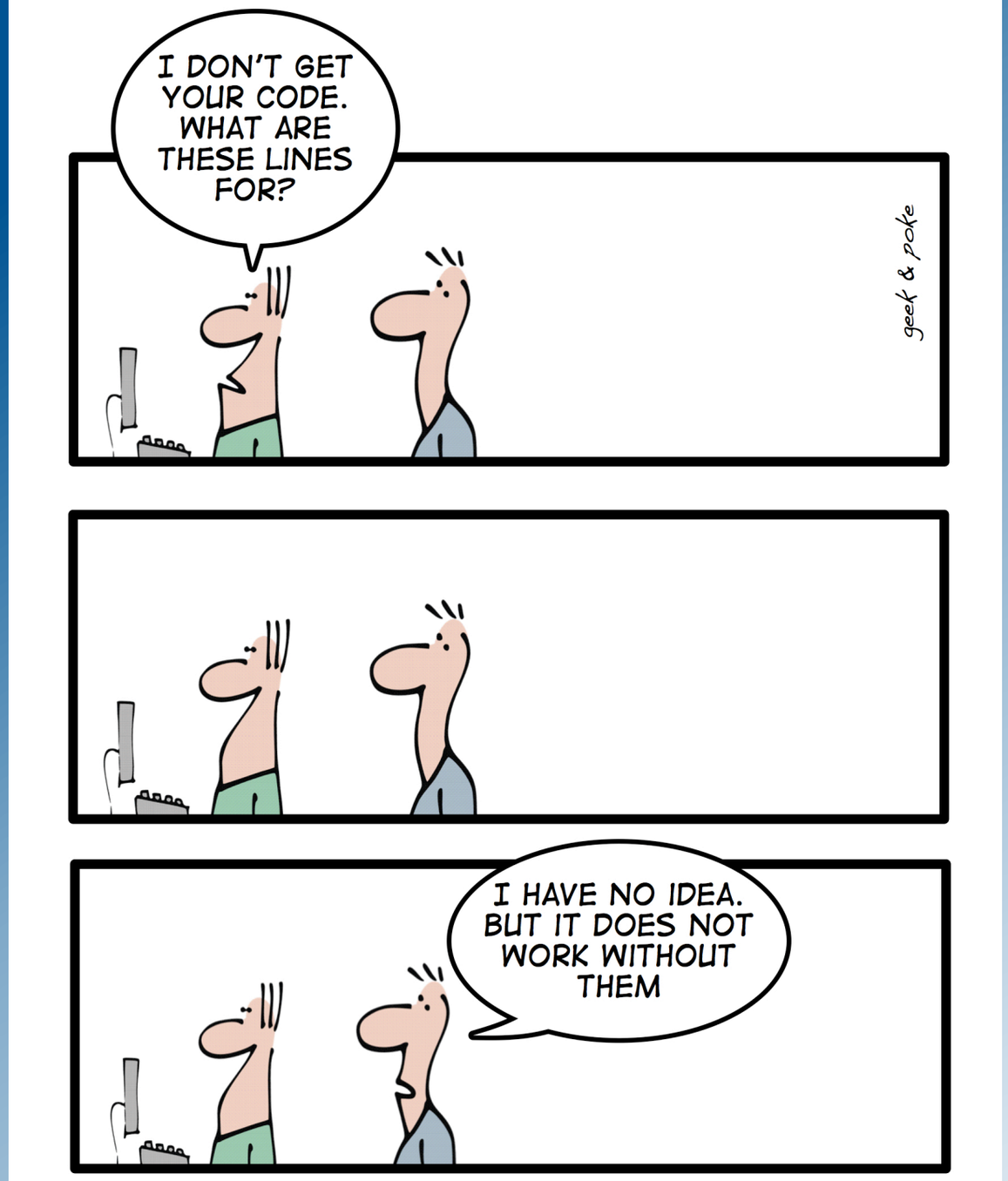


Lecture 5: Object-Oriented Programming

September 11, 2024

Objects

Create classes to define objects
Write methods and create attributes for objects
Instantiate objects from classes
Restrict access to an object's attributes

“Neopets”

Objects-Oriented Basics

- To build an object, you need a blueprint, or a class
- Classes include methods: things that an object can do

Neopets Python Program

```
# Simple Neopet
# Demonstrates a basic class and object

class Neopet:
    """A virtual pet"""
    def talk(self):
        print("Hi. I'm an instance of class Neopet.")

# main
pet = Neopet()
pet.talk()

input("\n\nPress the enter key to exit.")
```

Using constructors

```
# Constructor Neopet
# Demonstrates constructors

class Neopet(object):
    """A virtual pet"""
    def __init__(self):
        print("A new neopet has been born!")
    def talk(self):
        print("Hi. I'm an instance of class Neopet.")

# main
pet1 = Neopet()
pet2 = Neopet()
pet1.talk()
pet2.talk()

input("\n\nPress the enter key to exit.")
```

Using constructors

```
1  # Constructor Neopet
2  # Demonstrates constructors
3
4  class Neopet(object):
5      """A virtual pet"""
6      def __init__(self):
7          print("A new neopet has been born!")
8      def talk(self):
9          print("Hi. I'm an instance of class Neopet.")
10
11 # main
12 pet1 = Neopet()
13 pet2 = Neopet()
14 pet1.talk()
15 pet2.talk()
16
17 input("\n\nPress the enter key to exit.")
```

Using constructors

```
1  # Constructor Neopet
2  # Demonstrates constructors
3
4  class Neopet(object):
5      """A virtual pet"""
6      def __init__(self):
7          print("A new neopet has been born!")
8      def talk(self):
9          print("Hi. I'm an instance of class Neopet.")
10
11 # main
12 pet1 = Neopet()
13 pet2 = Neopet()
14 pet1.talk()
15 pet2.talk()
16
17 input("\n\nPress the enter key to exit.")
```

Using constructors

```
1  # Constructor Neopet
2  # Demonstrates constructors
3
4  class Neopet(object):
5      """A virtual pet"""
6      def __init__(self):
7          print("A new neopet has been born!")
8      def talk(self):
9          print("Hi. I'm an instance of class Neopet.")
10
11 # main
12 pet1 = Neopet()
13 pet2 = Neopet()
14 pet1.talk()
15 pet2.talk()
16
17 input("\n\nPress the enter key to exit.")
```

Using attributes

```
# Attribute Neopet
# Demonstrates creating and accessing object attributes

class Neopet(object):
    """A virtual pet"""
    def __init__(self, name):
        print("A new neopet has been born!")
        self.name = name

    def __str__(self):
        rep = "Neopet object \n"
        rep += "name: "+self.name + "\n"
        return rep

    def talk(self):
        print("Hi. I'm", self.name, "\n")

# main
pet1 = Neopet("Flotsam")
pet1.talk()

pet2 = Neopet("Jetsom")
pet2.talk()

print("Printing pet1:")
print(pet1)

print("Directly accessing pet1.name:")
print(pet1.name)

input("\n\nPress the enter key to exit.")

```

Using attributes

```
# Attribute Neopet
# Demonstrates creating and accessing object attributes

class Neopet(object):
    """A virtual pet"""
    def __init__(self, name):
        print("A new neopet has been born!")
        self.name = name

    def __str__(self):
        rep = "Neopet object \n"
        rep += "name: "+self.name + "\n"
        return rep

    def talk(self):
        print("Hi. I'm", self.name, "\n")

# main
pet1 = Neopet("Flotsam")
pet1.talk()

pet2 = Neopet("Jetsom")
pet2.talk()

print("Printing pet1:")
print(pet1)

print("Directly accessing pet1.name:")
print(pet1.name)

input("\n\nPress the enter key to exit.")

```

Accessing Attributes

```
# Attribute Neopet
# Demonstrates creating and accessing object attributes

class Neopet(object):
    """A virtual pet"""
    def __init__(self, name):
        print("A new neopet has been born!")
        self.name = name

    def __str__(self):
        rep = "Neopet object \n"
        rep += "name: "+self.name + "\n"
        return rep

    def talk(self):
        print("Hi. I'm", self.name, "\n")

# main
pet1 = Neopet("Flotsam")
pet1.talk()

pet2 = Neopet("Jetsom")
pet2.talk()

print("Printing pet1:")
print(pet1)

print("Directly accessing pet1.name:")
print(pet1.name)

input("\n\nPress the enter key to exit.")

```

Printing an Object

```
# Attribute Neopet
# Demonstrates creating and accessing object attributes

class Neopet(object):
    """A virtual pet"""
    def __init__(self, name):
        print("A new neopet has been born!")
        self.name = name

    def __str__(self):
        rep = "Neopet object \n"
        rep += "name: "+self.name + "\n"
        return rep

    def talk(self):
        print("Hi. I'm", self.name, "\n")

# main
pet1 = Neopet("Flotsam")
pet1.talk()

pet2 = Neopet("Jetsom")
pet2.talk()

print("Printing pet1:")
print(pet1)

print("Directly accessing pet1.name:")
print(pet1.name)

input("\n\nPress the enter key to exit.")

```

Class Attributes and Static Methods

- To build an object, you need a blueprint, or a class
- Class attribute is a post-it note stuck to the blueprint

```
# Classy Neopet
# Demonstrates class attributes and static methods

class Neopet(object):
    """A virtual pet"""
    total = 0

    def __init__(self, name):
        print("A new neopet has been born!")
        self.name = name
        Neopet.total += 1

# main
print("Accessing the class attribute Neopet.total:")
print(Neopet.total)

input("\n\nPress the enter key to exit.")
```

```
# Classy Neopet
# Demonstrates class attributes and static methods

class Neopet(object):
    """A virtual pet"""
    total = 0

    def status():
        print("\nThe total number of neopets is", Neopet.total)

    status = staticmethod(status)

    def __init__(self, name):
        print("A new neopet has been born!")
        self.name = name
        Neopet.total += 1

# main
print("Accessing the class attribute Neopet.total:")
print(Neopet.total)

print("\nCreating neopets.")
pet1 = Neopet("pet 1")
pet2 = Neopet("pet 2")
pet3 = Neopet("pet 3")

Neopet.status()

print("\nAccessing the class attribute through an object:")
print(pet1.total)


input("\n\nPress the enter key to exit.")
```

```
# Classy Neopet
# Demonstrates class attributes and static methods

class Neopet(object):
    """A virtual pet"""
    total = 0

    def status():
        print("\nThe total number of neopets is", Neopet.total)

    status = staticmethod(status)

    def __init__(self, name):
        print("A new neopet has been born!")
        self.name = name
        Neopet.total += 1

# main
print("Accessing the class attribute Neopet.total:")
print(Neopet.total)

print("\nCreating neopets.")
pet1 = Neopet("pet 1")
pet2 = Neopet("pet 2")
pet3 = Neopet("pet 3")

Neopet.status()

print("\nAccessing the class attribute through an object:")
print(pet1.total)


input("\n\nPress the enter key to exit.")
```

Object Encapsulation

- Functions are encapsulated and hide the details of their inner workings from the part of the program that calls it.

- “Client” code should avoid directly altering the value of an object’s attribute

Private attributes and private methods

```
# Private Neopet
# Demonstrates private variables and methods

class Neopet(object):
    """A virtual pet"""

    def __init__(self, name, mood):
        print("A new neopet has been born!")
        self.name = name    #public attribute
        self.__mood = mood  #private attribute

    def talk(self):
        print("\nI'm", self.name)
        print("Right now I feel", self.__mood, "\n")

# main
pet = Neopet(name = "Flotsam", mood = "happy")

print(pet._Neopet__mood)

input("\n\nPress the enter key to exit.")

```

Accessing private attributes

```
1  # Private Neopet
2  # Demonstrates private variables and methods
3
4  class Neopet(object):
5      """A virtual pet"""
6
7      def __init__(self, name, mood):
8          print "A new neopet has been born!"
9          self.name = name    #public attribute
10         self.__mood = mood  #private attribute
11
12     def talk(self):
13         print "\nI'm", self.name
14         print "Right now I feel", self.__mood, "\n"
15
16 # main
17 pet = Neopet(name = "Flotsam", mood = "happy")
18
19 print pet.mood
20
21 raw_input("\n\nPress the enter key to exit.")
22
```

WRONG

Accessing private attributes

```
1  # Private Neopet
2  # Demonstrates private variables and methods
3
4  class Neopet(object):
5      """A virtual pet"""
6
7      def __init__(self, name, mood):
8          print "A new neopet has been born!"
9          self.name = name     #public attribute
10         self.__mood = mood   #private attribute
11
12     def talk(self):
13         print "\nI'm", self.name
14         print "Right now I feel", self.__mood, "\n"
15
16 # main
17 pet = Neopet(name = "Flotsam", mood = "happy")
18
19 print pet._Neopet__mood
20
21 raw_input("\n\nPress the enter key to exit.")
22
```

Creating private methods

```
# Private Neopet
# Demonstrates private variables and methods

class Neopet(object):
    """A virtual pet"""

    def __init__(self, name, mood):
        print("A new neopet has been born!")
        self.name = name    #public attribute
        self.__mood = mood  #private attribute

    def talk(self):
        print("\nI'm", self.name)
        print("Right now I feel", self.__mood, "\n")

    def __private_method(self):
        print("This is a private method.")

    def public_method(self):
        print("This is a public method.")
        self.__private_method()

# main
pet = Neopet(name = "Flotsam", mood = "happy")
pet.talk()
pet.public_method()


input("\n\nPress the enter key to exit.")
```

Accessing private methods

```
# Private Neopet
# Demonstrates private variables and methods

class Neopet(object):
    """A virtual pet"""

    def __init__(self, name, mood):
        print("A new neopet has been born!")
        self.name = name    #public attribute
        self.__mood = mood  #private attribute

    def talk(self):
        print("\nI'm", self.name)
        print("Right now I feel", self.__mood, "\n")

    def __private_method(self):
        print("This is a private method.")

    def public_method(self):
        print("This is a public method.")
        self.__private_method()

# main
pet = Neopet(name = "Flotsam", mood = "happy")
pet.talk()
pet.public_method()


input("\n\nPress the enter key to exit.")
```

Respecting an object's privacy

```
# Private Neopet
# Demonstrates private variables and methods

class Neopet(object):
    """A virtual pet"""

    def __init__(self, name, mood):
        print("A new neopet has been born!")
        self.name = name    #public attribute
        self.__mood = mood  #private attribute

    def talk(self):
        print("\nI'm", self.name)
        print("Right now I feel", self.__mood, "\n")

    def __private_method(self):
        print("This is a private method.")

    def public_method(self):
        print("This is a public method.")
        self.__private_method()

# main
pet = Neopet(name = "Flotsam", mood = "happy")
pet.talk()
pet.public_method()


input("\n\nPress the enter key to exit.")
```

Controlling Attribute Access: Properties

```
# Property Neopet
# Demonstrates get methods and properties

class Neopet(object):
    """A virtual pet"""

    def __init__(self, name):
        print("A new neopet has been born!")
        self.__name = name

    def get_name(self):
        return self.__name


# main
pet = Neopet(name = "Flotsam")
print(pet.get_name())

input("\n\nPress the enter key to exit.")
```

Using set methods

```
# Property Neopet
# Demonstrates set methods and properties

class Neopet(object):
    """A virtual pet"""

    def __init__(self, name):
        print("A new neopet has been born!")
        self.__name = name

    def get_name(self):
        return self.__name

    def set_name(self, new_name):
        if new_name == "":
            print("A neopet's name can't be an empty string.")
        else:
            self.__name = new_name
            print("Name change successful.")


# main
pet = Neopet(name = "Flotsam")
print(pet.get_name())

pet.set_name("")

pet.set_name("Flounder")
print(pet.get_name())


input("\n\nPress the enter key to exit.")
```

Neopet Program

```
# Neopet Program
# A virtual pet to care for

class Neopet(object):
    """A virtual pet"""
    def __init__(self, name, hunger = 0, boredom = 0):
        self.name = name
        self.hunger = hunger
        self.boredom = boredom

    def __pass_time(self):
        self.hunger += 1
        self.boredom += 1

    def __get_mood(self):
        unhappiness = self.hunger + self.boredom
        if unhappiness < 5:
            mood = "happy"
        elif 5 <= unhappiness <= 10:
            mood = "okay"
        elif 11 <= unhappiness <= 15:
            mood = "frustrated"
        else:
            mood = "mad"
        return mood

    mood = property(__get_mood)

    def talk(self):
        print("I'm", self.name, "and I feel", self.mood, "now.\n")
        self.__pass_time()
```

```
# Neopet Program
# A virtual pet to care for

class Neopet(object):
    """A virtual pet"""
    def __init__(self, name, hunger = 0, boredom = 0):
        self.name = name
        self.hunger = hunger
        self.boredom = boredom

    def __pass_time(self):
        self.hunger += 1
        self.boredom += 1

    def __get_mood(self):
        unhappiness = self.hunger + self.boredom
        if unhappiness < 5:
            mood = "happy"
        elif 5 <= unhappiness <= 10:
            mood = "okay"
        elif 11 <= unhappiness <= 15:
            mood = "frustrated"
        else:
            mood = "mad"
        return mood

    mood = property(__get_mood)

    def talk(self):
        print("I'm", self.name, "and I feel", self.mood, "now.\n")
        self.__pass_time()
```

```
# Neopet Program
# A virtual pet to care for

class Neopet(object):
    """A virtual pet"""
    def __init__(self, name, hunger = 0, boredom = 0):
        self.name = name
        self.hunger = hunger
        self.boredom = boredom

    def __pass_time(self):
        self.hunger += 1
        self.boredom += 1

    def __get_mood(self):
        unhappiness = self.hunger + self.boredom
        if unhappiness < 5:
            mood = "happy"
        elif 5 <= unhappiness <= 10:
            mood = "okay"
        elif 11 <= unhappiness <= 15:
            mood = "frustrated"
        else:
            mood = "mad"
        return mood

    mood = property(__get_mood)

    def talk(self):
        print("I'm", self.name, "and I feel", self.mood, "now.\n")
        self.__pass_time()
```

```
# Neopet Program
# A virtual pet to care for

class Neopet(object):
    """A virtual pet"""
    def __init__(self, name, hunger = 0, boredom = 0):
        self.name = name
        self.hunger = hunger
        self.boredom = boredom

    def __pass_time(self):
        self.hunger += 1
        self.boredom += 1

    def __get_mood(self):
        unhappiness = self.hunger + self.boredom
        if unhappiness < 5:
            mood = "happy"
        elif 5 <= unhappiness <= 10:
            mood = "okay"
        elif 11 <= unhappiness <= 15:
            mood = "frustrated"
        else:
            mood = "mad"
        return mood

    mood = property(__get_mood)

    def talk(self):
        print("I'm", self.name, "and I feel", self.mood, "now.\n")
        self.__pass_time()
```

```
33      def eat(self, food = 4):
34          print("Burrrppp.  Thank you.")
35          self.hunger -= food
36          if self.hunger < 0:
37              self.hunger = 0
38          self.__pass_time()
39
40      def play(self, fun = 4):
41          print("Wheee!")
42          self.boredom -= fun
43          if self.boredom < 0:
44              self.boredom = 0
45          self.__pass_time()
46
47
48  def main():
49      pet_name = input("What do you want to name your neopet?: ")
50      pet = Neopet(pet_name)
51
```

```
33      def eat(self, food = 4):
34          print("Burrrppp.  Thank you.")
35          self.hunger -= food
36          if self.hunger < 0:
37              self.hunger = 0
38          self.__pass_time()
39
40      def play(self, fun = 4):
41          print("Wheee!")
42          self.boredom -= fun
43          if self.boredom < 0:
44              self.boredom = 0
45          self.__pass_time()
46
47
48  def main():
49      pet_name = input("What do you want to name your neopet?: ")
50      pet = Neopet(pet_name)
51
```

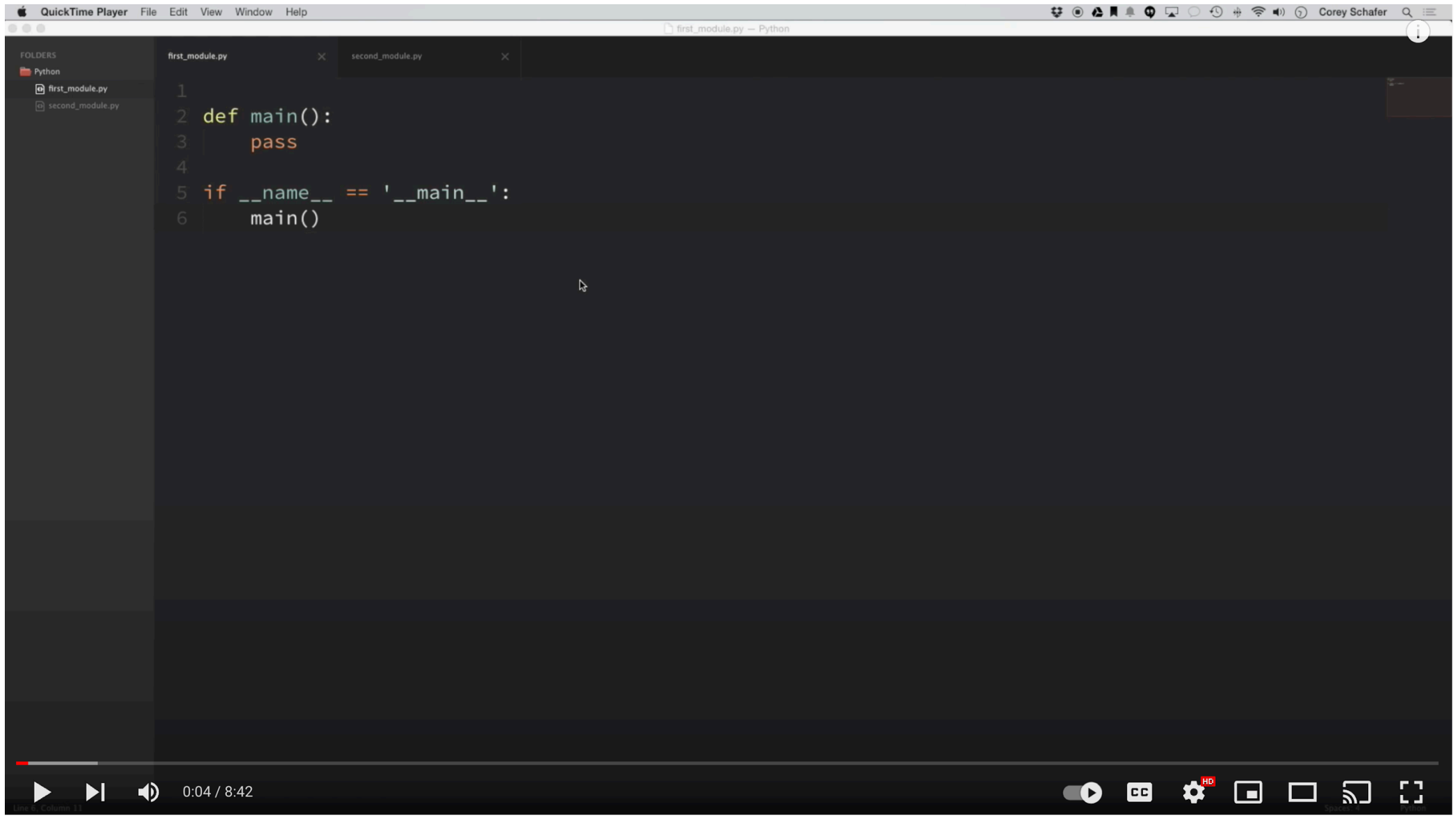
```
33      def eat(self, food = 4):
34          print("Burrrppp.  Thank you.")
35          self.hunger -= food
36          if self.hunger < 0:
37              self.hunger = 0
38          self.__pass_time()
39
40      def play(self, fun = 4):
41          print("Wheee!")
42          self.boredom -= fun
43          if self.boredom < 0:
44              self.boredom = 0
45          self.__pass_time()
46
47
48  def main():
49      pet_name = input("What do you want to name your neopet?: ")
50      pet = Neopet(pet_name)
51
```

```
    choice = None
    while choice != "0":
        print("""
        Neopet Options

        0 - Quit
        1 - Listen to your pet
        2 - Feed your pet
        3 - Play with your pet
        """)
        choice = input("Choice: ")
        print

        # exit
        if choice == "0":
            print("Good-bye.")

        # listen to your pet
        elif choice == "1":
            pet.talk()

        # feed your pet
        elif choice == "2":
            pet.eat()

        # play with your pet
        elif choice == "3":
            pet.play()

        # some unknown choice
        else:
            print("\nSorry, but", choice, "isn't a valid choice.")

main()
input("\n\nPress the enter key to exit.")
```

```
    choice = None
    while choice != "0":
        print("""
        Neopet Options

        0 - Quit
        1 - Listen to your pet
        2 - Feed your pet
        3 - Play with your pet
        """)
        choice = input("Choice: ")
        print

        # exit
        if choice == "0":
            print("Good-bye.")

        # listen to your pet
        elif choice == "1":
            pet.talk()

        # feed your pet
        elif choice == "2":
            pet.eat()

        # play with your pet
        elif choice == "3":
            pet.play()

        # some unknown choice
        else:
            print("\nSorry, but", choice, "isn't a valid choice.")

main()
input("\n\nPress the enter key to exit.")
```

if __name__ == ‘__main__’”

https://www.youtube.com/watch?v=sugvnHA7ElY



https://youtu.be/oaiQ5hYKHTE?si=66MdMneVhZ0BooDr