Giới thiệu về lập trình với Python của CS50

OpenCourseWare

Quyên tặng (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan) (https://www.threads.net/@davidjmalan)
(https://twitter.com/davidjmalan)

Bài giảng 9

Vân vân

- Qua nhiều bài học trước đây, chúng ta đã đề cập đến rất nhiều điều liên quan đến Python!

- Trong bài học này, chúng ta sẽ tập trung vào nhiều mục "vân vân" chưa được thảo luận trước đây. "Et cetera" có nghĩa đen là "và phần còn lại"!
- Quả thực, nếu xem tài liệu Python, bạn sẽ tìm thấy khá "phần còn lại" của các tính năng khác.

`set`

- Trong toán học, một tập hợp sẽ được coi là một tập hợp các số không trùng lặp.
- Trong cửa sổ soạn thảo văn bản, mã như sau:

```
students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
    {"name": "Padma", "house": "Ravenclaw"},
]

houses = []
for student in students:
    if student["house"] not in houses:
        houses.append(student["house"])

for house in sorted(houses):
    print(house)
```

  Hãy chú ý cách chúng ta có một danh sách các từ điển, mỗi từ điển đều là một sinh viên. Một danh sách trống được gọi `houses` là được tạo. Chúng tôi lặp lại từng `student` cái trong `students`. Nếu một học sinh `house` không có trong `houses`, chúng tôi sẽ thêm vào danh sách `houses`.
- Hóa ra chúng ta có thể sử dụng `set` các tính năng tích hợp sẵn để loại bỏ sự trùng lặp.
- Trong cửa sổ soạn thảo văn bản, mã như sau:

```
students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
    {"name": "Padma", "house": "Ravenclaw"},
]

houses = set()
for student in students:
    houses.add(student["house"])

for house in sorted(houses):
    print(house)
```

Lưu ý rằng không cần phải kiểm tra để đảm bảo không có sự trùng lặp. Đối `set` tượng sẽ tự động xử lý việc này cho chúng ta.

- Bạn có thể tìm hiểu thêm trong tài liệu của Python về `set` (https://docs.python.org/3/library/stdtypes.html#set).

Biến toàn cục

- Trong các ngôn ngữ lập trình khác, có khái niệm về các biến toàn cục mà bất kỳ hàm nào cũng có thể truy cập được.
- Chúng ta có thể tận dụng khả năng này trong Python. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
balance = 0


def main():
    print("Balance:", balance)


if __name__ == "__main__":
    main()
```

Lưu ý cách chúng ta tạo một biến toàn cục có tên là `balance`, bên ngoài bất kỳ hàm nào.

- Vì không có lỗi nào xảy ra khi thực thi mã ở trên nên bạn cho rằng mọi việc đều ổn. Tuy nhiên, không phải vậy! Trong cửa sổ soạn thảo văn bản, mã như sau:

```
balance = 0


def main():
    print("Balance:", balance)
    deposit(100)
    withdraw(50)
    print("Balance:", balance)


def deposit(n):
    balance += n


def withdraw(n):
    balance -= n


if __name__ == "__main__":
    main()
```

Hãy lưu ý cách hiện tại chúng tôi thêm chức năng thêm và rút tiền đến và đi từ `balance`. Tuy nhiên, khi thực thi mã này, chúng tôi gặp phải một lỗi! Chúng tôi thấy một lỗi có tên `UnboundLocalError`. Bạn có thể đoán được rằng, ít nhất là theo cách chúng ta đã mã hóa hiện tại `balance` cũng như các hàm `deposit` và của chúng ta `withdraw`, chúng ta không thể gán lại cho nó một giá trị giá trị bên trong một hàm.

- Để tương tác với biến toàn cục bên trong hàm, giải pháp là sử dụng từ `global` khóa. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
balance = 0


def main():
    print("Balance:", balance)
    deposit(100)
    withdraw(50)
    print("Balance:", balance)


def deposit(n):
    global balance
    balance += n


def withdraw(n):
    global balance
    balance -= n


if __name__ == "__main__":
    main()
```

Lưu ý cách `global` từ khóa cho biết mỗi hàm `balance` không tham chiếu đến biến cục bộ: thay vào đó, nó đề cập đến biến toàn cục mà chúng ta đặt ban đầu ở đầu mã. Bây giờ, mã của chúng tôi hoạt động!

- Bằng cách tận dụng sức mạnh từ kinh nghiệm lập trình hướng đối tượng, chúng ta có thể sửa đổi mã của mình để sử dụng một lớp thay vì biến toàn cục. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
class Account:
    def __init__(self):
        self._balance = 0

    @property
    def balance(self):
        return self._balance

    def deposit(self, n):
        self._balance += n

    def withdraw(self, n):
```

```
        self._balance -= n


def main():
    account = Account()
    print("Balance:", account.balance)
    account.deposit(100)
    account.withdraw(50)
    print("Balance:", account.balance)


if __name__ == "__main__":
    main()
```

Lưu ý cách chúng tôi sử dụng `account = Account()` để tạo tài khoản. Các lớp cho phép chúng ta giải quyết vấn đề cần một biến toàn cục này một cách rõ ràng hơn vì tất cả các phương thức của lớp này đều có thể truy cập được các biến thể hiện này bằng cách sử dụng `self`.

- Nói chung, các biến toàn cục nên được sử dụng khá tiết kiệm, nếu có!

Hằng số

- Một số ngôn ngữ cho phép bạn tạo các biến không thể thay đổi, được gọi là "hằng số". Các hằng số cho phép một người lập trình một cách phòng thủ và giảm thiểu khả năng các giá trị quan trọng bị thay đổi.
- Trong cửa sổ soạn thảo văn bản, mã như sau:

```
MEOWS = 3

for _ in range(MEOWS):
    print("meow")
```

Thông báo `MEOWS` là hằng số của chúng tôi trong trường hợp này. Các hằng số thường được biểu thị bằng tên biến viết hoa và được đặt ở đầu mã của chúng tôi. Mặc dù điều này *trông* giống như một hằng số nhưng trên thực tế, Python thực sự không có cơ chế ngăn chúng ta thay đổi giá trị đó trong mã của mình! Thay vào đó, bạn đang sử dụng hệ thống danh dự: nếu tên biến được viết hoa toàn bộ, đừng thay đổi nó!

- Người ta có thể tạo một lớp "hằng", bây giờ được đặt trong dấu ngoặc kép vì chúng ta biết Python không hoàn toàn hỗ trợ "hằng". Trong cửa sổ soạn thảo văn bản, mã như sau:

```
class Cat:
    MEOWS = 3

    def meow(self):
        for _ in range(Cat.MEOWS):
            print("meow")
```

```
cat = Cat()
cat.meow()
```

Bởi vì `MEOWS` được định nghĩa bên ngoài bất kỳ phương thức lớp cụ thể nào, tất cả chúng đều có quyền truy cập vào giá trị đó thông qua `Cat.MEOWS`.

Nhập gợi ý

- Trong các ngôn ngữ lập trình khác, người ta thể hiện rõ ràng loại biến bạn muốn sử dụng.
- Như chúng ta đã thấy trước đó trong khóa học, Python không yêu cầu khai báo rõ ràng các kiểu.
- Tuy nhiên, cách tốt nhất là đảm bảo tất cả các biến của bạn đều đúng loại.
- `mypy` là một chương trình có thể giúp bạn kiểm tra để đảm bảo tất cả các biến của bạn đều đúng loại.
- Bạn có thể cài đặt `mypy` bằng cách thực thi trong cửa sổ terminal của mình: `pip install mypy`.

Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def meow(n):
    for _ in range(n):
        print("meow")


number = input("Number: ")
meow(number)
```

Bạn có thể đã thấy nó `number = input("Number: )"` trả về a `string`, không phải a `int`. Nhưng `meow` có thể sẽ muốn một `int`!

- Một gợi ý kiểu có thể được thêm vào để cung cấp cho Python gợi ý về loại biến `meow` sẽ mong đợi. Trong cửa sổ soạn thảo văn bản, mã như sau:

  ```
  def meow(n: int):
      for _ in range(n):
          print("meow")


  number = input("Number: ")
  meow(number)
  ```

  Tuy nhiên, hãy lưu ý rằng chương trình của chúng tôi vẫn báo lỗi.
- Sau khi cài đặt `mypy`, thực hiện `mypy meows.py` trong cửa sổ terminal. `mypy` sẽ cung cấp một số hướng dẫn về cách khắc phục lỗi này.

- Bạn có thể chú thích tất cả các biến của bạn. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def meow(n: int):
    for _ in range(n):
        print("meow")


number: int = input("Number: ")
meow(number)
```

  Lưu ý cách `number` hiện được cung cấp gợi ý loại.

- Một lần nữa, việc thực thi `mypy meows.py` trong cửa sổ terminal cung cấp nhiều phản hồi cụ thể hơn cho bạn, người lập trình.

- Chúng ta có thể sửa lỗi cuối cùng bằng cách mã hóa như sau:

```
def meow(n: int):
    for _ in range(n):
        print("meow")


number: int = int(input("Number: "))
meow(number)
```

  Lưu ý cách chạy `mypy` cách không tạo ra lỗi vì nhập dữ liệu đầu vào của chúng tôi dưới dạng số nguyên.

- Hãy giới thiệu một lỗi mới bằng cách giả sử rằng lỗi đó `meow` sẽ trả về cho chúng ta một chuỗi hoặc `str`. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def meow(n: int):
    for _ in range(n):
        print("meow")


number: int = int(input("Number: "))
meows: str = meow(number)
print(meows)
```

  Lưu ý rằng `meow` chức năng này chỉ có tác dụng phụ. Bởi vì chúng tôi chỉ cố gắng in "meo meo", không trả về giá trị nên sẽ xảy ra lỗi khi chúng tôi cố lưu trữ giá trị trả về của `meow` trong `meows`.

- Chúng ta có thể sử dụng thêm gợi ý loại để kiểm tra lỗi, lần này chú thích các giá trị trả về của hàm. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def meow(n: int) -> None:
    for _ in range(n):
        print("meow")


number: int = int(input("Number: "))
```

```
meows: str = meow(number)
print(meows)
```

Lưu ý cách ký hiệu `-> None` cho biết `mypy` không có giá trị trả về.

- Chúng tôi có thể sửa đổi mã của mình để trả về một chuỗi nếu muốn:

```
def meow(n: int) -> str:
    return "meow\n" * n


number: int = int(input("Number: "))
meows: str = meow(number)
print(meows, end="")
```

Lưu ý cách chúng tôi lưu trữ trong `meows` nhiều `str` s. Chạy `mypy` không tạo ra lỗi.

- Bạn có thể tìm hiểu thêm trong tài liệu Type Hints (https://docs.python.org/3/library/typing.html) của Python .
- Bạn có thể tìm hiểu thêm `mypy` (https://mypy.readthedocs.io/)thông qua tài liệu riêng của chương trình.

Tài liệu

- Một cách tiêu chuẩn để nhận xét mục đích của hàm của bạn là sử dụng chuỗi tài liệu. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def meow(n):
    """Meow n times."""
    return "meow\n" * n


number = int(input("Number: "))
meows = meow(number)
print(meows, end="")
```

Hãy chú ý cách ba dấu ngoặc kép chỉ định chức năng của hàm.

- Bạn có thể sử dụng chuỗi tài liệu để chuẩn hóa cách bạn ghi lại các tính năng của hàm. Trong cửa sổ soạn thảo văn bản, viết mã như sau: s

```
def meow(n):
    """
    Meow n times.

    :param n: Number of times to meow
    :type n: int
    :raise TypeError: If n is not an int
    :return: A string of n meows, one per line
    :rtype: str
    """
```

```
    return "meow\n" * n


number = int(input("Number: "))
meows = meow(number)
print(meows, end="")
```

Lưu ý cách bao gồm nhiều đối số chuỗi tài liệu. Ví dụ, nó mô tả các tham số mà hàm lấy và những gì được hàm trả về.

- Các công cụ đã được thiết lập, chẳng hạn như Sphinx (https://www.sphinx-doc.org/en/master/index.html) , có thể được sử dụng để phân tích chuỗi tài liệu và tự động tạo tài liệu cho chúng tôi dưới dạng trang web và tệp PDF để bạn có thể xuất bản và chia sẻ với người khác.
- Bạn có thể tìm hiểu thêm trong tài liệu về docstrings (https://peps.python.org/pep-0257/) của Python .

`argparse`

- Giả sử chúng ta muốn sử dụng các đối số dòng lệnh trong chương trình của mình. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
import sys

if len(sys.argv) == 1:
    print("meow")
elif len(sys.argv) == 3 and sys.argv[1] == "-n":
    n = int(sys.argv[2])
    for _ in range(n):
        print("meow")
else:
    print("usage: meows.py [-n NUMBER]")
```

Lưu ý cách `sys` được nhập, từ đó chúng ta có quyền truy cập vào `sys.argv` —một mảng các đối số dòng lệnh được cung cấp cho chương trình của chúng ta khi chạy. Chúng ta có thể sử dụng một số `if` câu lệnh để kiểm tra xem việc sử dụng đó có chạy chương trình của chúng ta đúng cách hay không.

- Giả sử rằng chương trình này sẽ phức tạp hơn nhiều. Làm cách nào chúng tôi có thể kiểm tra tất cả các đối số mà người dùng có thể chèn vào? Chúng tôi có thể bỏ cuộc nếu có nhiều hơn một vài đối số dòng lệnh!
- May mắn thay, `argparse` có một thư viện xử lý tất cả việc phân tích cú pháp các chuỗi đối số dòng lệnh phức tạp. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
import argparse

parser = argparse.ArgumentParser()
```

```
parser.add_argument("-n")
args = parser.parse_args()

for _ in range(int(args.n)):
    print("meow")
```

Lưu ý cách `argparse` nhập thay vì `sys`. Một đối tượng được gọi `parser` được tạo ra từ một `ArgumentParser` lớp. Phương thức của lớp đó `add_argument` được sử dụng để cho biết `argparse` những đối số mà chúng tôi mong đợi từ người dùng khi họ chạy chương trình của chúng tôi. Cuối cùng, việc chạy `parse_args` phương thức của trình phân tích cú pháp sẽ đảm bảo rằng tất cả các đối số đã được người dùng đưa vào đúng cách.

- Chúng tôi cũng có thể lập trình rõ ràng hơn, sao cho người dùng của chúng tôi có thể nhận được một số thông tin về cách sử dụng đúng mã của chúng tôi khi họ không sử dụng chương trình một cách chính xác. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
import argparse

parser = argparse.ArgumentParser(description="Meow like a cat")
parser.add_argument("-n", help="number of times to meow")
args = parser.parse_args()

for _ in range(int(args.n)):
    print("meow")
```

Lưu ý cách người dùng được cung cấp một số tài liệu. Cụ thể, một `help` đối số được cung cấp. Bây giờ, nếu người dùng thực thi `python meows.py --help` hoặc `-h`, người dùng sẽ được cung cấp một số manh mối về cách sử dụng chương trình này.

- Chúng tôi có thể cải thiện hơn nữa chương trình này. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
import argparse

parser = argparse.ArgumentParser(description="Meow like a cat")
parser.add_argument("-n", default=1, help="number of times to meow", type=int)
args = parser.parse_args()

for _ in range(args.n):
    print("meow")
```

Lưu ý rằng không chỉ có tài liệu trợ giúp mà bạn còn có thể cung cấp một `default` giá trị khi người dùng không cung cấp đối số.

- Bạn có thể tìm hiểu thêm trong tài liệu của Python về `argparse` (https://docs.python.org/3/library/argparse.html).

Giải nén

- Sẽ không hay sao nếu có thể chia một biến thành hai biến? Trong cửa sổ soạn thảo văn bản, mã như sau:

```
first, _ = input("What's your name? ").split(" ")
print(f"hello, {first}")
```

Lưu ý cách chương trình này cố gắng lấy tên của người dùng bằng cách phân tách một cách ngây thơ trên một khoảng trắng.

- Hóa ra có nhiều cách khác để giải nén các biến. Bạn có thể viết mã mạnh mẽ và tinh tế hơn bằng cách hiểu cách giải nén các biến theo những cách có vẻ nâng cao hơn. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


print(total(100, 50, 25), "Knuts")
```

Lưu ý cách này trả về tổng giá trị của Knuts.

- Điều gì sẽ xảy ra nếu chúng ta muốn lưu trữ tiền của mình trong một danh sách? Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


coins = [100, 50, 25]

print(total(coins[0], coins[1], coins[2]), "Knuts")
```

`coins` Lưu ý cách tạo một danh sách được gọi . Chúng ta có thể chuyển từng giá trị vào bằng cách lập chỉ mục bằng cách sử dụng `0`, `1`, v.v.

- Điều này đang trở nên khá dài dòng. Sẽ thật tuyệt nếu chúng ta có thể chuyển danh sách tiền xu cho hàm của mình phải không?

- Để kích hoạt khả năng chuyển toàn bộ danh sách, chúng ta có thể sử dụng tính năng giải nén. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


coins = [100, 50, 25]

print(total(*coins), "Knuts")
```

Lưu ý cách `*` giải nén chuỗi danh sách tiền xu và chuyển từng phần tử riêng lẻ của nó vào `total`.

- Giả sử chúng ta có thể chuyển tên của loại tiền tệ theo bất kỳ thứ tự nào? Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


print(total(galleons=100, sickles=50, knuts=25), "Knuts")
```

  Lưu ý cách này vẫn tính toán chính xác.

- Khi bạn bắt đầu nói về "tên" và "giá trị", bạn có thể nghĩ đến từ điển! Bạn có thể thực hiện điều này như một từ điển. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


coins = {"galleons": 100, "sickles": 50, "knuts": 25}

print(total(coins["galleons"], coins["sickles"], coins["knuts"]), "Knuts")
```

  Lưu ý cách cung cấp một từ điển được gọi `coins`. Chúng ta có thể lập chỉ mục cho nó bằng cách sử dụng các khóa, chẳng hạn như "galleons" hoặc "liềm".

- Vì `total` hàm cần có ba đối số nên chúng ta không thể chuyển vào từ điển. Chúng ta có thể sử dụng giải nén để trợ giúp việc này. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


coins = {"galleons": 100, "sickles": 50, "knuts": 25}

print(total(**coins), "Knuts")
```

  Lưu ý cách `**` cho phép bạn giải nén một từ điển. Khi giải nén một từ điển, nó sẽ cung cấp cả khóa và giá trị.

`args` Và `kwargs`

- Hãy nhớ lại `print` tài liệu mà chúng ta đã xem trước đó trong khóa học này:

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

- `args` là các đối số vị trí, chẳng hạn như những đối số chúng tôi cung cấp để in như `print("Hello", "World")`.
- `kwargs` là các đối số được đặt tên hoặc "đối số từ khóa", chẳng hạn như những đối số chúng tôi cung cấp để in như `print(end="")`.

- Như chúng ta thấy trong nguyên mẫu của `print` hàm trên, chúng ta có thể yêu cầu hàm của mình mong đợi một đối số vị trí số hiện chưa xác định. Chúng ta cũng có thể yêu cầu nó dự đoán số lượng đối số từ khóa hiện chưa xác định. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def f(*args, **kwargs):
    print("Positional:", args)


f(100, 50, 25)
```

  Lưu ý cách thực thi mã này sẽ được in dưới dạng đối số vị trí.

- Chúng ta thậm chí có thể chuyển vào các đối số được đặt tên. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def f(*args, **kwargs):
    print("Named:", kwargs)


f(galleons=100, sickles=50, knuts=25)
```

  Lưu ý cách các giá trị được đặt tên được cung cấp dưới dạng từ điển.

- Nghĩ về `print` hàm trên, bạn có thể thấy cách `*objects` sử dụng bất kỳ số lượng đối số vị trí nào.
- Bạn có thể tìm hiểu thêm trong tài liệu của Python về `print` (https://docs.python.org/3/library/functions.html#print).

map

- Ngay từ đầu, chúng tôi đã bắt đầu với việc lập trình thủ tục.
- Sau này chúng tôi tiết lộ Python là ngôn ngữ lập trình hướng đối tượng.
- Chúng tôi đã thấy các gợi ý về lập trình hàm, trong đó các hàm có tác dụng phụ mà không có giá trị trả về. Chúng ta có thể minh họa điều này trong cửa sổ soạn thảo văn bản, gõ `code yell.py` và mã như sau:

```
def main():
    yell("This is CS50")


def yell(word):
    print(word.upper())


if __name__ == "__main__":
    main()
```

  Hãy chú ý cách `yell` hàm này được gọi đơn giản.

- Sẽ không hay ho gì nếu hét lên một danh sách các từ không giới hạn phải không? Sửa đổi mã của bạn như sau:

```
def main():
    yell(["This", "is", "CS50"])


def yell(words):
    uppercased = []
    for word in words:
        uppercased.append(word.upper())
    print(*uppercased)


if __name__ == "__main__":
    main()
```

  Lưu ý rằng chúng tôi tích lũy các từ viết hoa, lặp lại từng từ và viết hoa chúng. Danh sách chữ hoa được in bằng cách sử dụng `*` để giải nén nó.

- Loại bỏ dấu ngoặc, chúng ta có thể chuyển các từ làm đối số. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def main():
    yell("This", "is", "CS50")


def yell(*words):
    uppercased = []
    for word in words:
        uppercased.append(word.upper())
    print(*uppercased)


if __name__ == "__main__":
    main()
```

  Lưu ý cách `*words` cho phép hàm lấy nhiều đối số.

- `map` cho phép bạn ánh xạ một hàm tới một chuỗi các giá trị. Trong thực tế, chúng ta có thể mã hóa như sau:

```
def main():
    yell("This", "is", "CS50")


def yell(*words):
    uppercased = map(str.upper, words)
    print(*uppercased)


if __name__ == "__main__":
    main()
```

Lưu ý cách `map` lấy hai đối số. Đầu tiên, nó nhận một hàm mà chúng ta muốn áp dụng cho mọi thành phần của danh sách. Thứ hai, nó lấy chính danh sách đó mà chúng ta sẽ áp dụng hàm nói trên. Do đó, tất cả các từ trong `words` sẽ được chuyển cho `str.upper` hàm và trả về `uppercased`.

- Bạn có thể tìm hiểu thêm trong tài liệu của Python về `map` (https://docs.python.org/3/library/functions.html#map).

Danh sách hiểu List Comprehensions

- Khả năng hiểu danh sách cho phép bạn tạo một danh sách một cách nhanh chóng bằng một lớp lót trang nhã.
- Chúng ta có thể triển khai điều này trong mã của mình như sau:

```
def main():
    yell("This", "is", "CS50")


def yell(*words):
    uppercased = [arg.upper() for arg in words]
    print(*uppercased)


if __name__ == "__main__":
    main()
```

Lưu ý cách thay vì sử dụng `map`, chúng ta viết biểu thức Python trong dấu ngoặc vuông. Đối với mỗi đối số, `.upper` được áp dụng cho nó.

- Đưa khái niệm này đi xa hơn, hãy chuyển sang một chương trình khác.
- Trong cửa sổ soạn thảo văn bản, gõ `code gryffindors.py` và mã như sau:

```
students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
]

gryffindors = []
for student in students:
    if student["house"] == "Gryffindor":
        gryffindors.append(student["name"])

for gryffindor in sorted(gryffindors):
    print(gryffindor)
```

Lưu ý rằng chúng ta có một điều kiện khi tạo danh sách. *Nếu* nhà của học sinh đó là Gryffindor, chúng tôi sẽ thêm học sinh đó vào danh sách tên. Cuối cùng, chúng tôi in tất cả các tên.

- Tinh tế hơn, chúng ta có thể đơn giản hóa mã này bằng cách hiểu danh sách như sau:

```
students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
]

gryffindors = [
    student["name"] for student in students if student["house"] == "Gryffindor"
]

for gryffindor in sorted(gryffindors):
    print(gryffindor)
```

Hãy chú ý cách hiểu danh sách trên một dòng!

`filter`

- Việc sử dụng hàm của Python `filter` cho phép chúng ta trả về một tập hợp con của một chuỗi thỏa mãn một điều kiện nhất định.
- Trong cửa sổ soạn thảo văn bản, mã như sau:

```
students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
]


def is_gryffindor(s):
    return s["house"] == "Gryffindor"


gryffindors = filter(is_gryffindor, students)

for gryffindor in sorted(gryffindors, key=lambda s: s["name"]):
    print(gryffindor["name"])
```

Lưu ý cách tạo một hàm `is_gryffindor` được gọi. Đây là chức năng lọc của chúng tôi sẽ lấy một học sinh `s` và trả về `True` hoặc `False` tùy thuộc vào việc nhà của học sinh đó có phải là nhà Gryffindor hay không. Bạn có thể thấy hàm mới `filter` có hai đối số. Đầu tiên, nó nhận hàm sẽ được áp dụng cho từng phần tử trong một chuỗi—trong trường hợp này là

`is_gryffindor`. Thứ hai, nó lấy trình tự mà nó sẽ áp dụng chức năng lọc—trong trường hợp này là `students`. Trong `gryffindors`, chúng ta chỉ nên thấy những học sinh ở Gryffindor.

- `filter` cũng có thể sử dụng các hàm lambda như sau:

```
students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
]


gryffindors = filter(lambda s: s["house"] == "Gryffindor", students)

for gryffindor in sorted(gryffindors, key=lambda s: s["name"]):
    print(gryffindor["name"])
```

Lưu ý cách cung cấp cùng một danh sách sinh viên.

- Bạn có thể tìm hiểu thêm trong tài liệu của Python về `filter` (https://docs.python.org/3/library/functions.html#filter).

Hiểu từ điển Dictionary Comprehensions

- Chúng ta có thể áp dụng ý tưởng tương tự đằng sau việc hiểu danh sách vào từ điển. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
students = ["Hermione", "Harry", "Ron"]

gryffindors = []

for student in students:
    gryffindors.append({"name": student, "house": "Gryffindor"})

print(gryffindors)
```

Lưu ý cách mã này không (chưa!) sử dụng bất kỳ khả năng hiểu nào. Thay vào đó, nó tuân theo các mô hình tương tự mà chúng ta đã thấy trước đây.

- Bây giờ chúng ta có thể áp dụng khả năng hiểu từ điển bằng cách sửa đổi mã của mình như sau:

```
students = ["Hermione", "Harry", "Ron"]

gryffindors = [{"name": student, "house": "Gryffindor"} for student in students]

print(gryffindors)
```

Lưu ý cách tất cả mã trước được đơn giản hóa thành một dòng duy nhất trong đó cấu trúc của từ điển được cung cấp cho mỗi `student` mã trong `students`.

- Chúng ta thậm chí có thể đơn giản hóa hơn nữa như sau:

```
students = ["Hermione", "Harry", "Ron"]

gryffindors = {student: "Gryffindor" for student in students}

print(gryffindors)
```

Lưu ý cách xây dựng từ điển với các cặp khóa-giá trị.

`enumerate`

- Chúng tôi có thể mong muốn cung cấp một số thứ hạng của mỗi học sinh. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
students = ["Hermione", "Harry", "Ron"]

for i in range(len(students)):
    print(i + 1, students[i])
```

Hãy chú ý cách liệt kê từng học sinh khi chạy mã này.

- Bằng cách sử dụng phép liệt kê, chúng ta có thể làm tương tự:

```
students = ["Hermione", "Harry", "Ron"]

for i, student in enumerate(students):
    print(i + 1, student)
```

Lưu ý cách liệt kê trình bày chỉ mục và giá trị của mỗi `student`.

- Bạn có thể tìm hiểu thêm trong tài liệu của Python về `enumerate` (https://docs.python.org/3/library/functions.html#enumerate).

~~Máy phát điện và máy lặp~~ Generators và Iterators

- Trong Python, có một cách để bảo vệ hệ thống của bạn khỏi bị cạn kiệt tài nguyên khi các vấn đề mà chúng đang giải quyết trở nên quá lớn.
- Ở Hoa Kỳ, người ta có phong tục "đếm cừu" trong đầu khi khó ngủ.
- Trong cửa sổ soạn thảo văn bản, gõ `code sleep.py` và mã như sau:

```
n = int(input("What's n? "))
for i in range(n):
    print("🐑" * i)
```

Hãy chú ý cách chương trình này sẽ đếm số lượng cừu mà bạn yêu cầu.

- Chúng ta có thể làm cho chương trình của mình phức tạp hơn bằng cách thêm một `main` hàm bằng cách mã hóa như sau:

```
def main():
    n = int(input("What's n? "))
    for i in range(n):
        print("🐑" * i)


if __name__ == "__main__":
    main()
```

Lưu ý cách một `main` chức năng được cung cấp.

- Chúng tôi đã có thói quen trừu tượng hóa các phần mã của mình.
- Chúng ta có thể gọi hàm cừu bằng cách sửa đổi mã của mình như sau:

```
def main():
    n = int(input("What's n? "))
    for i in range(n):
        print(sheep(i))


def sheep(n):
    return "🐑" * n


if __name__ == "__main__":
    main()
```

Lưu ý cách `main` hàm thực hiện phép lặp.

- Chúng tôi có thể cung cấp cho `sheep` chức năng nhiều khả năng hơn. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def main():
    n = int(input("What's n? "))
    for s in sheep(n):
        print(s)


def sheep(n):
    flock = []
    for i in range(n):
        flock.append("🐑" * i)
    return flock


if __name__ == "__main__":
    main()
```

Hãy chú ý cách chúng ta tạo một đàn cừu và trả về `flock`.

- Khi thực thi mã của chúng tôi, bạn có thể thử các số lượng cừu khác nhau như `10`, `1000`, và `10000`. Điều gì sẽ xảy ra nếu bạn yêu cầu `1000000` cừu, chương trình của bạn có thể bị treo hoặc bị lỗi hoàn toàn. Vì bạn đã cố gắng tạo ra một danh sách lớn các con cừu nên máy tính của bạn có thể gặp khó khăn trong việc hoàn thành việc tính toán.
- Trình `yield` tạo có thể giải quyết vấn đề này bằng cách trả về một phần nhỏ kết quả mỗi lần. Trong cửa sổ soạn thảo văn bản, mã như sau:

```
def main():
    n = int(input("What's n? "))
    for s in sheep(n):
        print(s)


def sheep(n):
    for i in range(n):
        yield "🐑" * i


if __name__ == "__main__":
    main()
```

Lưu ý cách `yield` chỉ cung cấp một giá trị tại một thời điểm trong khi `for` vòng lặp tiếp tục hoạt động.

- Bạn có thể tìm hiểu thêm trong tài liệu về trình tạo (https://docs.python.org/3/howto/functional.html#generators) của Python .
- Bạn có thể tìm hiểu thêm trong tài liệu về trình vòng lặp (https://docs.python.org/3/howto/functional.html#iterators) của Python .

Chúc mừng!

- Khi thoát khỏi khóa học này, bạn sẽ có thêm mô hình tư duy và hộp công cụ để giải quyết các vấn đề liên quan đến lập trình.
- Đầu tiên, bạn đã tìm hiểu về hàm và biến.
- Thứ hai, bạn đã học về điều kiện.
- Thứ ba, bạn đã học về vòng lặp.
- Thứ tư, bạn đã học về các trường hợp ngoại lệ.
- Thứ năm, bạn đã học về thư viện.
- Thứ sáu, bạn đã học về bài kiểm tra đơn vị.
- Thứ bảy, bạn đã tìm hiểu về file I/O.
- Thứ tám, bạn đã học về biểu thức chính quy.

- Gần đây nhất, bạn đã học về lập trình hướng đối tượng.
- Hôm nay, bạn đã tìm hiểu về nhiều công cụ khác mà bạn có thể sử dụng.

Đây là CS50!

- Cùng nhau tạo một chương trình cuối cùng, nhập `code say.py` vào cửa sổ terminal của bạn và mã như sau:

  ```
  import cowsay
  import pyttsx3

  engine = pyttsx3.init()
  this = input("What's this? ")
  cowsay.cow(this)
  engine.say(this)
  engine.runAndWait()
  ```

  Hãy lưu ý rằng việc chạy chương trình này mang lại cho bạn một sự tiễn biệt đầy tinh thần như thế nào.
- Chúng tôi hy vọng rằng bạn sẽ sử dụng những gì đã học trong khóa học này để giải quyết các vấn đề thực tế trên thế giới, biến thế giới của chúng ta thành một nơi tốt đẹp hơn.
- Đây là CS50!