22 fig, ax = plt.subplots()
23 ani = FuncAnimation(fig, update, interval=1000//24)
24 plt.show()
25
26 out_im.save("image_dot.png")

```

Ta dùng sắc độ của điểm ảnh để quyết định xác suất điểm đó được chấm. Cụ thể, điểm ảnh càng đen thì khả năng được chấm càng lớn, tức mức xám càng thấp (càng gần 0 càng đen) thì khả năng được chọn sẽ cao lên. Tham số t quyết định mức độ “khuếch đại” xác suất này: t càng lớn thì “phân hóa” càng rõ (ảnh kết quả càng “nét”, các chấm điểm phân bố tập trung vào các vùng tối), t càng nhỏ thì càng “cào bằng” (ảnh kết quả càng “nhòe”, các chấm điểm phân bố càng ngẫu nhiên). Xem lại đồ thị các hàm số $y = x^t$ ở Phần 13.2 để hiểu rõ công thức “khuếch đại mũ” ở Dòng 19. Sau đây là vài giải thích thêm:

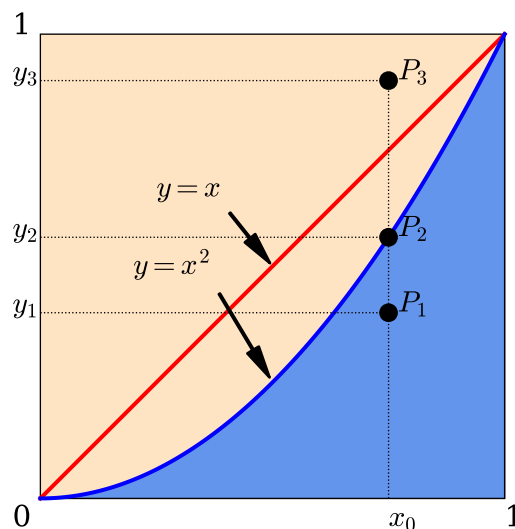
- Matplotlib hỗ trợ rất tốt Pillow. Cụ thể, phương thức `imshow` (Dòng 12) cho phép hiển thị ảnh trong đối tượng ảnh `Image` của Pillow.
- Phương thức `convert` với đối số "L" (Dòng 17) giúp chuyển ảnh về dạng **ảnh mức xám** (grayscale image). Khi đó, mỗi điểm ảnh mô tả một giá trị cường độ sáng là số nguyên trong phạm vi $[0, 255]$ với 0 là tối nhất (đen) và 255 là sáng nhất (trắng).
- Sau khi tạo danh sách tọa độ các điểm ảnh ở Dòng 18, Dòng 19 tính trọng số các điểm ảnh dựa vào độ tối của điểm ảnh. Các trọng số này được dùng trong hàm `random.choices` để chọn ngẫu nhiên các điểm ảnh ở Dòng 7 (trọng số càng lớn, khả năng được chọn càng cao). Các chấm đen được xuất ra ảnh kết quả tại các điểm ngẫu nhiên được chọn ở Dòng 8-9.
- Ảnh kết quả được tạo bằng hàm `Image.new` của Pillow ở Dòng 20. Mặc dù Pillow hỗ trợ rất nhiều chế độ màu nhưng ta nên dùng chế độ màu RGB khi dùng với Matplotlib (xem lại màu RGB ở Bài tập 5.3). Ảnh kết quả có kích thước đúng bằng ảnh gốc (`im`) và nền trắng (màu RGB là $(255, 255, 255)$).
- Khi chạy chương trình, các chấm đen sẽ được thêm từ từ vào ảnh kết xuất với mỗi lần cập nhật (gọi `update`) là K chấm. Nếu “nôn nóng” bạn có thể tăng giá trị cho K nhưng giá trị K nhỏ sẽ cho “độ phân giải thời gian” tốt hơn.
- Khi đóng cửa sổ hiển thị của Matplotlib, Dòng 26 lưu ảnh kết quả “hiện tại” vào file ảnh với định dạng PNG (Portable Network Graphics). Như vậy, nếu

để cửa sổ trình diễn quá lâu ta sẽ có ảnh bị “cháy” vì có quá nhiều chấm, ngược lại, nếu tắt cửa sổ hơi sớm thì ảnh sẽ bị “non” vì có ít chấm. Như vậy, nghệ thuật thực sự nằm ở việc điều chỉnh tham số t , K và canh thời gian dừng.¹⁰ Bạn hãy thử nghiệm các thông số này.

Có rất nhiều biến thể khác nhau của cách làm ngẫu nhiên trên, dẫn đến nhiều hiệu ứng nghệ thuật khác nhau. Ta sẽ thử nghiệm thêm trong phần Bài tập 13.10 và 13.11. Ý tưởng tương tự cũng có thể dùng cho nhiều lĩnh vực nghệ thuật khác như âm nhạc mà có lẽ ta sẽ gặp nhau trong một tài liệu khác hoặc bạn có thể tự tìm hiểu, thử nghiệm và sáng tạo.

13.7 Tính toán ngẫu nhiên

Ta đã thấy ngẫu nhiên xuất hiện trong nhiều lĩnh vực. Phần này trở lại với lĩnh vực cơ bản nhất của máy tính (và khoa học, kĩ thuật) là tính toán. Bài toán đặt ra là tính diện tích phần hình màu xanh trong hình dưới đây:¹¹



Phần hình ta cần tính diện tích được giới hạn bởi parabol $y = x^2$, trục hoành Ox và đường thẳng đứng cắt Ox tại 1. Hình trên cũng vẽ minh họa 3 điểm P_1, P_2, P_3 với cùng hoành độ x_0 . Điểm P_2 có tung độ $y_2 = x_0^2$ nên thuộc đường parabol, điểm P_1 nằm phía dưới parabol vì có tung độ $y_1 < x_0^2$ và điểm P_3 nằm phía trên parabol vì có tung độ $y_3 > x_0^2$. Như vậy, điểm có tọa độ (x, y) nằm trong phần hình màu xanh khi và chỉ khi $0 \leq y \leq x^2$.

Việc tính diện tích là rất đơn giản cho hình vuông bao ngoài (diện tích là 1) hay phần hình bên dưới đường thẳng $y = x$ (diện tích là $\frac{1}{2}$). Để tính diện tích hình

¹⁰Có lẽ tương tự như việc nướng cá!

¹¹Tôi dùng chính Matplotlib để vẽ hình này, sau đó lưu hình lại với định dạng PDF và chèn vào tài liệu này. Bạn nên xem mã nguồn của chương trình vẽ hình này tại https://github.com/vqhBook/python/blob/master/lesson13/parabol_draw.py và tra cứu thêm để thuần thục Matplotlib.

bên dưới parabol chính xác bằng Toán, ta cần dùng **lý thuyết tích phân** (integral calculus). Cụ thể, diện tích cần tính là:

$$S = \int_0^1 x^2 dx = \frac{x^3}{3} \Big|_0^1 = \frac{1}{3}$$

Mặc dù lời giải trên rất ngắn và chính xác nhưng Toán học đã phải mất thời gian rất lâu mới có được thành tựu này. Hơn nữa, để làm như vậy, bạn cần trình độ Toán 12 trở lên.¹² Bỏ qua mấy kí hiệu “ngồn ngộn” trên, ở đây, tôi chỉ cho bạn cách mà học sinh lớp 8 cũng có thể tính được đơn giản nhờ Python bằng mô phỏng ngẫu nhiên.

Xét thí nghiệm “tưởng tượng”: gieo một điểm ngẫu nhiên vào hình vuông đơn vị (hình vuông bao ngoài), tức là điểm có hoành độ và tung độ ngẫu nhiên trên đoạn $[0, 1]$. Ta quan tâm đến xác suất điểm rơi vào phần hình màu xanh (phần có diện tích S cần tính). Vì điểm gieo là ngẫu nhiên nên xác suất nó rơi vào các phần hình cùng diện tích là như nhau, do đó xác suất để điểm rơi vào phần hình màu xanh là:

$$P = \frac{\text{Diện tích phần màu xanh}}{\text{Diện tích hình vuông đơn vị}} = \frac{S}{1} = S$$

Như vậy, để tính S ta có thể tính P , mà xác suất P có thể được tính xấp xỉ bằng mô phỏng như ta đã làm trước giờ. Cụ thể, viết module cung cấp hàm mô phỏng như sau:

```

1 https://github.com/vqhBook/python/blob/master/lesson13/parabol\_area.py
2 from random import random
3
4 def parabol_area(N): # N là số điểm gieo mô phỏng
5     points = [(random(), random()) for _ in range(N)]
6     N_in = sum([y <= x**2 for x, y in points])
7     return N_in/N
8
9 if __name__ == "__main__":
10     N = int(input("Bạn muốn gieo bao nhiêu điểm: "))
11     print(f"Diện tích vùng dưới parabol là
12         {parabol_area(N):.4f}")

```

Hàm `parabol_area` chỉ đơn giản thực hiện việc gieo N điểm ngẫu nhiên trong hình vuông đơn vị (hàm `random.random` trả về một số thực ngẫu nhiên trong đoạn $[0, 1)$), đếm số điểm rơi vào phần hình bên dưới parabol và trả về tần suất (tỉ lệ điểm rơi vào). Khi N đủ lớn, tần suất này sẽ xấp xỉ xác suất như ta đã biết. Chạy module và tương tác như sau:

¹²Học sinh phổ thông được học tích phân ở Toán Giải tích Lớp 12. Ví dụ này cũng cho thấy, tích phân cùng với đạo hàm (Phần 9.3) là những khái niệm và công cụ rất quan trọng của Toán mà học sinh phổ thông nên nắm vững.

```

===== RESTART: D:\Python\lesson13\parabol_area.py =====
Bạn muốn gieo bao nhiêu điểm: 1_000_000
Diện tích vùng dưới parabol là 0.3336

1 >>> parabol_area(1000)
0.334

2 >>> parabol_area(1000)
0.328

```

Phương pháp tính toán dùng mô phỏng ngẫu nhiên như trên thường được gọi là phương pháp **Monte Carlo** (Monte Carlo method). Nó là phương pháp tuyệt vời vì rất tổng quát và rất dễ thực hiện (nhờ các công cụ mô phỏng như Python).¹³ Nó cũng rất thú vị vì dùng ngẫu nhiên để giải các bài toán **tất định** (deterministic).

Tóm tắt

- Ngẫu nhiên là bản chất, là hương vị của cuộc sống.
- Matplotlib là thư viện trực quan hóa miễn phí, mạnh mẽ và phổ biến trên Python. Matplotlib cho phép tạo các đồ họa tĩnh, tương tác và hoạt họa.
- Mô phỏng ngẫu nhiên là kĩ thuật tái hiện các quá trình, thí nghiệm, hoạt động, ... ngẫu nhiên trên máy.
- Tung đồng xu là thí nghiệm ngẫu nhiên hay được dùng để phân giải các tranh chấp “2 bên” vì có 2 kết quả sắp/ngửa với khả năng như nhau.
- Trò chơi may rủi là các trò chơi có tính ngẫu nhiên. Mô phỏng giúp ta trả lời nhiều câu hỏi trên các trò chơi này.
- Trực giác của ta thường bị sai khi đối mặt với các vấn đề liên quan đến ngẫu nhiên. Mô phỏng là cách đơn giản nhưng thuyết phục để phân tích các tình huống như vậy.
- Pillow là thư viện xử lý ảnh miễn phí, mạnh mẽ và phổ biến trên Python. Matplotlib hỗ trợ Pillow khá tốt.
- Các thủ pháp dùng ngẫu nhiên (chẳng hạn, vẽ chấm điểm dựa trên mức xám) mang lại hiệu ứng nghệ thuật rất mạnh.
- Monte Carlo là phương pháp tính toán dùng ngẫu nhiên. Phương pháp này giúp tính toán xấp xỉ nhiều đại lượng mà Toán học không tính chính xác nổi.

Bài tập

13.1 Dùng Matplotlib giải trực quan các yêu cầu sau đây:¹⁴

¹³Bạn cũng nên biết việc tính tích phân không hề đơn giản. Thậm chí, có nhiều trường hợp không tính được bằng lý thuyết. Tuy nhiên, ta luôn có thể dùng phương pháp Monte Carlo để tính xấp xỉ.

¹⁴SGK Toán 9 Tập 1.

- (a) Vẽ đồ thị của các hàm số $y = 2x$; $y = 2x + 5$; $y = -\frac{2}{3}x$; $y = -\frac{2}{3}x + 5$ trên cùng một mặt phẳng tọa độ.
- (b) Bốn đường thẳng trên cắt nhau tạo thành tứ giác $OABC$ (O là gốc tọa độ). Tứ giác $OABC$ có phải là hình bình hành không? Vì sao?

13.2 Dùng Matplotlib giải trực quan các yêu cầu sau đây:¹⁵

- (a) Vẽ đồ thị của các hàm số $y = \frac{1}{2}x^2$; $y = x^2$; $y = 2x^2$ trên cùng một mặt phẳng tọa độ.
- (b) Tìm ba điểm A, B, C có cùng hoành độ $x = -1.5$ theo thứ tự nằm trên ba đồ thị. Xác định tung độ tương ứng của chúng.
- (c) Tìm ba điểm A', B', C' có cùng hoành độ $x = 1.5$ theo thứ tự nằm trên ba đồ thị. Kiểm tra tính đối xứng của A và A' , B và B' , C và C' .
- (d) Với mỗi hàm số trên, hãy tìm giá trị của x để hàm số đó có giá trị nhỏ nhất.

13.3 Dùng kĩ thuật đồ họa tương tác của Matplotlib:

- (a) Khảo sát đường thẳng $y = ax + 1$ với a là giá trị thực trong khoảng $[-5, 5]$.
- (b) Khảo sát parabol $y = ax^2$ với a là giá trị thực trong khoảng $[-5, 5]$.

13.4 Dùng Matplotlib vẽ biểu đồ thanh và biểu đồ quạt từ bảng tần số của Bài tập 12.10d.

13.5 Gieo xúc xắc. Bên cạnh tung đồng xu, một mô hình ngẫu nhiên khác cũng hay được dùng là **gieo xúc xắc** (dice rolling). Xúc xắc (đồng chất) có thể được xem là mở rộng của đồng xu với 6 mặt có khả năng ra như nhau (như vậy, xác suất ra 1 mặt cụ thể trong 6 mặt $\{1, 2, \dots, 6\}$ là $\frac{1}{6}$). Thực hiện mô phỏng trả lời các câu hỏi sau:

- (a) Khi gieo 2 con xúc xắc đồng chất thì khả năng được tổng 2 mặt là 11 hay tổng 2 mặt là 12 lớn hơn?¹⁶
- (b) Nếu gieo 1 xúc xắc thì khả năng được mặt 6 là thấp, chỉ $\frac{1}{6}$. Nếu gieo 2 xúc xắc thì khả năng có mặt 6 (trong 2 con) thì cao hơn. Gieo ít nhất bao nhiêu con xúc xắc để khả năng có mặt 6 là lớn hơn 50%.¹⁷

13.6 Birthday Problem. Tương tự “Monty Hall Problem”, “**bài toán sinh nhật**” (birthday problem) là một bài toán xác suất dễ gây sai lầm trực giác. Bài toán này như sau: có một nhóm gồm n người, ta quan tâm đến xác suất có ít nhất 2 người cùng sinh nhật. Giả sử một năm có 365 ngày và mỗi người được sinh vào 1 ngày ngẫu nhiên. Hỏi n nhỏ nhất là bao nhiêu để xác suất này lớn hơn 50%? Thử nghiệm bằng cách mô phỏng. Bạn sẽ ngạc nhiên về kết quả.

¹⁵SGK Toán 9 Tập 2.

¹⁶Tương truyền, nhà bác học người Đức Leibniz đã sai lầm khi cho rằng 2 trường hợp có khả năng như nhau. Đây cũng là ví dụ cho thấy trực giác rất dễ mắc sai lầm.

¹⁷Tương truyền, các nhà Toán học Pháp Pascal và Fermat đã thảo luận nhiều về câu hỏi này.

13.7 Dùng phương pháp Monte Carlo tính xấp xỉ diện tích hình bên dưới các đường trong hình vuông đơn vị:

- (a) $y = x^3$.
- (b) $y = \sqrt{x}$.
- (c) $y = \sin x$.
- (d) Phần tư đường tròn có tâm là gốc tọa độ O và bán kính là 1. (Đối chiếu lại với công thức tính diện tích hình tròn.)

13.8 “Monty Hall Problem” có nhiều biến thể khác nhau. Xem các biến thể này tại https://en.wikipedia.org/wiki/Monty_Hall_problem#Variants và viết chương trình mô phỏng cho mỗi biến thể. Đối chiếu kết quả mô phỏng với phân tích lý thuyết.

13.9 Viết chương trình cho người dùng chơi trò:

- (a) Tôm Cua.
- (b) Let’s Make a Deal.

Thử dùng con rùa (Turtle) để tạo giao diện đồ họa.

13.10 Thử nghiệm các biến thể sau cho mã `random_dots.py` ở Phần 13.6:

- Tăng kích thước chấm điểm (thử kích thước cố định và ngẫu nhiên).
- Chọn màu cho chấm điểm (thử một màu và nhiều màu “đậm nhạt”).
- Vẽ chấm điểm với cường độ và màu riêng theo 3 kênh kênh màu red, green, blue (thử cùng một lượt và tuần tự 3 lượt).
- ... (Bạn tự sáng tạo và tự làm luôn đi!)

13.11 Đi ngẫu nhiên. Bạn đã thấy con rùa vẽ các đường thú vị ở Phần 6.5 khi nó bò ngẫu nhiên hay đúng thuật ngữ là **đi ngẫu nhiên** (random walk): tại mỗi bước di chuyển (bò/đi) con rùa chọn đại một trong các hướng có thể. Ở bài tập này, ta cũng dùng random walk để vẽ các đường ngẫu nhiên nhưng có định hướng hơn. Cụ thể, tại mỗi bước, ta ưu tiên chọn hướng đến vùng tối hơn.

Bạn nghiên cứu mã chương trình tại https://github.com/vqhBook/python/blob/master/lesson13/random_walks.py với tác phẩm kết quả https://github.com/vqhBook/python/blob/master/lesson13/image_walk.png. Bạn thử nghiệm điều chỉnh các tham số và nếu may mắn, bạn sẽ có một tác phẩm tuyệt vời hoặc ít nhất là độc nhất vô nhị vì không ai có thể vẽ lại được như vậy (trừ khi dùng kỹ thuật ở Bài tập 13.18).

Tương tự Bài tập 13.10, bạn thử nghiệm và sáng tạo các biến thể cho thủ pháp “đường đi ngẫu nhiên”.

13.12 Bạn đã nghe đến thủ pháp “vẩy mực” vẽ tranh chưa? Thiết kế “thuật toán ngẫu nhiên”, cài đặt chương trình Python và thử nghiệm tương tự thủ pháp “chấm điểm” và “đường đi” ngẫu nhiên.

13.13 Xử lý ảnh với Pillow. Thư viện Pillow cung cấp các module với nhiều hàm, lớp, phương thức giúp thao tác **xử lý ảnh** (image processing) đơn giản nhưng

mạnh mẽ. Thử chương trình minh họa tại https://github.com/vqhBook/python/blob/master/lesson13/violet_img.py.

Chương trình “tô màu” tím thương nhớ cho ảnh `image.jpeg`)¹⁸ Dòng 3 mở ảnh nguồn. Dòng 4 tách các kênh màu. Dòng 5 tạo ảnh cho kênh màu xanh lá (green) với giá trị 0 (không có cường độ). Dòng 6 trộn các kênh màu với kênh màu xanh lá đã “tắt” để được ảnh mới. Dòng 7 lưu và Dòng 8 hiển thị ảnh mới. Tóm lại, ta giữ 2 kênh màu đỏ (red) và xanh dương (blue) để tạo màu tím (violet).

Tra cứu các hàm của module `Image`, `ImageEnhance`, `ImageFilter`, `ImageOps` và các phương thức, trường của lớp `Image` (<https://pillow.readthedocs.io/en/stable/reference/index.html>) để viết chương trình dùng Pillow thực hiện các thao tác xử lý ảnh sau:

- (a) Đổi định dạng ảnh (convert).
- (b) Chuyển ảnh mức xám (grayscale).
- (c) Tạo ảnh âm bản (negative).
- (d) Xén ảnh (crop).
- (e) Các thao tác biến đổi hình học (geometrical transform): lật ảnh (flip), xoay ảnh (rotate), thay đổi kích thước (resize), thu phóng ảnh (scale).
- (f) Các thao tác biến đổi màu (color transform): chỉnh sáng tối (brightness), tương phản (contrast), độ nét (sharpness), lọc màu (filter), tách (split) – biến đổi – trộn (merge) các kênh màu.
- (g) Các thao tác trên nhiều ảnh: trộn ảnh (blending), dán ảnh (pasting).

13.14 Vẽ với Pillow. Thư viện Pillow cung cấp module `ImageDraw` hỗ trợ các thao tác vẽ hình đơn giản. Thử chương trình minh họa tại https://github.com/vqhBook/python/blob/master/lesson13/pillow_draw.py.

Chương trình vẽ hình vuông màu đỏ với hình tròn màu xanh dương bên trong (Bài tập 5.3(a)).¹⁹ Dòng 3 đặt kích thước ảnh. Dòng 4 tạo ảnh RGB mới với kích thước tương ứng và màu nền đỏ. Dòng 5 tạo đối tượng vẽ (`Draw`) cho ảnh tương ứng và Dòng 6 dùng đối tượng này để vẽ (và tô) hình tròn màu xanh. Dĩ nhiên, ta có thể dùng các chức năng vẽ kết hợp với các chức năng xử lý ảnh của Pillow như vẽ lên ảnh đã có, dán ảnh thêm vào, ...

Tra cứu module `ImageDraw` (<https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html>) để viết chương trình dùng Pillow vẽ các đường/hình sau:

- (a) Biểu tượng âm dương (mã `yin_yang.py` ở Phần 5.4).
- (b) Các đường ở Bài tập 5.1, bạn cũng tô màu luôn cho đẹp.

¹⁸Ảnh kết quả được để ở https://github.com/vqhBook/python/blob/master/lesson13/image_violet.jpeg.

¹⁹Ảnh kết quả được để ở https://github.com/vqhBook/python/blob/master/lesson13/pillow_draw.png.

- (c) Các hình ở Bài tập 5.3.
- (d) Các hình ở Bài tập 5.4.
- (e) Logo Python (Bài tập 5.10).
- (f) Các hình ở Bài tập 7.7.

13.15 Ảnh động GIF. GIF (Graphics Interchange Format) là định dạng ảnh được dùng phổ biến trên nhiều nền tảng và công nghệ máy tính nhờ kích thước lưu trữ ảnh nhỏ.²⁰ Hơn nữa, GIF cũng hỗ trợ **ảnh hoạt họa** (animated GIF) hay nói bình dân là ảnh động, tức là ảnh gồm nhiều khung hình (xem lại Phần 9.6).

Matplotlib và Pillow hỗ trợ việc tạo các ảnh động GIF rất đơn giản như minh họa trong chương trình tại https://github.com/vqhBook/python/blob/master/lesson13/con_lac_don.py. Chương trình tạo file ảnh động GIF trình diễn cảnh một con lắc đơn dao động.²¹

Con lắc đơn và qui luật dao động của nó được học trong chương trình Vật Lý Lớp 12. Ở đây, quan trọng là cách ta tạo file ảnh động: dùng phương thức save của đối tượng hoạt họa (FuncAnimation) như Dòng 25. Tham số đầu tiên là tên file ảnh sẽ tạo, tham số writer xác định “bộ tạo ảnh” mà ở đây ta dùng Pillow (dĩ nhiên, cả Matplotlib lẫn Pillow phải được cài đặt rồi). Ta cũng cần xác định số lượng khung trình diễn với tham số frames khi gọi FuncAnimation.

- (a) Tạo các file ảnh động GIF tương tự các file sau:
 - <https://upload.wikimedia.org/wikipedia/commons/a/a4/Animexample.gif>.
 - <https://upload.wikimedia.org/wikipedia/commons/e/e3/Animhorse.gif>.
 - https://upload.wikimedia.org/wikipedia/commons/0/03/Catenary_animation.gif.
- (b) Tìm hiểu cách dùng phương pháp Monte Carlo để tính số π ở https://en.wikipedia.org/wiki/Monte_Carlo_method#Overview, viết chương trình mô phỏng và tạo ảnh động GIF tương tự như ảnh https://en.wikipedia.org/wiki/Monte_Carlo_method#/media/File:Pi_30K.gif.
- (c) Lưu trữ các trình diễn trong bài này vào file ảnh động GIF.
- (d) Thực hiện các hình vẽ, trình diễn trong Bài 9 bằng Matplotlib (và/hoặc Pillow) và lưu lại trong file ảnh động GIF.

13.16 Video. Phương tiện tốt nhất để lưu trữ và trình diễn các hoạt họa, dĩ nhiên, là **video**. Có một bộ công cụ xử lý video (và audio, ảnh, ...) miễn phí nhưng

²⁰Bù lại, GIF chỉ nên dùng cho các ảnh có số lượng màu ít, không phù hợp với các ảnh chụp nhiều màu.

²¹Bạn có thể mở xem file GIF với bất kì ứng dụng hỗ trợ nào trên máy. File ảnh động trên cũng được để ở https://github.com/vqhBook/python/blob/master/lesson13/con_lac_don.gif.

rất mạnh và tiện lợi là FFmpeg (<https://ffmpeg.org/>). Matplotlib cũng hỗ trợ việc tạo các file video trình diễn với FFmpeg.

Trước hết, ta cần cài đặt FFmpeg. Vào trang download, <https://ffmpeg.org/download.html>, tải file nén chứa các chương trình FFmpeg.²² Sau đó giải nén vào một thư mục và ghi nhận đường dẫn đến tập tin chương trình `ffmpeg.exe` (chẳng hạn, trên máy tôi là `D:\ffmpeg\bin\ffmpeg.exe`).

Sau khi có chương trình FFmpeg (file `ffmpeg.exe`) ta dùng nó để tạo file video với Matplotlib như chương trình minh họa tại https://github.com/vqhBook/python/blob/master/lesson13/randot_video.py. Chương trình tạo file video trình diễn hoạt họa vẽ điểm ở Phần 13.6.²³

Mã chương trình là mã `random_dots.py` ở Phần 13.6 bổ sung thêm các hỗ trợ lưu file video. Cụ thể, ta dùng `FFMpegWriter` làm “bộ tạo video” mà đó chính là `ffmpeg.exe` nên trước đó, ở Dòng 28, ta cần đặt đường dẫn đến file chương trình này để Matplotlib biết chỗ chứa nó. Ta cũng có thể lưu với các định dạng video khác như AVI, MOV, ...

- (a) Lưu trữ các trình diễn trong bài này vào file video.
- (b) Thực hiện các hình vẽ, trình diễn trong Bài 9 bằng Matplotlib (và/hoặc Pillow) và lưu lại trong file video.

13.17 Histogram và Boxplot. Bên cạnh biểu đồ thanh và quạt, có 2 loại biểu đồ khác cũng hay được dùng để mô tả số liệu là **biểu đồ tần suất** (histogram) và **biểu đồ hộp** (boxplot). Hai biểu đồ này mô tả “phân bố” các số liệu của một biến định lượng. Thử chương trình minh họa tại https://github.com/vqhBook/python/blob/master/lesson13/histogram_boxplot.py.

Chương trình phát sinh N số thực ngẫu nhiên trong khoảng $[0, 1]$ (bằng hàm `random.random`) và khảo sát phân bố của chúng bằng histogram và boxplot. Như kết quả tại https://github.com/vqhBook/python/blob/master/lesson13/histogram_boxplot.png cho thấy, các số này phân bố khá “đều” trong khoảng $[0, 1]$. Điều đó có nghĩa là chúng được tạo “ngẫu nhiên” trong khoảng $[0, 1]$ như hứa hẹn của hàm `random.random`.

- (a) Tìm hiểu thêm về histogram và boxplot.
- (b) Tra cứu và xem các ví dụ minh họa vẽ histogram, boxplot của Matplotlib.
- (c) Làm lại Bài tập 11.10b với việc vẽ histogram và boxplot cho mẫu số liệu trước khi tính các thống kê trên chúng.

13.18 Giả ngẫu nhiên. Việc tạo các số ngẫu nhiên (hay tổng quát, các đối tượng ngẫu nhiên) là nền tảng của các phương pháp mô phỏng như Monte Carlo. Với Python, ta đã dùng “bộ tạo số ngẫu nhiên” cung cấp bởi module `random`. Vậy

²²Với máy Windows, dùng link <https://ffmpeg.zeranoe.com/builds/win64/static/ffmpeg-20200515-b18fd2b-win64-static.zip>.

²³Bạn có thể mở xem file MP4 với bất kì ứng dụng hỗ trợ nào trên máy. File video trên cũng được để ở https://github.com/vqhBook/python/blob/master/lesson13/random_dots.mp4.

random có tạo được các số “ngẫu nhiên thực sự” hay không? Câu trả lời là không như minh họa sau cho thấy:

```

1 >>> import random
2 >>> [round(random.random(), 4) for _ in range(5)]
  [0.1849, 0.9827, 0.6923, 0.0604, 0.7122]
3 >>> random.seed(1)
4 >>> [round(random.random(), 4) for _ in range(5)]
  [0.1344, 0.8474, 0.7638, 0.2551, 0.4954]
5 >>> random.seed(1)
6 >>> [round(random.random(), 4) for _ in range(5)]
  [0.1344, 0.8474, 0.7638, 0.2551, 0.4954]
7 >>> random.seed(101)
8 >>> print(random.randrange(10), random.choice("abcd"))
  9 b
9 >>> random.seed(101)
10 >>> print(random.randrange(10), random.choice("abcd"))
  9 b

```

Hàm `random.random` tạo số thực ngẫu nhiên trong khoảng $[0, 1]$ như ta đã biết. Điều đó cũng có nghĩa, kết quả trả về của `random()` là không đoán được, nó có thể là “bất kì” số nào (trong khoảng $[0, 1]$). Như vậy, về lý thuyết, kết quả của Dòng 2 là không đoán được và không lặp lại được.

Tuy nhiên, cách module `random` tạo số ngẫu nhiên là hoàn toàn tất định và đoán được, hoặc ít nhất là lặp lại được như Dòng 4, 6 cho thấy. Nói nôm na, từ một “mầm” (seed) cho trước, `random` tạo số “trông có vẻ ngẫu nhiên” (nhưng hoàn toàn xác định từ mầm), rồi tạo số tiếp theo từ số trước đó thành dãy hoàn toàn xác định từ mầm ban đầu. Do đó, các số này được gọi là **giả ngẫu nhiên** (pseudorandom).

Dù không thật sự ngẫu nhiên (ta không nên dùng module `random` tạo số ngẫu nhiên cho chương trình xổ số hay các trò may rủi ăn tiền) nhưng ta có thể dùng chúng cho việc mô phỏng vì chúng có các “tính chất” của các số ngẫu nhiên thật. Hơn nữa, trong trường hợp này, số giả ngẫu nhiên lại có một lợi thế là có thể **tái tạo** (reproducing) lại được cùng một dãy số ngẫu nhiên, do đó tái tạo lại được kết quả mô phỏng. Chẳng hạn, ta có thể “vẽ lại” được các bức ảnh ngẫu nhiên bằng cách đặt cùng mầm trước khi vẽ.

- (a) Tra cứu module `random` (<https://docs.python.org/3/library/random.html>).
- (b) Thử nghiệm việc tái tạo kết quả mô phỏng, ngẫu nhiên cho các chương trình trong Bài này (chẳng hạn, các chương trình vẽ ngẫu nhiên ở Bài tập **13.10**, **13.11**).

Bài 14

Tự viết lấy lớp

Lớp là khuôn mẫu chung của đối tượng còn đối tượng là thể hiện cụ thể của lớp. Ta đã học về đối tượng, về việc dùng lớp có sẵn để tạo đối tượng trong các bài trước, nay ta học cách tự mình định nghĩa lấy lớp. Điều này mở ra một khung trời nữa trong Python (tương tự việc tự định nghĩa lấy hàm ở Bài 8). Lớp cũng là nhân tố cơ bản của kĩ thuật lập trình hướng đối tượng, một kĩ thuật lập trình mạnh mẽ và quan trọng nhất hiện nay. Trước khi học bài này, bạn nên xem lại giới thiệu về đối tượng và lớp ở Bài 10.

14.1 Lớp là khuôn mẫu của các đối tượng thể hiện

Tới bài này ta đã gặp rất nhiều lớp khác nhau. Đơn giản nhất là lớp của các dữ liệu cơ bản mà ta hay gọi là kiểu như: `str` lớp của các chuỗi, `int` lớp của các số nguyên, `float` lớp của các số thực chấm động, ... Phức tạp hơn là lớp của các dữ liệu ghép mà ta hay gọi là cấu trúc dữ liệu như: `list` lớp của các danh sách, `dict` lớp của các từ điển, ... Ta cũng gặp lớp của các đối tượng khác như `decimal.Decimal`, `fractions.Fraction`, `turtle.Turtle`, ...

Ta dĩ nhiên đã dùng rất nhiều các đối tượng cụ thể (mà nhiều trường hợp ta gọi là dữ liệu) của các lớp này. Ta cũng tương tác trên các đối tượng này qua các thuộc tính (phương thức, trường, toán tử mà cũng là phương thức như ta đã biết). Trong Phần 10.8 ta cũng đã tự hỏi các phương thức này từ đâu mà có. Minh họa sau “tự” tạo một lớp mới và dùng nó để thấy rõ “lớp là khuôn mẫu của các đối tượng”:

```
1 >>> class C:
2     a = 0
3     def f1(obj=None): return C.a
4     def f2(obj): return obj.b
5
6 >>> [att for att in dir(C) if not att.startswith("__")]
['a', 'f1', 'f2']
7 >>> print(C, type(C), callable(C))
```

```

<class '__main__.C'> <class 'type'> True
8 >>> c1 = C(); c2 = C(); c1.b = 1; c2.b = 2
9 >>> print(c1, type(c1), c1.__class__ is c2.__class__ is C)
<__main__.C object at 0x03C2A3D0> <class '__main__.C'> True
10 >>> [att for att in dir(c1) if not att.startswith("__")]
['a', 'b', 'f1', 'f2']
11 >>> C.a = 5; c1.a = 10
12 >>> print(C.a, c1.a, c2.a, C.f1(), c1.f1(), c2.a is C.a)
5 10 5 5 5 True
13 >>> print(C.f2(c1), C.f2(c2), c1.f2(), c2.f2())
1 2 1 2
14 >>> print(type(C.f2), type(c1.f2), c1.f2 is C.f2)
<class 'function'> <class 'method'> False
15 >>> c1.f2 = lambda obj: 2*obj.b
16 >>> print(type(c1.f2), c1.f2(c1))
<class 'function'> 2
17 >>> c1.f2()
...
TypeError: <lambda>() missing 1 required positional argument: 'obj'

```

Lệnh **class** (class statement) với cú pháp:

```

class <Name>:
    <Suite>

```

giúp ta định nghĩa một lớp mới có tên là <Name>. **class** là từ khóa còn <Suite> là một khối lệnh, được gọi là thân lớp. Khi gặp lệnh **class**, Python tạo không gian tên cục bộ rồi thực thi thân lớp trong không gian tên này (tương tự việc thực thi hàm, xem lại Phần 10.7) và sau khi thực thi xong, nó được chuyển thành không gian tên của **đối tượng lớp** (class object) do tên lớp tham chiếu đến (tương tự việc nạp module, xem lại Phần 10.8). Ở đây, các Dòng 1-4 định nghĩa lớp tên C mà sau khi thân của nó (Dòng 2-4) được thực thi xong thì đối tượng lớp C có các thuộc tính là biến a và 2 hàm f1, f2.¹

Đặc biệt, đối tượng lớp C là gọi được và ta đã “gọi” nó như hàm ở Dòng 8. Lỗi gọi đối tượng lớp sẽ **tạo** (instantiation) một **thể hiện** (instance) của lớp. Một lớp có thể có nhiều thể hiện cụ thể khác nhau. Ở đây, ta tạo 2 thể hiện c1, c2 của lớp C. Các thể hiện cũng là các đối tượng và ta có thể gán biến (tức là trường) cho nó như đã làm với con rùa ở Phần 10.8. Cụ thể, ta đã gán biến b giá trị 1 cho c1 và biến b giá trị 2 cho c2.

Python gán thuộc tính đặc biệt `__class__` cho các đối tượng thể hiện tham chiếu đến đối tượng lớp mà đã được dùng để tạo thể hiện đó. Ta đã dùng từ sớm hàm `type` để lấy về kiểu của đối tượng, nay ta biết rằng nó trả về đối tượng lớp

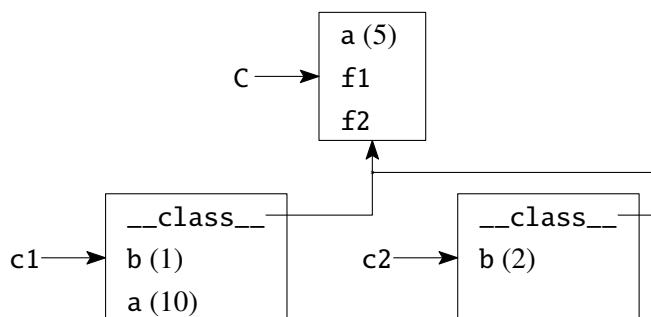
¹ Dĩ nhiên C còn có các thuộc tính đặc biệt (`__xyz__`) khác.

tham chiếu bởi thuộc tính `__class__` như kết quả của Dòng 9 cho thấy. Ta có thể hình dung cài đặt của hàm `type` là:

```
def type(obj): return obj.__class__
```

Kết quả của Dòng 10 hé lộ phần nào cơ chế tra thuộc tính của Python: Python tìm trong không gian tên của đối tượng trước mà nếu không có sẽ tìm trong không gian tên của đối tượng lớp (do thuộc tính `__class__` tham chiếu). Như vậy, mặc dù bản thân `c1` chỉ có thuộc tính `b` nhưng nó “thừa kế” các thuộc tính của `C`. Đây (cùng với khả năng tạo thể hiện từ lớp) chính là cách mà lớp cung cấp khuôn mẫu chung cho các thể hiện của nó: *các thể hiện “thừa kế” mọi thuộc tính của lớp*. Cơ chế tìm kiếm này cũng cho phép các thể hiện là các trường hợp đặc biệt, riêng lẻ của lớp vì chúng có thể có các thuộc tính riêng, “đề” các thuộc tính của lớp. Nó cũng cho phép sự khác biệt giữa các thể hiện vì các thuộc tính riêng của đối tượng nằm trong không gian tên của đối tượng. Ta sẽ thấy bức tranh đầy đủ của cơ chế này ở Phần 14.4.

Lệnh gán đầu tiên ở Dòng 11 rất bình thường, gán lại thuộc tính `a` của đối tượng lớp `C` giá trị mới 5 thay cho giá trị cũ 0. Lệnh gán thứ 2 cần lưu ý, thuộc tính mới `a` của thể hiện `c1` được tạo với giá trị 10. Khác với việc tra (đọc), việc gán (định nghĩa) sẽ đặt tên ngay tại không gian tên của đối tượng (tương tự như khác biệt giữa việc dùng và định nghĩa biến trong không gian tên cục bộ ở Phần 10.7). Như vậy, sau dòng này ta có “hiện trạng” các đối tượng `c1`, `c2`, `C` như hình sau và do đó có kết quả của Dòng 12.



Dòng 13 cho thấy cái mà ta vẫn hay gọi là “phương thức”. Lời gọi `C.f2(c1)` và `C.f2(c2)` là bình thường nếu nhớ rằng, mặc dù trùng tên nhưng `b` của `c1` khác với `b` của `c2`. Lời gọi `c1.f2()` (và `c2.f2()`) là đặc biệt vì có sự can thiệp của Python để hình thành “cơ chế phương thức”. Cụ thể, vì `c1` không có thuộc tính `f2` nên Python tra trong `C` và vì `f2` của `C` là đối tượng hàm nên Python tạo **đối tượng phương thức** (method object) “bọc” `C.f2` vào `c1` để khi đối tượng phương thức này được gọi thì hàm `C.f2` được gọi với đối số đầu tiên là `c1`. Nghĩa là, lời gọi `c1.f2()` sẽ được chuyển thành lời gọi `C.f2(c1)`. Thật ra, ta cũng đã gặp hiện tượng này ở Dòng 12, cụ thể, `C.f1()` gọi `C.f1` với đối số mặc định `None` nhưng `c1.f1()` gọi `C.f1` với đối số là `c1` dù kết quả vẫn như nhau.

Như vậy, về mặt thuật ngữ, `C.f2` là hàm (tham chiếu đến đối tượng hàm) còn

c1.f2 là phương thức (tham chiếu đến đối tượng phương thức). Chúng khác nhau dù rằng phương thức chỉ đơn giản là bọc đối tượng với hàm. Khác biệt “nhỏ nhỏ” này lại có ý nghĩa cực kì lớn và là bí mật đằng sau của **phương thức** (method), cái mà ta đã nói là “hàm gắn với đối tượng” ở Phần 10.5.

Đặc biệt lưu ý Dòng 15, ta đã gán thuộc tính f2 của c1 thành hàm mới. Hàm này (kết quả của biểu thức lambda) có tham số và thân hàm “tương tự” hàm C.f2. Tuy nhiên, vì c1 đã có thuộc tính f2 nên “cơ chế phương thức” trên không xảy ra. Cụ thể, c1.f2 là hàm mới (và không liên quan gì đến C.f2). Dĩ nhiên, vì c2 không có thuộc tính “riêng” f2 nên c2.f2 vẫn là phương thức. Vì c1.f2 chỉ là hàm nên nó được gọi như hàm bình thường, do đó lỗi thực thi ở lời gọi c1.f2() là dễ hiểu vì thiếu đối số. Như vậy, về mặt kĩ thuật thì phương thức là “hàm gắn với lớp của đối tượng” và lớp là không gian tên mà các đối tượng thể hiện có thể “thừa kế”.

Với các kiến thức này, ta làm lại Bài tập 10.2 một cách tường minh bằng lớp thay vì ngầm định bằng bao đóng hàm như sau:

https://github.com/vqhBook/python/blob/master/lesson14/write_char.py

```

1 import turtle as t
2 import string
3
4 class WriteChar:
5     def __init__(self, char):
6         self.char = char
7
8     def write(self):
9         t.clear()
10        t.write(self.char, align="center", font=f"Arial {s}")
11        t.update()
12
13
14 s = 300
15 t.hideturtle(); t.tracer(False); t.color("red")
16 t.up(); t.right(90); t.forward(2*s//3); t.down()
17
18 for char in string.ascii_letters + string.digits:
19     t.onkey(WriteChar(char).write, char)
20
21 t.listen()
22 t.exitonclick()
```

Mỗi phím kí tự là cụ thể khác nhau nhưng chúng có chung cách xuất ra màn hình con rùa. Ta giải quyết vấn đề “riêng-chung” này bằng cách dùng lớp cung cấp phương thức chung xuất kí tự và các thể hiện có thông tin trạng thái riêng là các kí tự khác nhau. Dòng 18-19 tạo các đối tượng thể hiện khác nhau của lớp WriteChar, mỗi thể hiện ứng với một phím kí tự (char). Các thể hiện này dùng chung hàm write của lớp WriteChar làm trình xử lý sự kiện; đúng hơn, mỗi thể hiện có một

phương thức `write` riêng (bọc hàm `WriteChar.write` với thể hiện) và phương thức này “biết” thể hiện tương ứng nên truy cập được kí tự riêng (`self.char` ở Dòng 10) của thể hiện đó. Ta sẽ rõ hơn trong các phần sau.

14.2 Đóng gói dữ liệu với thao tác

Ta đã thấy bản chất của lớp về mặt kĩ thuật ở phần trên. Quan trọng hơn là cách lớp được dùng. Về cơ bản, lớp giúp ta **đóng gói** (encapsulation) dữ liệu với thao tác, tức là trường với phương thức. Trong mã `sqrt_cal2.py` ở Phần 8.5 và `sqrt_cal3.py` ở Phần 11.1, ta đã giải quyết Bài tập 3.4 để khai phương các số bằng các phương pháp khác nhau. Ta nhận thấy rằng một phương pháp khai phương không chỉ có thao tác (hàm khai phương) mà còn có tên (chuỗi). Hơn nữa, chúng có chung một thao tác là khai phương một dãy số. Lớp cho phép ta gắn kết tất cả các thứ này lại một cách chặt chẽ và thống nhất. Dùng lớp, các mã trên có thể được viết lại đẹp hơn như sau:

https://github.com/vqhBook/python/blob/master/lesson14/sqrt_cal.py

```

1  import math
2
3  class SqrtMethod:
4      """Class for square root methods"""
5
6      def __init__(self, func, name):
7          self.func = func
8          self.name = name
9
10     def cal_sqrt(self, nums):
11         print(f"Tính căn bằng phương pháp {self.name}:")
12         for num in nums:
13             print(f"Căn của {num} là {self.func(num):.9f}")
14
15     if __name__ == "__main__":
16         sqrt_methods = [
17             SqrtMethod(math.sqrt, "math's sqrt"),
18             SqrtMethod(lambda x: math.pow(x, 0.5), "math's pow"),
19             SqrtMethod(lambda x: pow(x, 0.5), "built-in pow"),
20             SqrtMethod(lambda x: x ** 0.5, "exponentiation
21                 ↪ operator"),
22             ]
23         nums = [0.0196, 1.21, 2, 3, 4, 225/256]
24         for sqrt_method in sqrt_methods:
25             sqrt_method.cal_sqrt(nums)

```

Dòng 3-13 là định nghĩa của lớp `SqrtMethod` mô tả các phương pháp tính căn. Mỗi đối tượng (thể hiện) của lớp này mô tả một phương pháp cụ thể và đều có 2

thuộc tính là tên phương pháp (`name`, chuỗi) và hàm thực hiện việc tính căn (`func`, hàm nhận một số và trả về căn của số đó). Ngoài ra, chúng có chung một phương thức là tính căn cho các số của một danh sách số (phương thức `cal_sqrt` được định nghĩa ở Dòng 10-13). Lưu ý, qui ước đặt tên cho lớp theo PEP 8 là “CapWord”. Hơn nữa, lớp cũng có thể có docstring (Dòng 4) như hàm và module.

Các lớp có thể định nghĩa một phương thức đặc biệt, `__init__` (như ở Dòng 6-8), gọi là **phương thức tạo** (constructor), giúp khởi tạo (thường là đặt các giá trị ban đầu cho các thuộc tính) các đối tượng của lớp. Để tạo đối tượng của lớp ta dùng cú pháp tương tự lời gọi hàm:

`<Class>(<Args>)`

Python sẽ tạo một đối tượng thể hiện của lớp `<Class>` và trả về đối tượng đó; nếu lớp không định nghĩa phương thức tạo thì các đối số `<Args>` được để trống; ngược lại, phương thức tạo được gọi với các đối số tương ứng trước khi đối tượng được trả về (ở đây, các đối số hàm tính căn và tên được gán cho thuộc tính `func` và `name` của đối tượng). Trong minh họa ở Phần 14.1 ta đã không dùng phương thức tạo. Nếu ta cho rằng các đối tượng của lớp C đều sẽ có trường `b` thì ta có thể cung cấp phương thức tạo giúp gán giá trị ban đầu cho nó và khi đó ta sẽ viết gọn hơn các lệnh ở Dòng 8.

Như ta đã biết, khi gọi phương thức trên đối tượng, Python truyền thể hiện đó vào tham số đầu tiên của hàm tương ứng trên lớp nên tham số này thường được đặt tên là `self`. Tên này không bắt buộc (ta đã đặt tên là `obj` trong minh họa ở Phần 14.1) nhưng được khuyến cáo dùng vì ý nghĩa của nó.² Đến đây, bạn phải phân biệt được hàm với phương thức nhé. Chẳng hạn, với `x` là thể hiện của lớp `SqrtMethod` thì `x.cal_sqrt` là phương thức còn `x.func` là hàm.

Ngoài ra, lớp thường được dùng ở nhiều nơi nên nó thường được định nghĩa trong một module và được dùng ở các module hay chương trình khác. Đây cũng là lí do mà ta thêm việc kiểm tra ở Dòng 15 để file mã `sqrt_cal.py` có thể được dùng như module lẫn chương trình (xem lại Bài tập 8.10).

Đóng gói thường giúp hiện thực các quan hệ “**có-một**” (“has-a”)³ khi một đối tượng có dùng một hoặc nhiều đối tượng khác bên trong. Chẳng hạn, mỗi đối tượng `SqrtMethod` có một hàm tính căn (`func`) và một chuỗi tên (`name`) bên trong nó. Sự **kết hợp** (composition) nhiều đối tượng thành phần bên trong các đối tượng, đặc biệt khi lồng chứa nhiều mức, thường mang lại sức mạnh to lớn như ta đã biết.

14.3 Tự quản hóa và nhân cách hóa

Phần này tiếp tục các trình diễn trong Phần 9.6 nhưng với cách viết hay hơn nhiều nhờ lớp. Chương trình tại https://github.com/vqhBook/python/blob/master/lesson14/tri_color_wheel.py trình diễn 2 “bánh xe tam sắc” quay đều.

²Nhiều ngôn ngữ lập trình đặt tên cho tham số này là `this` với ý là “chính nó”.

³Chơi chữ, cùng với “is-a” mà ta sẽ biết ở phần sau.

Trong mã `bloom_square.py` (Phần 9.6) và `Pisa_tower.py` (Phần 9.6), các đối tượng trình diễn (hình vuông và quả bóng) hoàn toàn bị động, mọi công việc đều do chương trình tính toán, sắp đặt. Ở đây, đối tượng trình diễn của ta, các “bánh xe tam sắc”, chủ động hơn nhiều nhờ lớp `TriColorWheel`. Các bánh xe này tự nhận biết trạng thái của chúng (tâm, bán kính, màu sắc, vận tốc góc và góc quay); chương trình chỉ tạo ra chúng (Dòng 46-47, 48-50) và tương tác (Dòng 47, 50 đăng kí việc thông báo tăng/giảm vận tốc góc bằng các phím tương ứng; Dòng 40, 41 thông báo việc quay); còn lại, chúng tự mình hoạt động. Chúng là các “hệ tự quản”, chúng có trạng thái, hành vi và có thể tương tác với bên ngoài.

Với lớp và đối tượng, ta thực sự thay đổi tư duy lập trình: *chương trình thay vì quản lý mọi thứ thì bây giờ, nó chỉ là người giám sát, thông báo, tương tác, đồng bộ, ... còn các đối tượng, chúng tự quản lý chúng*. Điều này cũng có thể nói là nhân cách hóa các đối tượng, làm cho các đối tượng sống động như người (hoặc ít nhất, như một sinh vật).

Nếu bạn chưa thấy tầm quan trọng của việc để các đối tượng tự quản chúng thì thử chương trình tại https://github.com/vqhBook/python/blob/master/lesson14/many_wheels.py. Chương trình này cũng quản lý các bánh xe nhưng nhiều hơn (đặt file mã chung thư mục với `tri_color_wheel.py`). Bạn có thể nhấp vào 1 bánh xe để “chọn” nó và tăng giảm vận tốc quay bằng phím mũi tên trái (Left), phải (Right). Thử tưởng tượng, nếu bạn phải quản lý mọi việc của mọi đối tượng, bạn làm sao cho xuể?!

14.4 Kế thừa

Ta đã thấy cách hàm và module giúp giải quyết vấn đề dùng lại các khối lệnh và dữ liệu. Hơn như vậy, lớp giúp ta dùng lại các thuộc tính theo cách tinh vi hơn, gọi là **kế thừa** (inheritance). Chữ kế thừa trong thực tế thường mang 2 nghĩa: (1) thừa hưởng tài sản, của cải do người khác để lại như trong “thừa kế”; (2) thừa hưởng bản chất, đặc tính của giống loài như trong “di truyền”. Có thể nói, theo nghĩa (1) thì việc kế thừa là ở mức cá thể, tức đối tượng; còn theo nghĩa (2) thì việc kế thừa là ở mức “giống loài”, tức là kiểu/lớp. Chữ kế thừa trong lập trình (Python và đa số các ngôn ngữ khác) là ở mức lớp.⁴

Ta đã thấy cách một thể hiện kế thừa thuộc tính từ lớp của nó ở Phần 14.1. Trong phần này, ta sẽ thấy cách một lớp kế thừa thuộc tính từ các lớp khác. Về mặt thuật ngữ, khi lớp *A* kế thừa thuộc tính từ lớp *B*, ta nói *A* là **lớp con** (sub class, derived class, child class) còn *B* là **lớp cha** (super class, base class, parent class). Ta bắt đầu phần này hơi kĩ thuật một chút để làm rõ “bức tranh kế thừa” còn dang dở ở Phần 14.1. Thử minh họa sau:

⁴Thuật ngữ kĩ thuật gọi nghĩa (1) là prototype-based inheritance còn nghĩa (2) là class-based inheritance. Nhiều ngôn ngữ lập trình (trong đó có Python) dùng cách (2) nhưng cũng có một số ngôn ngữ dùng cách (1).

```

1  >>> class B1:
2      a1 = 1
3      def f(self): return "B1"
4
5  >>> class B2:
6      a, a2 = 0, 2
7      def f(self): return "B2"
8
9  >>> class C(B1, B2): a = 5
10
11 >>> c1 = C(); c2 = C(); b = B2(); c2.a = 10
12 >>> [att for att in dir(c1) if not att.startswith("__")]
    ['a', 'a1', 'a2', 'f']
13 >>> print(C.__bases__, C.__bases__[0] is B1)
    (<class '__main__.B1'>, <class '__main__.B2'>) True
14 >>> print(c2.a1, c1.a, c2.a, b.a, c1.f(), b.f(), B2.f(c1))
    1 5 10 0 B1 B2 B2
15 >>> b.a1
    ...
    AttributeError: 'B2' object has no attribute 'a1'
16 >>> print(B1.__bases__, B2.__bases__, object.__bases__)
    (<class 'object'>,) (<class 'object'>,) ()
17 >>> issubclass(C, B1), issubclass(C, B2), issubclass(B1, B2)
    (True, True, False)
18 >>> isinstance(c1, C), isinstance(c1, B1), isinstance(c1, B2)
    (True, True, True)
19 >>> isinstance(b, C), isinstance(b, B1), isinstance(b, B2)
    (False, False, True)

```

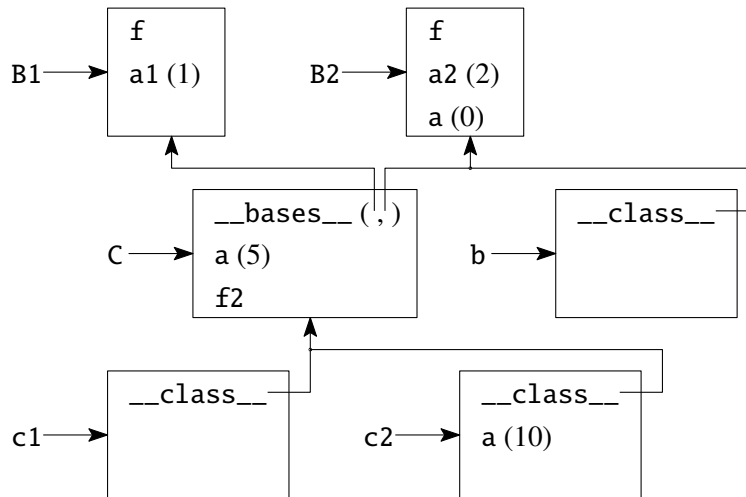
Để khai báo kế thừa, ta nêu tên các lớp cha trong định nghĩa của lớp con như ở Dòng 9. Thường thì mỗi lớp con chỉ kế thừa từ một lớp cha nhưng Python cho phép **đa kế thừa** (multiple inheritance) như trường hợp của lớp C, kế thừa từ cả lớp B1 và B2.⁵ Lưu ý thứ tự liệt kê danh sách các lớp cơ sở (ở đây, B1 trước B2) là rất quan trọng như ta sẽ thấy.

Sau khi tạo đối tượng thể hiện c1 của lớp C ở Dòng 11 thì c1 có các thuộc tính kế thừa từ C và cả B1, B2 như kết quả của Dòng 12 cho thấy. Lưu ý, ở Dòng 11 ta không chỉ tạo các thể hiện mà còn tạo thuộc tính c2.a với giá trị 10. Ta đã biết rằng thuộc tính này được đặt tại không gian tên của c2. “Hiện trạng” các đối tượng sau Dòng 11 được mô tả ở hình dưới.

Cách thức tra thuộc tính “từ dưới lên” ở Phần 14.1 (nếu không có ở đối tượng

⁵Ta lạm dụng thuật ngữ “cha-con” vì “con chỉ có 1 cha”. Tốt hơn nên gọi là **lớp dẫn xuất** với **lớp cơ sở**.

thì lần theo `__class__` tra ở lớp của đối tượng) được mở rộng thành tra thuộc tính “từ dưới lên và từ trái sang”: nếu không thấy ở lớp của đối tượng thì lần theo `__bases__` của lớp tra ở các lớp cơ sở theo thứ tự liệt kê. Lưu ý, việc này có thể được tiếp tục ở mức cao hơn: nếu các lớp cơ sở không có thì tra tiếp ở các lớp cơ sở của các lớp cơ sở, ... cho đến khi thấy hoặc báo lỗi `AttributeError` nếu không.



Bạn suy ngẫm với sự trợ giúp của hình trên để hiểu rõ các kết quả của Dòng 14, 15. Đặc biệt, lời gọi `c1.f()` sẽ dùng `B1.f` chứ không phải `B2.f` vì `B1` được liệt kê trước `B2` trong khai báo danh sách lớp cơ sở của `C`. Để dùng `B2.f`, ta cần chỉ rõ bằng lời gọi `B2.f(c1)`.

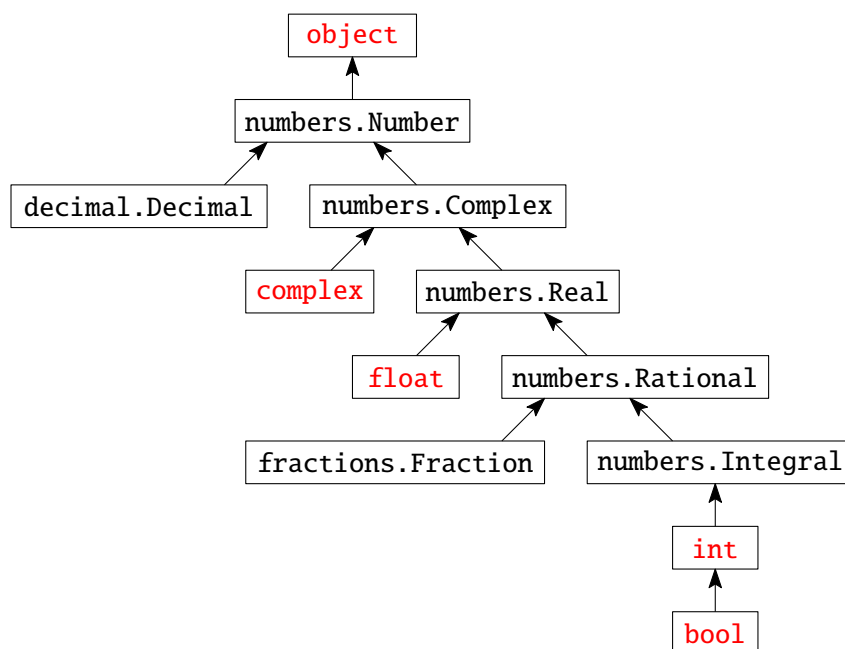
Câu hỏi tự nhiên là việc tra “từ dưới lên” ở trên sẽ dừng lúc nào. Câu trả lời như kết quả Dòng 16 cho thấy: tại lớp `object` là “thủy tổ” của “muôn loài” trong Python (ta đã gặp `object` ở Phần 10.1). Như vậy, để đầy đủ, hình trên phải có thêm lớp `object` ở trên cùng làm lớp cơ sở cho `B1`, `B2`. Về mã, khi khai báo lớp mà không để danh sách lớp cơ sở thì Python tự động để lớp đó kế thừa từ `object`. Dĩ nhiên, lớp `object` chỉ có một số thuộc tính đặc biệt (`__xyz__`) dùng chung cho tất cả các đối tượng.

Hình trên thường được gọi là **sơ đồ kế thừa** (inheritance diagram). Nó mô tả 2 loại quan hệ kế thừa:

- **lớp con** (subclass): cho biết lớp nào kế thừa từ lớp nào. Quan hệ này được hiểu theo nghĩa rộng, khi `A` kế thừa từ `B` thì `A` là lớp con của `B` và `A` cũng là lớp con của mọi lớp mà `B` là lớp con; hơn nữa, mọi lớp đều là lớp con của chính nó. Như vậy, mọi lớp đều là lớp con của `object` và ở đây, lớp `C` là lớp con của các lớp `C`, `B1`, `B2`, `object`. Cách hiểu rộng này thường được làm rõ bằng thuật ngữ “con cháu” hay **“hậu duệ”** (descendant) thay cho lớp con. Hàm `issubclass` kiểm tra quan hệ này.
- **“là một”** (“is-a”): cho biết đối tượng nào là thể hiện của lớp nào. Quan hệ này cũng được hiểu theo nghĩa rộng, khi thể hiện `x` được tạo từ lớp `X` thì `x` là một thể hiện của `X`; hơn nữa, `x` cũng là thể hiện của mọi lớp mà `X` là lớp con.

Như vậy, mọi đối tượng đều là một thể hiện của object và ở đây, c1 là một C, B1, B2, object. Hàm `isinstance` kiểm tra quan hệ này.

Nhân tiện, sau đây là sơ đồ kế thừa các kiểu (lớp) số có sẵn của Python (còn gọi là numeric tower hay numeric hierarchy, <https://docs.python.org/3/library/numbers.html#the-numeric-tower>):



`object`, `complex`, `float`, `int`, `bool` là các lớp dựng sẵn. Các lớp còn lại được định nghĩa trong các module tương ứng.

Dĩ nhiên, lớp trên cùng là `object`. Các lớp `float`, `int`, `bool` mô tả các số thực, số nguyên, luận lý như ta đã biết. Đặc biệt, lớp `bool` kế thừa từ lớp `int`.⁶ Lớp `complex` mô tả **số phức** (complex number), là kiểu số tổng quát hơn số thực như sơ đồ kế thừa cho thấy.⁷ Lớp `decimal.Decimal` hỗ trợ làm việc với số thực cố định và lớp `fractions.Fraction` hỗ trợ kiểu phân số là các lựa chọn thay thế cho số thực chấm động (`float`) trong các trường hợp đặc biệt như ta đã biết.

14.5 Đa hình và nạp chồng toán tử

Ta đã thấy một hiện tượng rất thú vị trong Toán, trong ngôn ngữ và trong Python là **“lạm dụng kí hiệu”** (abuse of notation, abuse of language, abuse of terminology), tức là việc dùng cùng kí hiệu cho các mục đích khác nhau. Tôi đã nói rằng “ngữ cảnh” sẽ giúp Python “phân giải” ý nghĩa chính xác của kí hiệu. Chẳng hạn, trường

⁶Do đó bạn có thể dùng các biểu thức lạ lùng như `True + True` hay `True*10` được các giá trị tương ứng là 2 và 10.

⁷Số phức là một mở rộng rất quan trọng của số thực. Số phức được học trong chương trình Toán phổ thông ở Lớp 12.

hợp dấu - thì tùy theo cách viết (cấu trúc biểu thức) mà nó được hiểu là toán tử trừ 2 ngôi hay toán tử đối 1 ngôi.

Trường hợp phức tạp hơn, dấu + vừa được dùng như là phép cộng trên số vừa được dùng như là phép nối chuỗi thì sao? Python phân giải dựa trên kiểu của đối tượng. Trường hợp thứ nhất xảy ra khi thao tác trên số còn trường hợp thứ hai là trên chuỗi. Thử minh họa sau:

```
1 1 + 2, "1" + "2"
   (3, '12')
2 >>> 1.__add__(2), "1".__add__("2")
   (3, '12')
3 >>> int.__add__(1, 2), str.__add__("1", "2")
   (3, '12')
```

Nếu như cách viết ở Dòng 1 khá bí hiểm thì Dòng 2 (thậm chí Dòng 3) cho thấy rõ bí mật.⁸ Thật sự có 2 phương thức khác nhau nhưng trùng tên `__add__`: một của lớp `int` và một của lớp `str`.⁹ Khi ta dùng toán tử +, Python thay nó bằng phương thức `__add__` và tùy theo đối tượng là thể hiện của lớp nào mà phương thức tương ứng được gọi do hệ quả của cách tra thuộc tính mà ta đã biết ở Phần 14.4. Hiện tượng này được gọi là **nạp chồng toán tử** (operator overloading) trong Python.

Một cách tổng quát, các phương thức nói chung đều có thể được nạp chồng chứ không chỉ là toán tử. Thử minh họa sau:

```
1 >>> class Animal:
2     def speak(self): print("...")
3 >>> class Cat(Animal):
4     def speak(self): print("meo...meo")
5 >>> class Dog(Animal):
6     def speak(self): print("gau...gau")
7 >>> class Cow(Animal):
8     def speak(self): print("umm...bo")
9 >>> pets = [Cat(), Dog(), Cow()]
10 >>> for pet in pets: pet.speak()
meo...meo
gau...gau
umm...bo
```

Cùng tên `speak` nhưng phương thức cụ thể được gọi (`Cat.speak`, `Dog.speak`, `Cow.speak` hay `Animal.speak`) tùy thuộc vào đối tượng nhận lời gọi. Nói một cách bóng bẩy là *cùng một tác động của môi trường, hành vi của các đối tượng lại khác nhau tùy theo lớp của nó*. Hiện tượng này được gọi là **nạp chồng phương thức**.

⁸Lệnh đầu ở Dòng 2 phải viết dấu chấm (.) cách ra khỏi hằng số 1 để khỏi nhập nhầm với dấu chấm thập phân. Python “chưa khôn lắm” trong trường hợp này.

⁹Đĩ nhiên, còn nhiều phương thức có tên `__add__` ở các lớp khác nữa.

(method overloading) mà điển hình nhất là nạp chồng phương thức tạo `__init__` mà ta đã dùng nhiều.¹⁰

Nạp chồng toán tử, hay tổng quát hơn là nạp chồng phương thức, được dùng thường xuyên một cách tích cực trong Python và được gọi là **đa hình** (polymorphism). Lợi ích của đa hình là cực kì lớn như minh họa sau:

```
1 >>> def double(x): return x + x
2
3 >>> double(1), double(1.5), double("hi"), double([1, 2])
(2, 3.0, 'hihi', [1, 2, 1, 2])
```

Bất kì đối tượng nào “biết cộng” ta đều có thể “gấp đôi” nó bằng cách “cộng với chính nó”. Khả năng làm việc mà không cần biết đến chi tiết (không cần biết cụ thể đối tượng nào và cách nó cộng) này còn được gọi là **trừu tượng hóa** (abstraction) và là một công cụ cực kì quan trọng trong khoa học máy tính và lập trình.

Trong mã nguồn `k_max.py` ở Phần 11.2, ta đã viết chương trình tìm số lớn thứ k trong danh sách các số nguyên người dùng đã nhập. Nếu cũng yêu cầu như vậy nhưng cho các số thực thì làm sao? Rất đơn giản, ta chỉ cần thay lời gọi hàm `int` bằng hàm `float` (Dòng 3) để tạo số thực từ chuỗi số người dùng nhập (và dĩ nhiên, thay thông báo nhập ở Dòng 2 là nhập số thực thay cho số nguyên). Cũng lưu ý là, khi đó, người dùng vẫn có thể nhập số nguyên vì số nguyên là trường hợp đặc biệt của số thực.

Nếu cũng yêu cầu như vậy nhưng cho các phân số thì làm sao? Trước hết, ta dùng lớp để tạo một kiểu phân số. Vì kiểu này cũng được dùng cho nhiều chương trình khác nhau (phân số là một dạng số hay dùng không kém gì số thực, số nguyên) nên ta viết riêng nó trong một module. Tạo module `fraction` (file mã `fraction.py`) với nội dung như sau:

<https://github.com/vqhBook/python/blob/master/lesson14/fraction.py>

```
1 import math
2
3 class Fraction:
4     def __init__(self, numer=0, denom=1):
5         if isinstance(numer, str):
6             # đổi số là chuỗi tử/mẫu hoặc chỉ có tử
7             nums = numer.split("/")
8             numer = int(nums[0])
9             denom = 1 if len(nums) == 1 else int(nums[1])
10
11         gcd = math.gcd(numer, denom)
12         if denom < 0:
13             gcd = -gcd
14         self._numer = numer // gcd # tử số
```

¹⁰Còn một thứ gần gũi nữa là **nạp chồng hàm** (function overloading) nhưng nói một cách chặt chẽ thì Python không có thứ này.


```

15         self._denom = denom // gcd # mẫu số
16
17     def __repr__(self):
18         if self._denom == 1:
19             return str(self._numer)
20         else:
21             return "%d/%d" % (self._numer, self._denom)
22
23     def __float__(self):
24         return self._numer / self._denom
25
26     def __lt__(self, other):
27         return float(self) < float(other)

```

Phân số (fraction) hay **số hữu tỉ** (rational number) là số có dạng $\frac{a}{b}$ với a, b là các số nguyên, $b \neq 0$; a được gọi là **tử số** (numerator), b được gọi là **mẫu số** (denominator). Chẳng hạn, phân số $1/3$ (tức là $\frac{1}{3}$) có tử số là 1 và mẫu số là 3. Về mặt Toán học, mọi số nguyên đều là phân số (với mẫu số là 1) và mọi phân số đều là số thực.

Tên các phương thức của lớp `Fraction` có dạng khá lạ `__xyz__`. Python dùng tên dạng này với ý nghĩa đặc biệt như sau:¹¹

- `__init__`: là phương thức tạo mà ta đã biết. Ở đây, nó cho phép tạo phân số từ một chuỗi biểu diễn phân số có dạng là tử/mẫu với tử, mẫu là các chuỗi số nguyên. Nó cũng chấp nhận chuỗi chỉ gồm 1 số nguyên mô tả tử số (còn mẫu số sẽ là 1). Trực tiếp hơn, nó nhận 2 số nguyên mô tả tương ứng tử số, mẫu số. Phương thức này cũng thực hiện việc tối giản phân số bằng cách chia (nguyên) tử và mẫu cho ước số chung lớn nhất của chúng. Hơn nữa, nó cũng thực hiện việc “chuẩn dấu”, tức là việc giữ cho mẫu là số nguyên dương.
- `__repr__`: phương thức này được dùng để tạo ra chuỗi biểu diễn cho đối tượng, chẳng hạn trong kết xuất hay `print` (mà thực chất là kết quả của lời gọi hàm `str`). Ở đây, nó đưa ra biểu diễn của phân số dạng tử/mẫu. Trường hợp phân số có mẫu là 1, nó trùng với số nguyên có giá trị là tử số, khi đó ta chỉ đưa ra biểu diễn như là số nguyên.
- `__float__`: phương thức này giúp đối tượng được chuyển thành số thực `float` (chẳng hạn trong lời gọi hàm `float`). Ở đây, ta đưa ra số thực (gần đúng) bằng cách chia tử số cho mẫu số bằng phép chia thông thường.
- `__lt__`: phương thức này được dùng để kiểm tra quan hệ nhỏ hơn trên 2 đối tượng (chẳng hạn trong biểu thức dùng toán tử `<`). Ở đây, ta trước hết tạo số thực gần đúng cho phân số và so sánh dựa trên 2 số thực đó. Lưu ý, số dĩ ta gọi được hàm `float` trên phân số vì ta đã “trang bị” phương thức `__float__` (như đã nói trên) cho lớp phân số này.

¹¹Tên dạng này vẫn hợp lệ bình thường nhưng vì Python dùng chúng với ý nghĩa đặc biệt nên ta tránh đặt tên như vậy (dĩ nhiên, trừ khi ta cố ý đặt có mục đích).

Ngoài ra, lớp `Fraction` dùng 2 trường `_numer` và `_denom` để chứa tử số và mẫu số. Bạn có thấy cách đặt tên đặc biệt cho nó (bắt đầu bằng dấu `_`), ta sẽ biết dụng ý của việc này ở Phần 14.6.

Sau khi đã có lớp phân số định nghĩa như trên, ta sửa lại chương trình `k_max.py` để làm việc với phân số như sau:

```

1  https://github.com/vqhBook/python/blob/master/lesson14/k\_max.py
2  from fraction import Fraction
3
4  nums = []
5  while text := input("Nhập phân số (Enter để dừng): "):
6      nums.append(Fraction(text))
7
8  nums.sort(reverse=True)
9  print("Các số đã nhập theo thứ tự giảm dần là:")
10 print(nums)
11
12 k = int(input("Bạn muốn tìm số lớn thứ mấy? "))
13 if 0 < k <= len(nums):
14     print(f"Số lớn thứ {k} đã nhập là {nums[k - 1]}")

```

Tương tự như việc sửa cho số thực, ta cũng chỉ cần sửa đúng chỗ tạo số từ chuỗi (Dòng 5), thay vì gọi `int` để tạo số nguyên, ta gọi `fraction.Fraction` để tạo phân số (mà chính là gọi phương thức tạo `__init__` của lớp `Fraction`). Ngoài ra, chương trình cũng đã xuất danh sách phân số đã nhập (sau khi sắp giảm dần) để dễ theo dõi (Dòng 9). Cũng lưu ý là người dùng vẫn có thể nhập số nguyên vì số nguyên là trường hợp đặc biệt của phân số! (Bạn biết rồi đó, thật ra, ta có được điều này là nhờ phương thức tạo `__init__` của lớp `Fraction` chấp nhận chuỗi chỉ gồm 1 số nguyên).

Câu hỏi đặt ra là: tại sao ta chỉ cần sửa chương trình `k_max.py` ít như vậy? hay tại sao phân số (lớp `Fraction`) lại hòa nhập vào “hạ tầng Python” nhanh như vậy? Chẳng hạn, tại sao phân số cũng được xuất ra đẹp như vậy hay tại sao phương thức `sort` có thể sắp xếp được các phân số? Câu trả lời chính ở việc nạp chồng các phương thức đặc biệt mà ta đã nói ở trên của lớp `Fraction`. Chẳng hạn, nhờ nạp chồng phương thức `__lt__` (tức là toán tử `<`) mà ta có thể so sánh nhỏ hơn được trên các phân số (mà chỉ cần như vậy, phương thức `sort` có thể dùng để sắp xếp các phân số).

Thật vậy, chạy module `fraction` để có định nghĩa lớp `Fraction` và tương tác tiếp trong phiên làm việc đó như sau:

```

===== RESTART: D:\Python\lesson14\fraction.py =====
1 >>> print(a := Fraction("10/4"), float(a))
5/2 2.5
2 >>> print(b := Fraction("2"), float(b))

```

```

2 2.0
3 >>> print(a < b, a < Fraction(3))
False True
4 >>> a + b
...
TypeError: unsupported operand type(s) for +: 'Fraction'
and 'Fraction'

```

Ta như vậy đã thực hiện được việc tạo phân số từ chuỗi hoặc số nguyên (phương thức `__init__`), kết xuất “đẹp” cho phân số (`__repr__`), tạo số thực float từ phân số (`__float__`) và so sánh bé hơn (`__lt__`). Lưu ý, việc cộng 2 phân số ở Dòng 4 báo lỗi thực thi vì hiện giờ lớp phân số (`Fraction`) của ta chưa hỗ trợ thao tác cộng (nói cách khác, chưa nạp chồng toán tử `+`). Để làm điều này, ta bổ sung thêm phương thức sau vào lớp `Fraction` (<https://github.com/vqhBook/python/blob/master/lesson14/fraction2.py>):

```

29 def __add__(self, other):
30     r_numer = self._numer*other._denom +
        ↪ self._denom*other._numer
31     r_denom = self._denom*other._denom
32     return Fraction(r_numer, r_denom)

```

Chạy lại tương tác trên, ta sẽ có kết quả `a + b` là phân số `9/2`. Phương thức `__add__` như vậy giúp nạp chồng toán tử cộng (`+`) mà ở đây, để cộng phân số, ta dùng cách qui đồng mẫu số quen thuộc:

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}$$

Chúc mừng, bạn đã định nghĩa được một lớp rất quan trọng để biểu diễn phân số. Chia buồn, Python thật ra đã làm việc này, thư viện chuẩn của Python đã định nghĩa lớp `Fraction` trong module `fractions` (xem sơ đồ lớp ở Phần 14.4 để biết “vị trí” của lớp này), xem Bài tập 3.9. Như vậy, không cần module của ta (`fraction`) chương trình `k_max.py` trên có thể dùng module `fractions` cung cấp sẵn để làm việc với phân số. Nghĩa là, ta chỉ cần sửa Dòng 1 trong file mã `k_max.py` thành:

```

1 from fractions import Fraction

```

mà không cần “mắc công” viết module `fraction`. Dĩ nhiên, lớp `Fraction` của “người ta” “xịn” hơn của mình nhiều!¹²

Bạn tra cứu thêm các phương thức đặc biệt của Python tại <https://docs.python.org/3/reference/datamodel.html#special-method-names>.

¹²Cũng chưa hẳn à, bạn chạy thử sẽ biết.

14.6 Tính riêng tư và che dấu dữ liệu

Trong định nghĩa của lớp `Fraction` ở mã `fraction.py` Phần 14.5, ta đã dùng trường `_numer`, `_denom` để lưu tử và mẫu của một phân số. Điều đặc biệt chính là tiền tố `_` của tên. Về mặt kỹ thuật, không có gì đặc biệt cả vì đó là cách đặt tên hợp lệ bình thường. Chúng đặc biệt ở qui ước sử dụng: các tên bắt đầu bằng dấu `_` được cộng đồng Python qui ước chung là **tên riêng** (private name). Ở đây, ta dùng tên riêng vì không muốn **bên ngoài** (client), nơi dùng các đối tượng của lớp `Fraction`, trực tiếp truy cập chúng.

Tại sao vậy? Nếu như việc đọc là chấp nhận được thì việc thay đổi trực tiếp chúng là rất nguy hiểm vì theo logic, lớp `Fraction`, luôn để tử số, mẫu số ở “dạng chuẩn”.¹³ Nếu người dùng can thiệp thì qui tắc này không còn đúng nữa.¹⁴ Đây cũng chính là lý do mà các nhà sản xuất đều cấm người dùng “mổ xẻ” các sản phẩm của họ.¹⁵

Không đâu xa lạ, lớp của các chú rùa (`turtle.Turtle`) cũng dùng kỹ thuật này. Bạn hãy tạo một chú rùa (thể hiện của lớp `turtle.Turtle`) rồi dùng hàm `dir` để xem các thuộc tính của nó sẽ thấy. Hoặc tốt hơn, bạn xem mã nguồn của lớp `Turtle` ở file mã `turtle.py` (xem Bài tập 14.16).

Thuật ngữ đóng gói (encapsulation) ở Phần 14.2 không chỉ có nghĩa là kết hợp thao tác (hàm) vào dữ liệu mà còn, mạnh hơn, ám chỉ việc che chắn, bảo vệ dữ liệu. Cụ thể, thông tin trạng thái của đối tượng không thể trực tiếp truy cập, sửa đổi từ bên ngoài mà phải qua các phương thức của nó. Các phương thức này ngoài việc truy cập thông tin trạng thái còn chịu trách nhiệm thay đổi chúng. Rất tiếc, Python không hỗ trợ ngữ nghĩa “đầy đủ” này của đóng gói.

Lưu ý, qui ước đặt tên trên chỉ là qui ước, nó có tác dụng với người ngay hơn là kẻ gian! Thử minh họa sau:

```

1  >>> class C:
2      def __init__(self, x, y):
3          self._x, self.__y = x, y
4      def __repr__(self):
5          return f"{self._x}, {self.__y}"
6
7
8  >>> print(c := C(1, 2))
1, 2
9  >>> c._x = 0; c.__y = 0; print(c)
0, 2
10 >>> dir(c)
```

¹³Thực tế, nhiều thông tin còn riêng tư đến mức không được phép đọc như tuổi, tiền lương, ...

¹⁴Logic kiểm tra bằng ở file mã `fraction3.py` không còn đúng nữa.

¹⁵Các chính sách bảo hành đều có qui định không bảo hành với sản phẩm đã được tháo rời hay chỉnh sửa bên trong.

```

11 ['_C__y', ..., '__y', '_x']
>>> c._x = 5; c.__y = 10; c._C__y = 15; print(c)
5, 15

```

Việc sửa giá trị thuộc tính `_x` của các thể hiện của lớp C từ bên ngoài lớp là được phép. Đặc biệt, có vẻ việc sửa giá trị thuộc tính `__y` là không thành công như kết quả của Dòng 9 cho thấy. Bí mật nằm ở chỗ, các tên bắt đầu bằng `__` mà không kết thúc là `__` dùng bên trong lớp sẽ được Python đổi tên theo cơ chế “xào tên” (name mangling). Cụ thể, tên `__y` đã được đổi lại thành tên `_C__y` như kết quả Dòng 10 cho thấy. Do đó lệnh gán `c.__y = 10` đã tạo một trường mới là `__y` cho `c`. Việc truy cập `__y` ở Dòng 5 cũng dùng tên đã “xào” là `_C__y` do lệnh này nằm trong định nghĩa lớp C.

Mặc dù có vẻ giúp “bảo vệ” các tên riêng nhưng cơ chế “xào tên” được dùng với mục đích khác (ta sẽ tìm hiểu thêm sau). Hơn nữa, bên ngoài vẫn có thể sửa các trường với tên đã “xào” như Dòng 11. Rất tiếc, Python không có khả năng **che dấu dữ liệu** (data hiding) thật sự cho các thành phần riêng tư của đối tượng mà điều này khá quan trọng trong nhiều tình huống. Dĩ nhiên, ta vẫn có thể vượt qua được với những giải pháp khác nhau (ta sẽ thấy sau).

14.7 Lập trình hướng đối tượng

Việc thiết kế và lập trình dùng mô hình đối tượng và lớp với các đặc trưng như đóng gói, kế thừa, đa hình, ... thường được gọi là mô thức **lập trình hướng đối tượng** (Object-Oriented Programming, OOP). Theo mô thức này, *chương trình là một hệ thống các đối tượng tương tác với nhau và với người dùng*.

Trong mã `turtle_race.py` (Phần 10.2), ta có một “loại rùa” rất đặc biệt là Pizza, hơn nữa, 4 chú ninja rùa cũng đặc biệt không kém. Dùng OOP ta viết lại mã một cách hay hơn bằng chương trình tại https://github.com/vqhBook/python/blob/master/lesson14/turtle_race.py. Chương trình định nghĩa 2 lớp mới là `PizzaTurtle` và `NinjaTurtle` đều kế thừa từ lớp `turtle.Turtle`. Do đó, mọi miếng Pizza (các đối tượng tạo từ lớp `PizzaTurtle`) và mọi chú ninja rùa (các đối tượng tạo từ lớp `NinjaTurtle`) đều là rùa (`turtle.Turtle`). Quan trọng hơn, chúng là các chú rùa đặc biệt (chẳng hạn, các `PizzaTurtle` có hình dáng miếng Pizza, có thể được người dùng “kéo-thả”, có 2 trường mới là bán kính và giá trị, ...).

Đặc biệt lưu ý, *lớp con không kế thừa phương thức tạo từ lớp cha* nên trong phương thức tạo của lớp con (như Dòng 8-12 của lớp `PizzaTurtle`), ta thường gọi phương thức tạo của lớp cha (Dòng 9). Hàm dựng sẵn `super` trả về “đối tượng cha” mà từ đó ta có thể dùng để gọi các phương thức (hay truy cập các thuộc tính) đã có từ lớp cha. Như đã biết, ta cũng có thể gọi bằng cách chỉ rõ không gian tên, chẳng hạn, có thể thay Dòng 9 trên bằng:

```

9 t.Turtle.__init__(self)

```

Để quản lý tất cả các miếng pizza (tức là tất cả các thể hiện của lớp `PizzaTurtle`), lớp này dùng danh sách `all_pizzas` mà mỗi khi một miếng pizza được tạo thì nó được thêm vào danh sách này (Dòng 14). Thuộc tính này thường được gọi là **thuộc tính của lớp** (class attribute) vì nó có giá trị chung, dùng cho mọi thể hiện của lớp; ngược lại, các thuộc tính như `radius` hay `value` được gọi là **thuộc tính của thể hiện** (instance attribute) vì nó có giá trị riêng cho từng thể hiện. Về mặt kĩ thuật, `radius` (`value`, ...) nằm ở không gian tên của thể hiện còn `all_pizzas` nằm ở không gian tên của lớp.

Tương tự, bạn phân tích để hiểu rõ hơn lớp `NinjaTurtle` và toàn bộ chương trình. Các lớp `PizzaTurtle` và `NinjaTurtle` cũng thể hiện tính tự quản rất cao vì chúng tự quản lý các đối tượng của chúng. Để thấy rõ lợi ích của việc này, bạn thử tạo nhiều miếng Pizza và nhiều chú rùa hơn như chương trình tại https://github.com/vqhBook/python/blob/master/lesson14/turtle_race2.py.

14.8 Đối tượng duyệt được và bộ duyệt

Đến giờ, ta đã “duyet qua” rất nhiều “thứ” như dãy, tập hợp, từ điển và đối tượng `range` (Phần 7.1, 7.7, 11.1). Vậy điểm chung của những thứ này là gì để ta có thể duyệt qua chúng? Về logic, chúng là các đối tượng **duyet được** (iterable): chúng cho phép ta truy cập lần lượt các phần tử. Về kĩ thuật, lớp của các đối tượng này có nạp chồng phương thức đặc biệt `__iter__` mà khi được gọi sẽ trả về một **bộ duyệt** (iterator), là đối tượng hỗ trợ việc duyệt trên đối tượng được duyệt.

Để là bộ duyệt, lớp của đối tượng phải nạp chồng phương thức đặc biệt `__next__` mà mỗi lần được gọi sẽ trả về lần lượt các phần tử của đối tượng được duyệt hoặc phát sinh lỗi thực thi `StopIteration` khi hết phần tử (ta sẽ tìm hiểu thêm lỗi thực thi này ở Bài ??). Cách thức tương tác này giữa đối tượng được duyệt và bộ duyệt thường được gọi là giao thức duyệt (iteration protocol). Lưu ý, một đối tượng có thể đóng vai trò cả 2, nghĩa là nó có thể “tự duyệt” (nó có cả 2 phương thức trên mà `__iter__` trả về chính nó).¹⁶

Python tự động thực hiện giao thức duyệt trong một số cấu trúc mã như lệnh lặp `for`, bộ tạo danh sách, hàm `list`, ... Ta cũng có thể tự mình thực hiện tương minh giao thức duyệt bằng cách gọi các hàm `iter` (hàm này gọi phương thức `__iter__`) và `next` (hàm này gọi phương thức `__next__`) như minh họa sau:

```
1 >>> iterable = range(5)
2 >>> for i in iterable: print(i, end=" ")
0 1 2 3 4
3 >>> iterator = iter(iterable)
4 >>> while iterator: print(next(iterator), end=" ")
```

¹⁶Đây cũng là nguyên nhân làm lẫn lộn 2 khái niệm iterable và iterator. Hơn nữa, về mặt kĩ thuật, một iterator cũng được yêu cầu cài đặt phương thức `__iter__` để trả về chính nó, do đó, các iterator cũng là iterable!

```
0 1 2 3 4
... StopIteration
```

Ở Dòng 4, ta bị lỗi thực thi `StopIteration` khi duyệt hết danh sách nhưng đó là một phần của giao thức duyệt (mọi thứ sẽ tự nhiên trong Bài ?? khi ta học được cách xử lý lỗi này).

Ứng dụng giao thức duyệt trên, ta cung cấp một bộ duyệt giúp duyệt qua các số nguyên “có thể vô hạn”. Tạo module `range_iterator` có mã như sau:

https://github.com/vqhBook/python/blob/master/lesson14/range_iterator.py

```
1 class range:
2     def __init__(self, start=0, stop=None, step=1):
3         self._start = start
4         self._stop = stop
5         self._step = step
6         self._cur = self._start
7
8     def __iter__(self):
9         # self._cur = self._start
10        return self
11
12    def __next__(self):
13        if (self._stop != None and
14            (self._step > 0 and self._cur >= self._stop
15             or self._step < 0 and self._cur <= self._stop)):
16            raise StopIteration
17        self._cur = (prev := self._cur) + self._step
18        return prev
```

Giả sử file mã được để ở thư mục `D:\Python\lesson14`, chạy IDLE từ đầu và tương tác như sau:

```
1 >>> import os; os.chdir(r"D:\Python\lesson14")
2 >>> import range_iterator
3 >>> [i for i in range_iterator.range(1, 11)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4 >>> list(range_iterator.range(0, -11, -2))
[0, -2, -4, -6, -8, -10]
5 >>> for i in range_iterator.range(): print(i)
0
1
2
...
```

Lưu ý, về “lý thuyết”, lệnh duyệt ở Dòng 5 duyệt qua “tất cả” các số tự nhiên nếu bạn đợi được :) (Dĩ nhiên, bạn có thể nhấn `Ctrl+C` để dừng.) Nhận xét là lớp

`range_iterator.range` có cách dùng rất giống lớp có sẵn `range`.¹⁷

Đặc biệt, bộ duyệt trên của ta cho phép duyệt qua tất cả các số tự nhiên! Rõ ràng, nếu dùng danh sách, ta không thể có được điều này vì bộ nhớ là hữu hạn nên danh sách phải hữu hạn. Ở đây, nhờ cách **lượng giá muộn** (lazy evaluation) hay **“gọi khi cần”** (call-by-need), mà ta có được khả năng này (xem lại Phần 7.6). Cụ thể, ta chỉ cần nhớ số hiện tại (và các tham số) mà khi được yêu cầu ta mới “sinh” số kế tiếp. Bạn có thấy ghi chú ở Dòng 9 không, khác biệt giữa việc dùng và không dùng lệnh đó là gì?

Các hàm dựng sẵn `map` (Phần 11.6), `reversed`, `zip` (Phần 12.1), `enumerate` (Phần 12.1), `filter` (Phần 11.3) cũng trả về các bộ duyệt (thay vì danh sách đầy đủ) với cơ chế hoạt động tương tự mà mục đích chính là tăng tính hiệu quả (tiết kiệm bộ nhớ và thời gian thực thi). Các đối tượng duyệt được (iterable) có thể được dùng ở rất nhiều nơi trong Python. Chẳng hạn, các thao tác thu danh sách ở Bài tập 11.2 thật ra dùng được với mọi đối tượng duyệt được mà danh sách chỉ là một trường hợp.¹⁸

Tóm tắt

- Dữ liệu và thao tác có thể được đóng gói chung thành các thành phần của lớp.
- Sau khi lớp được định nghĩa, ta có thể tạo và dùng các thể hiện của lớp tương tự việc có thể dùng hàm (gọi hàm) sau khi hàm được định nghĩa.
- Phương thức tạo giúp khởi động các thể hiện của lớp.
- Các đối tượng tự quản lý trạng thái của mình và tương tác với nhau qua phương thức. Chương trình là một hệ thống các đối tượng tương tác với nhau.
- Kế thừa giúp lớp con được dùng lại các thuộc tính của lớp cha. Hơn nữa, lớp con có thể mở rộng, đề, xóa, nạp chồng các thuộc tính của lớp cha.
- Sơ đồ kế thừa thể hiện mối quan hệ kế thừa giữa các lớp. Lớp `object` là lớp cao nhất trong sơ đồ kế thừa của Python.
- Việc nạp chồng (tạo mới hay ghi đè) các phương thức đặc biệt (`__xxx__`) giúp lớp hòa nhập với hệ thống lớp có sẵn của Python.
- Python không hỗ trợ việc che dấu các thành phần riêng tư của lớp, đối tượng.
- Lập trình hướng đối tượng với các đặc trưng như đóng gói, kế thừa, đa hình, ... là mô thức lập trình hiện đại, mạnh mẽ và được ưa chuộng khi viết các chương trình lớn, phức tạp.
- Giao thức duyệt là cách Python hỗ trợ việc duyệt hiệu quả.

¹⁷Ta hay gọi `range` là hàm nhưng đúng hơn, nó là lớp và lời gọi `range` là lời gọi đối tượng lớp để tạo thể hiện của lớp (mà ta đã gọi là đối tượng `range`). Hơn nữa, lớp `range` cũng hỗ trợ các thao tác khác để nó trở thành một dãy như danh sách, bộ và chuỗi.

¹⁸Bạn tra cứu lại một lần nữa các hàm `all`, `any`, `max`, `min`, `sum` để thấy chúng nhận đối tượng duyệt được (iterable).

Bài tập

14.1 Xem lại Phần 12.4 để thấy cách dùng tập hợp giải quyết vấn đề trùng số cho bài toán “k-max”. Sửa lớp Fraction trong file mã fraction.py để dùng được cách này.

Gợi ý: nạp chồng phương thức `__hash__` và toán tử so sánh bằng (phương thức `__eq__`).¹⁹

14.2 Khác biệt giữa `__repr__` và `__str__`. Cả 2 phương thức đặc biệt, `__repr__` và `__str__`, đều được dùng để trả về chuỗi biểu diễn (string representation) của đối tượng. Tuy nhiên, chuỗi trả về từ `__repr__` thường “hình thức” và có thể được dùng để tạo lại đối tượng bằng hàm `eval`; ngược lại, chuỗi trả về từ `__str__` thường “thân thiện” và thường được dùng để “in ấn”. Lưu ý, `object.__str__` gọi và trả về `object.__repr__()` nên nếu lớp không nạp chồng phương thức `__str__` thì nó trả về kết quả từ `__repr__`. Thử minh họa sau để hiểu rõ hơn:

```
1 >>> from fractions import Fraction
2 >>> a = Fraction(2, 6)
3 >>> print(a)
1/3
4 >>> a, repr(a), a.__repr__(), str(a)
(Fraction(1, 3), 'Fraction(1, 3)', 'Fraction(1, 3)', '1/3')
5 >>> eval(repr(a))
Fraction(1, 3)
```

- Tôi đã hơi lười biếng khi viết mã fraction.py ở Phần 14.5, cụ thể, tôi chỉ nạp chồng phương thức `__repr__` cho lớp Fraction và trả về chuỗi biểu diễn “đẹp”. Viết lại lớp Fraction với đầy đủ 2 phương thức `__repr__` (trả về chuỗi hình thức) và `__str__` (trả về chuỗi thân thiện) như lớp `fractions.Fraction` của thư viện chuẩn.
- Tập thói quen viết đầy đủ và đúng ý nghĩa sử dụng 2 phương thức `__repr__` và `__str__` khi định nghĩa các lớp.

14.3 Hoàn chỉnh lớp Fraction của module fraction ở Phần 14.5 để cung cấp thêm các phép toán số học, phép toán so sánh khác. Sau đó dùng lớp này, làm lại Bài tập 7.2 với các phân số.

14.4 Biểu thức chứa biến. Trong Bài tập 2.2 ta đã thực hiện việc tính giá trị của một biểu thức có chứa biến (còn gọi là chữ) tại giá trị được cho của các biến (các biểu thức M, N với biến x, y). Viết lớp Expression đóng gói biểu thức chứa biến với các thao tác phù hợp. Lớp này gồm các thành phần chính sau:

- Trường biểu thức: chuỗi mô tả một biểu thức chứa biến, hằng số, toán tử, dấu đóng mở ngoặc hợp lệ như biểu thức số học của Python. Để đơn giản,

¹⁹Xem đáp án tại <https://github.com/vqhBook/python/blob/master/lesson14/fraction3.py>.

biến phải là chữ cái thường a, b, c, \dots, z . Chẳng hạn biểu thức $x^2 + 4y^2 - 4xy$ được mô tả bởi chuỗi " $x**2 + 4*y**2 - 4*x*y$ " gồm 2 biến là x, y .

- Trường tên: chuỗi tên (kí hiệu gọn) cho biểu thức (như M, N trong Bài tập 2.2).
- Phương thức trả về chuỗi biểu diễn dạng $\langle \text{Tên} \rangle = \langle \text{Biểu thức} \rangle$.
- Phương thức cho biết danh sách biến (sắp theo thứ tự alphabet).
- Phương thức tính giá trị của biểu thức tại giá trị được cho của các biến. Đối số nên là một từ điển với các cặp tên biến : giá trị (xem Bài tập 12.6).

14.5 Viết lớp `Vector2D` đóng gói một cặp số với các thao tác phù hợp (xem Bài tập 12.2). Nên dùng các kĩ thuật nạp chồng phương thức, toán tử.

14.6 Viết lớp `Time` đóng gói thời điểm trong ngày với các thao tác phù hợp (xem Bài tập 6.8).

14.7 Viết lớp `NaturalNumber` đóng gói số nguyên không âm, biểu diễn bằng danh sách các kí số, với các thao tác phù hợp (xem Bài tập 11.5).

14.8 Viết lớp `Polynomial` đóng gói một đa thức, biểu diễn bằng danh sách các hệ số, với các thao tác phù hợp (xem Bài tập 11.4).

14.9 Viết lớp `Polygon` đóng gói một đa giác, biểu diễn bằng danh sách các đỉnh, với các thao tác phù hợp (xem Bài tập 11.9).

14.10 Viết lớp đóng gói bảng dữ liệu với các thao tác thống kê phù hợp (xem Bài tập 11.10, 12.10).

14.11 Đối tượng gọi được. Ta đã thực hiện việc gọi hàm rất nhiều tới giờ. Một cách tổng quát, lời gọi hàm là một biểu thức dùng **toán tử gọi** (call operator) với cú pháp:²⁰

$$\langle \text{Obj} \rangle (\langle \text{Args} \rangle)$$

Các đối tượng ($\langle \text{Obj} \rangle$) có thể được gọi như vậy là **đối tượng gọi được** (callable object) mà ta đã biết là hàm (thực thi thân hàm), phương thức (thực thi thân hàm với đối số đầu tiên là đối tượng được gọi) và lớp (tạo thể hiện của lớp và gọi phương thức tạo nếu có). Hơn nữa, bất kì đối tượng nào cũng có thể gọi được nếu nó nạp chồng toán tử gọi, cụ thể là phương thức `__call__`.

Trong mã `write_char.py` ở Phần 14.1, ta đã dùng lớp để lưu giữ thông tin trạng thái thay cho bao đóng hàm. Nay dùng đối tượng gọi được, ta có cách viết hay hơn như chương trình tại https://github.com/vqhBook/python/blob/master/lesson14/write_char2.py. Về mã, ta chỉ đơn giản đặt lại tên phương thức `write` trong lớp `WriteChar` ở file mã `write_char.py` thành `__call__` nhưng hiệu quả mang lại là ta có thể dùng chính đối tượng của lớp `WriteChar` như hàm (Dòng 19).

²⁰Toán tử gọi có độ ưu tiên cao hơn các toán tử số học.

Tương tự, sửa lại lớp `SqrtMethod` trong mã `sqrt_cal.py` Phần 14.2 để các đối tượng thể hiện của lớp có thể gọi được mà khi gọi sẽ tính căn của một số hoặc danh các sách số (chính là công việc của phương thức `cal_sqrt`). Đồng thời sửa lại cách dùng và chạy thử.

14.12 Bộ với thành phần được đặt tên. Kiểu bộ (tuple) của Python chỉ cho phép truy cập các thành phần theo chỉ số. Đôi khi ta muốn đặt tên cho các thành phần và truy cập các thành phần theo tên (khi đó, các thành phần thường được gọi là **trường** (field)). Hàm `collections.namedtuple` cho phép tạo các lớp cung cấp khả năng này. Thử minh họa sau:

```
1 >>> from collections import namedtuple
2 >>> Vector2D = namedtuple("Vector2D", ["x", "y"])
3 >>> print(v1 := Vector2D(5, 10), v2 := Vector2D(y=10, x=5))
Vector2D(x=5, y=10) Vector2D(x=5, y=10)
4 >>> print(type(v1), type(Vector2D), v1.__class__ is
↳ Vector2D)
<class '__main__.Vector2D'> <class 'type'> True
5 >>> print(v1[0], v1.x, v1 == v2)
5 5 True
6 >>> print("x = %d, y = %d" % v1)
x = 5, y = 10
7 >>> v1.x = 0
...
AttributeError: can't set attribute
8 >>> v3 = v1._replace(x = 0); print(v3, v1)
Vector2D(x=0, y=10) Vector2D(x=5, y=10)
```

Đặc biệt lưu ý, hàm `collections.namedtuple` trả về một lớp mà từ đó ta có thể tạo các thể hiện có hoạt động tương tự bộ (dãy bất biến) nhưng thêm khả năng truy cập thành phần theo tên trường đã chọn.²¹

Dùng `collections.namedtuple` ta có thể mô tả các dòng của các bảng dữ liệu đẹp hơn bằng các bộ với các thành phần có tên trùng với tên cột (trường) của bảng. Chẳng hạn, minh họa ở Phần 12.2 có thể được viết lại như chương trình tại https://github.com/vqhBook/python/blob/master/lesson14/students_list.py.

- (a) Tra cứu để hiểu rõ `namedtuple` tại <https://docs.python.org/3/library/collections.html#collections.namedtuple>.
- (b) Với cách mô tả các dòng bằng các bộ được đặt tên thành phần ở trên, viết hàm tra cứu sinh viên theo mã số, theo tên.
- (c) Làm lại Bài tập 12.7a với các bộ được đặt tên thành phần phù hợp.

²¹Ta thường tạo lớp bằng cách “cồng kềnh” là dùng lệnh `class`. Ví dụ này cho thấy lớp cũng có thể được tạo rất “ngắn gọn” bằng một lời gọi hàm (tương tự như khả năng tạo hàm từ biểu thức `lambda` thay cho lệnh `def`).

14.13 Kiểu bản ghi. Một dòng dữ liệu trong bảng dữ liệu thường được gọi là một **bản ghi** (record). Nhiều ngôn ngữ hỗ trợ kiểu dữ liệu để làm việc với các bản ghi gọi là **kiểu record** hay **kiểu cấu trúc** struct type. Trong Python ta dễ dàng tạo kiểu này, chẳng hạn, như module `record.py` tại <https://github.com/vqhBook/python/blob/master/lesson14/record.py>. Dùng module này ta có thể viết lại minh họa ở Phần 12.2 như chương trình tại https://github.com/vqhBook/python/blob/master/lesson14/students_list2.py.

(a) Nghiên cứu mã `record.py` ở trên.

(b) Làm lại Bài tập 14.12(b)-(c) dùng `Record` thay cho `namedtuple`.

14.14 Dãy Fibonacci. **Dãy Fibonacci** (Fibonacci sequence) là dãy số được xác định theo qui tắc:

- $F_0 = 0, F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}, n \geq 2$

Nói nôm na, dãy bắt đầu với 0, 1 và tiếp tục với qui tắc “số sau là tổng của 2 số liền trước”. Vài số hạng đầu tiên của dãy là: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Viết lớp cung cấp bộ duyệt cho dãy Fibonacci duyệt qua các cặp (n, F_n) với $n = 0, 1, 2, \dots$

Gợi ý: cần nhớ những gì để sinh số kế tiếp? (Số hạng thứ mấy và 2 số hạng trước đó.)

14.15 Tự viết lấy hàm thay cho các hàm dựng sẵn sau: `map`, `reversed`, `zip`, `enumerate`, và `filter`.

14.16 Tra cứu các lớp sau đây (dùng File → Open Module... trong IDLE để xem mã nguồn của module tương ứng và tra cứu thêm từ Python Docs và các nguồn “dễ đọc” khác):²²

- `fraction.Fraction` (đối chiếu với lớp `Fraction` ở Bài tập 14.3)
- `turtle.Vector2D` (đối chiếu với lớp `Vector2D` ở Bài tập 14.5)
- `turtle.Turtle`
- `datetime.date`, `datetime.time`

²²Mã nguồn gồm nhiều kĩ thuật mà bạn có thể chưa được học nên đoạn mã nào khó thì bỏ qua. Cố gắng nắm được bức tranh chung của lớp: dùng để làm gì, phương thức tạo, lớp cơ sở, nạp chồng toán tử, phương thức đặc biệt, ...