LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG
What is Object-oriented programming?

Lập trình hướng đối tượng là gì? Hãy tưởng tượng bạn cần mô tả một quả táo cho người chưa từng nhìn thấy nó trước đây, bạn sẽ làm thế nào? Bạn có thể bắt đầu bằng cách nói rằng quả táo là một loại trái cây. Bạn có thể nói về nhiều loại táo khác nhau, mỗi loại có màu sắc, hương vị và tên gọi riêng. Khi giải thích các khái niệm cho máy tính của bạn, hãy tiếp cận theo cách tương tự. Máy tính của bạn không biết quả táo là gì, hoặc trái cây là gì. Nếu bạn muốn máy tính hiểu những điều này, bạn phải mô tả chúng trong chương trình và mã của mình.

Cho đến nay, chúng ta đã thảo luận về các yếu tố **cú pháp như biến, hàm, vòng lặp và một số cấu trúc dữ liệu phức tạp hơn như danh sách và từ điển**. Đây là những công cụ mạnh mẽ trong hộp dụng cụ của chuyên gia IT, nhưng việc dịch các khái niệm thế giới thực, như quả táo hay tài khoản người dùng, thành chương trình vẫn khá khó khăn. Để giúp máy tính dễ dàng hiểu các khái niệm mới này, Python sử dụng một mô hình lập trình được gọi là lập trình hướng đối tượng, **mô hình hóa các khái niệm bằng cách sử dụng lớp và đối tượng**. Đây là một mô hình linh hoạt, mạnh mẽ, trong đó các lớp đại diện và định nghĩa các khái niệm, còn các đối tượng là các phiên bản của lớp.

Trong ví dụ về quả táo, chúng ta có thể có một lớp tên là "táo" định nghĩa đặc điểm của quả táo. Sau đó, chúng ta có thể có nhiều phiên bản của lớp táo, đó là các đối tượng riêng lẻ của lớp đó. Ý tưởng về lập trình hướng đối tượng có thể nghe có vẻ trừu tượng và phức tạp, nhưng bạn đã sử dụng các đối tượng mà không nhận ra. Hầu như mọi thứ trong Python đều là đối tượng, tất cả các số, chuỗi, danh sách và từ điển mà chúng ta đã thấy và sử dụng trong các bài tập và câu đố, đều là đối tượng. Và mỗi đối tượng đều là một thể hiện của một lớp đại diện cho một khái niệm.

Ý tưởng chủ đạo của **lập trình hướng đối tượng** là **thuộc tính và phương thức liên quan đến một kiểu**. Các thuộc tính là các đặc điểm liên quan đến một kiểu, còn các phương thức là các hàm liên quan đến một kiểu (The attributes are the characteristics associated to a type, and the methods are the functions associated to a type). Trong ví dụ về quả táo, các thuộc tính là màu sắc và hương vị. Phương thức là gì? Tùy thuộc vào chúng ta sẽ làm gì với quả táo. Chúng ta có thể có một phương thức cắt chia quả táo thành bốn miếng, hoặc có một phương thức ăn làm giảm lượng táo còn lại sau mỗi lần nhai.

Hãy nghĩ về một ví dụ liên quan hơn đến IT, như một tệp tin trong máy tính của chúng ta. ***Một tệp tin có nhiều thuộc tính, như tên, kích cỡ, ngày tạo, quyền truy cập, nội dung và nhiều hơn nữa***. Thực tế, có rất nhiều thuộc tính tệp tin khác nhau mà Python đã phân thành nhiều lớp để xử lý chúng. Đối tượng tệp tin điển hình tập trung vào nội dung của tệp, vì vậy đối tượng này có một loạt các phương thức để đọc và chỉnh sửa nội dung bên trong tệp.

Hi vọng những ví dụ này giúp làm rõ hơn về lập trình hướng đối tượng, nhưng đừng lo nếu bạn chưa hoàn toàn nắm được. Trong video tiếp theo, chúng ta sẽ tìm hiểu cách áp dụng những khái niệm này với một số lớp và đối tượng mà chúng ta đã sử dụng trong Python, giúp chúng ta đi sâu hơn vào việc hiểu cách thức hoạt động của lập trình hướng đối tượng.

Các thuộc tính là đặc điểm liên quan đến một kiểu, và các phương thức là các hàm liên quan đến một kiểu:

- **Thuộc tính:** Đây là các **đặc điểm, tính chất hoặc giá trị mà một đối tượng thuộc về một kiểu** cụ thể có thể có. Ví dụ, đối với một đối tượng thuộc lớp "Xe hơi", các thuộc tính có thể bao gồm màu sắc, hãng sản xuất, số chỗ ngồi và tốc độ tối đa.
- **Phương thức**: Đây là **các hàm liên quan đến một kiểu cụ thể**, cho phép thực hiện hành động hoặc thao tác dữ liệu của đối tượng. Ví dụ, đối với một đối tượng thuộc lớp "Xe hơi", các phương thức có thể bao gồm chạy xe, dừng xe, đỗ xe và thay đổi tốc độ.

Lập trình hướng đối tượng được định nghĩa: Trong **lập trình hướng đối tượng**, các khái niệm được mô hình hóa dưới dạng lớp và đối tượng. Một ý tưởng được định nghĩa bằng một lớp và một thực thể của lớp này được gọi là đối tượng. Hầu như mọi thứ trong Python đều là đối tượng, bao gồm chuỗi, danh sách, từ điển và số. Khi chúng ta tạo một danh sách trong Python, chúng ta đang tạo một đối tượng là một thực thể của lớp danh sách, đại diện cho khái niệm của danh sách. Các lớp cũng có các thuộc tính và phương thức liên quan đến chúng.

Một ví dụ về lớp và đối tượng trong Python:

Giả sử chúng ta muốn mô tả một hình chữ nhật. Chúng ta có thể tạo một lớp "HinhChuNhat" với các *thuộc tính là chiều dài và chiều rộng*, và các phương thức để tính diện tích và chu vi của hình chữ nhật:

```
class HinhChuNhat:
    def __init__(self, chieu_dai, chieu_rong):
        self.chieu_dai = chieu_dai
        self.chieu_rong = chieu_rong
    def tinh_dien_tich(self):
        return self.chieu_dai * self.chieu_rong
    def tinh_chu_vi(self):
        return 2 * (self.chieu_dai + self.chieu_rong)
```

Ở đây, chúng ta đã tạo một lớp "HinhChuNhat" với các thuộc tính "chieu_dai" và "chieu_rong", cũng như hai phương thức "tinh_dien_tich" và "tinh_chu_vi".

Giờ chúng ta có thể tạo một đối tượng thuộc lớp "HinhChuNhat" và sử dụng các phương thức của nó:

```
hcn1 = HinhChuNhat(4, 5)
dien_tich = hcn1.tinh_dien_tich()
chu_vi = hcn1.tinh_chu_vi()
print(f"Diện tích của hình chữ nhật: {dien_tich}") print(f"Chu vi của hình chữ nhật: {chu_vi}")
```

Trong ví dụ này, **hcn1** là một đối tượng thuộc lớp "HinhChuNhat" với chiều dài là 4 và chiều rộng là 5. Chúng ta sử dụng phương thức "tinh_dien_tich" và "tinh_chu_vi" để tính diện tích và chu vi của hình chữ nhật, rồi in kết quả ra màn hình.

Như vậy, lớp "HinhChuNhat" đại diện cho khái niệm của một hình chữ nhật, với các thuộc tính chiều dài và chiều rộng cũng như các phương thức để tính diện tích và chu vi. Đối tượng "hcn1" là một thực thể của lớp "HinhChuNhat", biểu diễn một hình chữ nhật cụ thể với các giá trị chiều dài và chiều rộng cho trước.

Hy vọng ví dụ này sẽ giúp bạn hiểu rõ hơn về các thuộc tính, phương thức, lớp và đối tượng trong lập trình hướng đối tượng.

```
class HinhChuNhat:
    def __init__(self, chieu_dai, chieu_rong):
        self.chieu_dai = chieu_dai
        self.chieu_rong = chieu_rong
    def tinh_dien_tich(self):
        return self.chieu_dai * self.chieu_rong
    def tinh_chu_vi(self):
        return 2 * (self.chieu_dai + self.chieu_rong)

hcn1 = HinhChuNhat(4, 5)
dien_tich = hcn1.tinh_dien_tich()
chu_vi = hcn1.tinh_chu_vi()

print(f"Diện tích của hình chữ nhật: {dien_tich}")
print(f"Chu vi của hình chữ nhật: {chu_vi}")
```

Kết quả:

Diện tích của hình chữ nhật: 20
Chu vi của hình chữ nhật: 18

Classes and Objects in Detail

Lớp và đối tượng trong Python: Chúng ta đã sử dụng hàm **type** để kiểm tra kiểu của một biến. Khi sử dụng hàm **type**, Python cho chúng ta biết giá trị hoặc biến thuộc về lớp nào. Vì đây là một lớp, nó có một số thuộc tính và phương thức liên quan. Ví dụ với lớp chuỗi (string), thuộc tính duy nhất là nội dung của chuỗi. Còn **về phương thức**, chúng ta đã tìm hiểu một số phương thức của lớp chuỗi như **upper()** để tạo một phiên bản chữ hoa của chuỗi, hay **isnumeric()** để kiểm tra xem nội dung có phải là số hay không.
Mỗi chuỗi mà chúng ta đã sử dụng trong Python cho đến nay là một thực thể khác nhau của lớp chuỗi. Chúng đều có cùng các phương thức, nhưng nội dung khác nhau. Điều này có nghĩa là kết quả của việc gọi các phương thức cũng sẽ khác nhau.
Chúng ta có thể sử dụng hàm **dir** để liệt kê tất cả các thuộc tính và phương thức trong một lớp. Hàm này sẽ in ra một danh sách tất cả các thuộc tính và phương thức. Trong danh sách này, những phương thức đặc biệt có dấu gạch dưới kép ở đầu và cuối. Chúng không được gọi bằng tên kỳ lạ này mà thay vào đó, chúng được gọi bởi một số hàm nội bộ của Python. Ví dụ, **phương thức __len__** được gọi bởi hàm **len** mà chúng ta đã sử dụng trước đây để tìm chiều dài của chuỗi. Hoặc **phương thức __ge__** được sử dụng để so sánh xem một chuỗi có lớn hơn hoặc bằng chuỗi khác khi sử dụng toán tử lớn hơn hoặc bằng.
Sau những phương thức đặc biệt, chúng ta thấy nhiều phương thức chuỗi mà chúng ta đã gặp trước đó. Danh sách này chỉ đưa ra tên của tất cả các phương thức, nhưng không cho chúng ta biết làm thế nào để sử dụng chúng. Để biết cách sử dụng chúng, chúng ta có thể sử dụng hàm **help**.
Khi chúng ta sử dụng hàm **help** trên bất kỳ biến hoặc giá trị nào, chúng ta sẽ hiển thị tất cả tài liệu cho lớp tương ứng. Trong trường hợp này, chúng ta đang xem tài liệu cho lớp **str**, lớp của đối tượng chuỗi. Như trước đây, nó bắt đầu với các phương thức đặc biệt. Nếu chúng ta cuộn xuống, chúng ta sẽ thấy các phương thức mà chúng ta đã biết. Tài liệu cho chúng ta biết các tham số mà phương thức nhận và kiểu giá trị trả về. Nó cũng bao gồm một giải thích về những gì phương thức làm. Đối với phương thức **count**, chúng ta thấy rằng nó nhận chuỗi con sẽ được đếm và có các tham số start và end tùy chọn để chỉ ra

đoạn nào của chuỗi sẽ được xem xét. Chúng ta biết chúng là tùy chọn vì chúng được viết giữa các dấu ngoặc vuông.
Nói chung, **việc đọc và hiểu tài liệu của phương thức rất quan trọng** khi bạn viết mã của riêng mình. Sử dụng các hàm **dir** và **help** giúp bạn dễ dàng tìm hiểu cách sử dụng một thứ gì đó lần đầu tiên. Khi bạn hoàn thành việc xem tài liệu, bạn chỉ cần **gõ q để thoát**.
Python có rất nhiều lớp đã được định nghĩa sẵn cho chúng ta, điều này rất hữu ích. Nhưng sức mạnh của lập trình hướng đối tượng là chúng ta có thể định nghĩa các lớp của riêng mình với các thuộc tính và phương thức của chúng. Mặc dù bạn có thể không cần làm điều này khi viết một đoạn mã đơn giản, nhưng khi chương trình của bạn ngày càng phức tạp, lập trình hướng đối tượng sẽ giúp bạn tận dụng tối đa ngôn ngữ lập trình. Điều này bao gồm cả việc định nghĩa các lớp của riêng bạn. Trong phần tiếp theo, chúng ta sẽ tìm hiểu cách viết định nghĩa lớp của riêng mình với các thuộc tính và phương thức.

Tóm lại Chúng ta có thể sử dụng hàm type() để xác định lớp mà một biến hoặc giá trị thuộc về. Ví dụ, type(" ") cho chúng ta biết đây là một lớp chuỗi. Trong trường hợp này, chỉ có một thuộc tính là giá trị chuỗi, nhưng có rất nhiều phương thức liên quan đến lớp này. Chúng ta đã thấy phương thức upper(), trả về chuỗi viết hoa tất cả các ký tự, cũng như isnumeric() trả về giá trị boolean cho biết liệu chuỗi có phải là một số hay không. Bạn có thể sử dụng hàm dir() để in tất cả các thuộc tính và phương thức của một đối tượng. Mỗi chuỗi là một thực thể của lớp chuỗi, có cùng các phương thức của lớp cha. Vì nội dung của chuỗi khác nhau, các phương thức sẽ trả về các giá trị khác nhau. Bạn cũng có thể sử dụng hàm help() trên một đối tượng, sẽ trả về tài liệu cho lớp tương ứng. Điều này sẽ hiển thị tất cả các phương thức cho lớp, cùng với các tham số mà các phương thức nhận, kiểu giá trị trả về và mô tả về các phương thức.

Defining New Classes

Định nghĩa các lớp mới

Chúng ta đã nói trước đây rằng mục đích của lập trình hướng đối tượng là giúp định nghĩa một khái niệm thế giới thực theo cách mà máy tính hiểu được. Định nghĩa một khái niệm thế giới thực trong mã lập trình có thể khó khăn. Vậy, hãy xem cách chúng ta có thể biểu diễn một khái niệm trong mã Python. Chúng ta sẽ thực hiện từng bước và giữ cho nó đơn giản. Hãy lấy ví dụ về táo của chúng ta trước đây. Chúng ta có thể sử dụng mã này để định nghĩa một lớp Táo cơ bản:

```
Class Apple:
        pass
```

Chắc chắn, nó không giống như nhiều nhưng với hai dòng này, chúng ta đã định nghĩa lớp đầu tiên của mình. Hãy xem cú pháp. Trong Python, chúng ta sử dụng từ khóa dành riêng "class" để thông báo cho máy tính rằng chúng ta đang bắt đầu một lớp mới. Sau đó là tên của lớp và dấu hai chấm. Hướng dẫn về kiểu Python **khuyến khích tên lớp nên bắt đầu bằng một chữ cái hoa**. Vì vậy, chúng ta sẽ sử dụng quy ước đó. Trong trường hợp này, lớp của chúng ta có tên là Apple. Định nghĩa lớp tuân theo cùng một mẫu của các khối khác mà chúng ta đã thấy trước đây như hàm, vòng lặp hoặc nhánh có điều kiện. Sau dòng với định nghĩa lớp là phần thân của lớp, được thụt lề sang phải. Trong trường hợp này, chúng ta chưa thêm bất cứ thứ gì vào phần thân, vì vậy chúng ta sử dụng **từ khóa "pass"** để cho thấy phần thân đang trống. Chúng ta cũng có thể sử dụng từ khóa này như một trình giữ chỗ trong bất kỳ khối Python trống nào.
Vậy chúng ta có thể mở rộng định nghĩa của lớp Táo như thế nào? Có lẽ nó sẽ có các thuộc tính giống nhau đại diện cho lớp Táo.

Định nghĩa các lớp (Tùy chọn)

Chúng ta có thể tạo và định nghĩa các lớp của mình trong Python tương tự như cách chúng ta định nghĩa các hàm. Chúng ta bắt đầu với từ khóa class, theo sau là tên của lớp và dấu hai chấm. Hướng dẫn về kiểu

Python khuyến khích tên lớp nên bắt đầu bằng một chữ cái hoa. Sau dòng định nghĩa lớp là phần thân của lớp, được thụt lề sang phải. Bên trong phần thân của lớp, chúng ta có thể định nghĩa các thuộc tính cho lớp.
Hãy xem ví dụ về lớp Apple của chúng ta:

```
class Apple:
        color = ""
        flavor = ""
```

Chúng ta có thể tạo một thể hiện mới của lớp mới bằng cách gán nó cho một biến. Điều này được thực hiện bằng cách gọi tên lớp như thể nó là một hàm. Chúng ta có thể đặt các thuộc tính của thể hiện lớp bằng cách truy cập chúng thông qua ký hiệu dấu chấm. Ký hiệu dấu chấm có thể được sử dụng để đặt hoặc truy xuất các thuộc tính đối tượng, cũng như gọi các phương thức liên quan đến lớp.

```
jonagold = Apple()
jonagold.color = "red"
jonagold.flavor = "sweet"
```

Chúng ta đã tạo một thể hiện Apple có tên là jonagold và đặt các thuộc tính màu và hương vị cho đối tượng Apple này. Chúng ta có thể tạo một thể hiện khác của Apple và đặt các thuộc tính khác nhau để phân biệt giữa hai giống táo khác nhau.

```
golden = Apple()
golden.color = "Yellow"
golden.flavor = "Soft"
```

Bây giờ chúng ta có một đối tượng Apple khác có tên là golden cũng có các thuộc tính màu và hương vị. Nhưng các thuộc tính này có các giá trị khác nhau.

Bài tập

Question 1: Yêu cầu: Sử dụng dot notation để truy cập các phương thức và thuộc tính của đối tượng. Cho một lớp tên là Birds với hai thuộc tính: color và number. Birds có một phương thức tên là count() để đếm số chim (thêm giá trị vào number). Hãy chọn đoạn mã đúng để in ra số chim. Đáp án: c. bluejay.count() print(bluejay.number)

Question 2: Yêu cầu: Điền vào các chỗ trống trong đoạn mã để hoàn thành chức năng trao đổi quả táo và ý tưởng giữa hai người. Khi trao đổi quả táo, số lượng táo của mỗi người sẽ hoán đổi cho nhau. Khi trao đổi ý tưởng, số lượng ý tưởng của mỗi người sẽ được cộng dồn.

Question 3: Yêu cầu: Điền vào các chỗ trống trong hàm max_elevation_city để trả về tên thành phố và quốc gia (cách nhau bằng dấu phẩy) khi so sánh 3 đối tượng City đã tạo cho một dân số tối thiểu được chỉ định. Ví dụ: khi gọi hàm với dân số tối thiểu 1 triệu, kết quả sẽ là "Sofia, Bulgaria".

Question 4: Yêu cầu: Điểm khác biệt giữa đối tượng và lớp là gì? Đáp án: b. An object is a specific instance of a class

Question 5: Yêu cầu: Chúng ta có hai đồ nội thất: một cái bàn gỗ màu nâu và một chiếc ghế sofa da màu đỏ. Điền vào các chỗ trống sau khi tạo ra từng đối tượng của lớp Furniture, để hàm describe_furniture có thể định dạng một câu mô tả các đồ vật theo cú pháp sau: "This piece of furniture is made of {color} {material}".

Đáp án

Dưới đây là các bài tập và đáp án tương ứng:

Question 1: Đáp án: c. bluejay.count() print(bluejay.number)

Question 2: Đáp án:

```
class Person:
   apples = 0
   ideas = 0
```

```
johanna = Person()
johanna.apples = 1
johanna.ideas = 1

martin = Person()
martin.apples = 2
martin.ideas = 1
        def exchange_apples(you, me):
                temp = you.apples
                you.apples = me.apples
                me.apples = temp
                return you.apples, me.apples
        def exchange_ideas(you, me):
                you.ideas = you
                me.ideas = you.ideas + me.ideas, you.ideas + me.ideas
                return you.ideas, me.ideas
```

Question 3: Đáp án:

```
"# define a basic city class
class City:
        name = ""
        country = ""
        elevation = 0
        population = 0

        # create a new instance of the City class and
        # define each attribute
        city1 = City()
        city1.name = "Cusco"
        city1.country = "Peru"
        city1.elevation = 3399
        city1.population = 358052

        # create a new instance of the City class and
        # define each attribute
        city2 = City()
        city2.name = "Sofia"
        city2.country = "Bulgaria"
        city2.elevation = 2290
        city2.population = 1241675

        # create a new instance of the City class and
        # define each attribute
        city3 = City()
        city3.name = "Seoul"
        city3.country = "South Korea"
        city3.elevation = 38
        city3.population = 9733509
        def max_elevation_city(min_population):
```

```
                return_city = City()
                if city1.population >= min_population and city1.elevation >
                        return_city.elevation: return_city = city1
                if city2.population >= min_population and city2.elevation >
                        return_city.elevation: return_city = city2
                if city3.population >= min_population and city3.elevation >
                         return_city.elevation: return_city = city3
                if return_city.name:
                        return "{}, {}".format(return_city.name, return_city.country) else: return ""
```

Question 4: Đáp án: b. An object is a specific instance of a class

Question 5: Đáp án:

```
class Furniture:
        color = ""
        material = ""
        table = Furniture()
        table.color = "brown"
         table.material = "wood"
        couch = Furniture()
        couch.color = "red"
        couch.material = "leather"
```

Khi chạy đoạn mã này, bạn sẽ nhận được kết quả sau:

```
This piece of furniture is made of brown wood
This piece of furniture is made of red leather
```

Instance Methods (Optional)

Phần này giới thiệu về Instance Methods và giải thích các khái niệm liên quan. Dưới đây là phân tích và giải thích chi tiết:

1. Instance Methods: Là các phương thức (hàm) được định nghĩa bên trong một class và hoạt động trên các thuộc tính của một instance cụ thể. Khi gọi một phương thức trên một đối tượng, chúng ta sẽ thực thi các hàm hoạt động trên các thuộc tính của instance đó.
2. Định nghĩa phương thức trong class: Chúng ta có thể tạo các phương thức bên trong class bằng cách tạo các hàm trong phần định nghĩa class. Các phương thức này có thể nhận một tham số self, đại diện cho instance mà phương thức đang được thực thi. Điều này cho phép truy cập các thuộc tính của instance thông qua dot notation, ví dụ như self.name, sẽ truy cập thuộc tính name của instance cụ thể của đối tượng class.
3. Biến instance: Khi có các biến chứa các giá trị khác nhau đối với các instance khác nhau của cùng một class, chúng được gọi là biến instance.

Phần này cũng đề cập đến việc sử dụng phương thức có trả về giá trị. Chẳng hạn, chúng ta có thể tạo một phương thức chuyển đổi tuổi của một con lợn sang tuổi lợn. Khi giá trị của thuộc tính thay đổi, giá trị trả về của phương thức cũng thay đổi theo.

Ngoài ra, phần này cũng giới thiệu về một số loại phương thức đặc biệt, bao gồm phương thức constructor, mà chúng ta sẽ tìm hiểu trong phần tiếp theo.

1. **Instance Methods:** Đây là các phương thức (hàm) được định nghĩa bên trong một class và hoạt động trên các thuộc tính của một instance cụ thể. Khi bạn gọi một phương thức trên một đối tượng, chúng ta thực thi các hàm hoạt động trên các thuộc tính của instance đó. Phương thức này thường sử dụng từ khóa **self** để tham chiếu đến các thuộc tính hoặc phương thức khác của cùng một instance.

Ví dụ về một class **Dog** với một instance method:

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        print(f"{self.name} says Woof!")

dog1 = Dog("Max", 3)
dog1.bark()  # Output: Max says Woof!
```

Trong ví dụ trên, chúng ta đã tạo một class **Dog** với một phương thức **bark()**. Khi chúng ta gọi phương thức **bark()** trên một instance của class **Dog**, phương thức sẽ in ra tên của chú chó đó kèm theo "says Woof!".

2. **Biến instance**: Đây là các biến có giá trị khác nhau cho các instance khác nhau của cùng một class. Biến instance được khởi tạo bên trong phương thức **__init__()** và được truy cập thông qua tham chiếu **self**.

Trong ví dụ về class **Dog**, **name** và **age** là các biến instance, vì chúng có giá trị riêng biệt cho mỗi instance của class **Dog**.

```
dog2 = Dog("Buddy", 5)
print(dog1.name)  # Output: Max
print(dog2.name)  # Output: Buddy
```

Như bạn thấy, mỗi instance **dog1** và **dog2** của class **Dog** có các giá trị khác nhau cho thuộc tính **name**.

Constructors and Special Methods

Trong phần này, chúng ta sẽ tìm hiểu về các phương thức đặc biệt trong Python như Constructors và một số phương thức khác.

Constructors Để khởi tạo giá trị cho thuộc tính của một đối tượng khi tạo nó, chúng ta sử dụng phương thức đặc biệt gọi là constructor. Constructor là phương thức được gọi khi chúng ta khởi tạo một đối tượng từ một lớp (class). Constructor luôn có tên là **__init__**.

Ví dụ về lớp Apple có constructor như sau:

```
class Apple:
    def __init__(self, color, flavor):
        self.color = color
        self.flavor = flavor
```

Khi bạn khởi tạo một đối tượng từ lớp Apple, constructor sẽ được gọi và giá trị màu sắc (color) và hương vị (flavor) sẽ được thiết lập ngay từ đầu:

```
jonagold = Apple("red", "sweet")
print(jonagold.color)  # Output: Red
```

Phương thức đặc biệt __str__ Phương thức đặc biệt **__str__** cho phép chúng ta định nghĩa cách một đối tượng sẽ được in ra khi truyền vào hàm **print()**. Nếu một đối tượng không có phương thức đặc biệt này được định nghĩa, Python sẽ sử dụng biểu diễn mặc định, và in ra vị trí của đối tượng trong bộ nhớ, không hữu ích cho lắm.

Dưới đây là lớp Apple với phương thức đặc biệt **__str__** đã được thêm vào:

```
class Apple:
    def __init__(self, color, flavor):
        self.color = color
        self.flavor = flavor

    def __str__(self):
        return "This apple is {} and its flavor is {}".format(self.color, self.flavor)
```

Giờ đây, khi chúng ta truyền một đối tượng Apple vào hàm **print()**, chúng ta sẽ nhận được một chuỗi định dạng đẹp:

```
jonagold = Apple("red", "sweet")
print(jonagold)  # Output: This apple is red and its flavor is sweet
```

Khi tạo ra các lớp và đối tượng, nên suy nghĩ về cách lớp của bạn có thể được sử dụng và định nghĩa phương thức **__str__** khi tạo các đối tượng mà bạn muốn in ra sau này.

I. Tổng quan về Documenting Functions, Classes, and Methods (Viết tài liệu cho Hàm, Lớp và Phương thức)

Thế giới của các lớp (*classes*) và phương thức (*methods*) có thể hơi khó hiểu khi bạn vẫn đang tìm hiểu. Đó là lý do tại sao hàm trợ giúp (*help function*) có thể rất hữu ích. Bạn có thể nhớ rằng chúng ta vẫn có thể sử dụng hàm trợ giúp Python (*Python help function*) để tìm hiểu tài liệu về các lớp và phương thức. Chúng ta cũng có thể làm điều này với các lớp, phương thức và hàm của chúng ta.

II. Sử dụng hàm trợ giúp với lớp (Using the Help Function with a Class)

Chúng ta sẽ bắt đầu với lớp Apple (*Apple class*) mà chúng ta đã sử dụng trước đây và sau đó chúng ta sẽ yêu cầu sự trợ giúp. Khi chúng ta yêu cầu trợ giúp với lớp của chúng ta, chúng ta sẽ nhận được danh sách các phương thức được định nghĩa trong lớp. Trong ví dụ này, các phương thức đã định nghĩa là

hàm tạo (*constructor*) và chuyển đổi thành chuỗi (*conversion to string*). Nhưng tài liệu này rất ngắn gọn và thực sự không giải thích nhiều.

III. Cách tạo Docstring (Creating a Docstring)

Chúng ta muốn các phương thức, lớp và hàm của chúng ta cung cấp thông tin chi tiết hơn khi chúng ta hoặc người khác sử dụng hàm trợ giúp. Chúng ta có thể làm điều đó bằng cách thêm một docstring. Một **docstring** là một đoạn văn bản ngắn gọn giải thích về công dụng của một thứ gì đó.

IV. Ví dụ về cách tạo và sử dụng Docstring (Example of Creating and Using a Docstring)

Ví dụ về cách tạo một hàm đơn giản với docstring:

pythonCopy code

def to_seconds(hours, minutes, seconds): """Returns the amount of seconds in the given hours, minutes, and seconds.""" return hours*3600 + minutes*60 + seconds

Trong ví dụ này, hàm **to_seconds** có một docstring ngay sau dòng định nghĩa hàm. Đoạn văn này giải thích công dụng của hàm: chuyển đổi thời gian từ giờ, phút và giây thành số giây.

Chúng ta có thể sử dụng hàm trợ giúp để xem docstring của hàm này bằng lệnh: **help(to_seconds)**.

V. Thêm Docstrings vào lớp và phương thức (Adding Docstrings to Classes and Methods)

Chúng ta cũng có thể thêm docstrings vào các lớp và phương thức. Điều này giúp chúng ta hiểu rõ hơn về chức năng của các lớp và phương thức này.

VI. Kết luận (Conclusion)

Docstrings rất hữu ích để tìm hiểu cách sử dụng một hàm mà bạn chưa từng sử dụng trước đây. Nếu bạn đang đọc một đoạn mã do người khác viết, docstrings cho phép chúng ta hiểu mã nguồn tốt hơn vì các lớp, phương thức và hàm được tài liệu hóa rõ ràng. Do đó, khi viết mã của bạn, hãy thêm docstrings để giải thích các hàm, lớp và phương thức của bạn. Điều này sẽ tạo ra sự khác biệt lớn cho bất kỳ ai có thể sử dụng mã của bạn.

Documenting with Docstrings

1. **Định nghĩa lớp Apple:**

```
class Apple:
    def __init__(self, color, flavor):
        self.color = color
        self.flavor = flavor

    def __str__(self):
        return "This apple is {} and its flavor is {}".format(self.color, self.flavor)
```

- Trước tiên, ta định nghĩa một lớp **Apple** với hai phương thức (methods): **__init__** và **__str__**.
- **__init__** là một phương thức đặc biệt, được gọi là constructor, dùng để khởi tạo một đối tượng mới từ lớp này. Constructor nhận vào hai tham số: **color** và **flavor**, và gán chúng vào các thuộc tính tương ứng của đối tượng (**self.color** và **self.flavor**).
- **__str__** cũng là một phương thức đặc biệt, được sử dụng khi cần biểu diễn đối tượng dưới dạng chuỗi (string).

2. **Sử dụng hàm help để xem thông tin về lớp Apple**:

```
help(Apple)
```

Hàm **help** sẽ trả về thông tin về các phương thức và thuộc tính của lớp. Tuy nhiên, ở đây, thông tin trả về khá ngắn và không giải thích rõ ràng về mục đích của các phương thức.

3. **Định nghĩa hàm to_seconds với docstring:**

```
def to_seconds(hours, minutes, seconds):
```

```
    """Returns the amount of seconds in the given hours, minutes and seconds."""
    return hours*3600+minutes*60+seconds
```
Ở đây, ta định nghĩa một hàm **to_seconds**, với một docstring đặt ngay sau dòng định nghĩa hàm. Docstring là một chuỗi văn bản ngắn, được bao quanh bởi ba dấu ngoặc kép, mô tả công việc mà hàm, phương thức hoặc lớp thực hiện.

4. **Sử dụng hàm help để xem thông tin về hàm to_seconds:**

```
help(to_seconds)
```

- Khi ta sử dụng hàm **help** trên hàm **to_seconds**, docstring mà ta đã định nghĩa sẽ được hiển thị, giúp ta hiểu rõ hơn về công việc mà hàm thực hiện.

Như vậy, việc sử dụng docstrings giúp chúng ta rõ ràng hóa mục đích của các hàm, phương thức và lớp trong Python, đặc biệt hữu ích khi ta làm việc với thư viện hoặc mã nguồn do người khác viết.

Classes and Methods Cheat Sheet (Optional)

Phần này cung cấp một tổng quan về cách định nghĩa và sử dụng lớp (classes) và phương thức (methods) trong Python.

1. **Định nghĩa lớp và phương thức**:

```
class ClassName:
    def method_name(self, other_parameters):
        body_of_method
```

- Ta bắt đầu bằng việc sử dụng từ khóa **class** để tạo một lớp mới với tên **ClassName**.
- Trong lớp, ta có thể định nghĩa các phương thức, bắt đầu bằng từ khóa **def**. **self** trong danh sách tham số là một tham chiếu đến đối tượng hiện tại, và **other_parameters** là các tham số khác mà phương thức có thể nhận.

2. **Lớp và Thể hiện (Classes and Instances)**:
   - Lớp định nghĩa hành vi của tất cả các thể hiện (instances) thuộc lớp đó.
   - Mỗi biến thuộc một lớp cụ thể đều là một thể hiện hoặc đối tượng (object).
   - Đối tượng có thể có các thuộc tính (attributes), lưu trữ thông tin về đối tượng.
   - Ta có thể khiến đối tượng thực hiện công việc bằng cách gọi các phương thức của nó.
   - Tham số đầu tiên của các phương thức (**self**) đại diện cho đối tượng hiện tại.
   - Phương thức giống như hàm, nhưng chúng chỉ có thể được sử dụng thông qua một lớp.
3. **Phương thức đặc biệt (Special methods)**:
   - Phương thức đặc biệt bắt đầu và kết thúc với **__**.
   - Các phương thức đặc biệt có tên cụ thể, như **__init__** cho constructor hoặc **__str__** cho việc chuyển đổi sang chuỗi.
4. **Tài liệu hóa lớp, phương thức và hàm (Documenting classes, methods, and functions)**:

```
class ClassName:
    """Documentation for the class."""
    def method_name(self, other_parameters):
        """Documentation for the method."""
```

```
    body_of_method

def function_name(parameters):
    """Documentation for the function."""
    body_of_function
```

- Ta có thể thêm tài liệu cho lớp, phương thức, và hàm bằng cách sử dụng docstrings ngay sau dòng định nghĩa. Docstrings là một chuỗi văn bản ngắn, mô tả công việc mà hàm, phương thức, hoặc lớp thực hiện.

About Jupyter Notebooks (Optional)

Đây là một giới thiệu về Jupyter Notebooks, một công cụ mạnh mẽ để viết và chạy code Python, cũng như hiển thị kết quả và giải thích bằng văn bản, hình ảnh, và nhiều hơn nữa. Jupyter Notebook là một công nghệ mã nguồn mở mà bạn có thể cài đặt và chạy trên máy tính của mình. Tuy nhiên, bạn cũng có thể chạy Jupyter Notebook trong trình duyệt web mà không cần cài đặt bất kỳ thứ gì.

1. **Mở Jupyter Notebook**: Khi bạn mở một Jupyter Notebook, bạn sẽ thấy một số văn bản giải thích và một số đoạn mã.
2. **Chạy Mã**: Bạn có thể chạy từng cell mã (một khối đơn lẻ của mã) bằng cách nhấn nút "run" trên thanh công cụ hoặc nhấn Shift-Enter trên bàn phím. Khi một cell mã đã được chạy, số lượng sẽ xuất hiện bên cạnh cell đó.
3. **Hiển thị Kết quả**: Nếu cell mã tạo ra bất kỳ đầu ra nào (như in ra một giá trị), đầu ra đó sẽ xuất hiện ngay dưới cell.
4. **Sửa Đổi và Chạy Lại Mã**: Nếu bạn thay đổi mã trong một cell, bạn cần phải chạy lại cell đó để áp dụng thay đổi. Điều này đặc biệt quan trọng nếu cell chứa định nghĩa lớp hoặc hàm, vì thay đổi định nghĩa nhưng không chạy lại cell sẽ không ảnh hưởng đến bất kỳ mã nào khác đang sử dụng lớp hoặc hàm đó.

Jupyter Notebook là một công cụ rất mạnh mẽ để học Python vì nó cho phép bạn thử nghiệm và thấy kết quả ngay lập tức, cũng như tạo ra tài liệu đẹp và dễ đọc với mã, kết quả, và giải thích.

Help with Jupyter Notebooks (Optional)

Help with Jupyter Notebooks

We've aimed to make our Jupyter notebooks easy to use. But, if you get stuck, you can find more information here.

If you still need help, the discussion forums are a great place to find it! Use the forums to ask questions and source answers from your fellow learners.

If you want to learn more about Jupyter Notebooks as a technology, check out these resources:

- Jupyter Notebook Tutorial, by datacamp.com
- How to use Jupyter Notebooks, by codeacademy.com
- Teaching and Learning with Jupyter, by university professors using Jupyter

Practice Notebook: Methods and Classes

Đoạn mã trên đang yêu cầu bạn tạo một lớp (class) Python tên là "Elevator" với một số phương thức (methods). Dưới đây là một giải thích từng phần của yêu cầu:

1. **Định nghĩa lớp Elevator**: Đầu tiên, bạn được yêu cầu tạo một lớp tên là Elevator với ba thuộc tính: bottom (tầng dưới cùng), top (tầng trên cùng), và current (tầng hiện tại).

2. **Phương thức up**: Phương thức này giúp thang máy di chuyển lên một tầng. Tuy nhiên, nếu thang máy đang ở tầng trên cùng, nó không thể di chuyển lên.
3. **Phương thức down**: Tương tự như phương thức up, nhưng giúp thang máy di chuyển xuống một tầng. Nếu thang máy đang ở tầng dưới cùng, nó không thể di chuyển xuống.
4. **Phương thức go_to**: Phương thức này giúp thang máy di chuyển đến một tầng cụ thể. Tuy nhiên, nếu tầng yêu cầu không nằm trong phạm vi tầng dưới cùng và tầng trên cùng, thang máy không thể di chuyển đến tầng đó.
5. **Kiểm tra**: Bạn cần kiểm tra xem phương thức up, down và go_to có hoạt động đúng không bằng cách gọi chúng và in ra tầng hiện tại sau khi thực hiện mỗi phương thức.
6. **Thêm phương thức str**: Cuối cùng, bạn cần thêm một phương thức **str** để khi in ra thang máy, bạn sẽ nhận được thông báo về tầng hiện tại, chẳng hạn "Current floor: 5".

Hãy nhớ rằng, trong Python, "**init**" và "**str**" là các phương thức đặc biệt. "**init**" được gọi khi tạo một thực thể mới của lớp, trong khi "**str**" được gọi khi bạn cố gắng in ra thực thể đó dưới dạng một chuỗi.