327 Object-oriented Programming

Lecture 3

9/8/2021

Professor Barron

From last time… terminology

- data in an object
  - instance variables
  - instance data
  - members
  - fields
  - attributes
  - properties
- behaviors
  - methods

Today

- Revisiting Point class
- Organizing project modules and imports
- Docstrings
- Instance/class/static members
- Public/private members
- Immutable/Mutable defaults
- Getters/setters
- Notebook case-study
  - particularly useful for homework 1

Organizing a project

- Each python file is a module
  - accounts.py
- If everything is in one folder…
- Some options
  - `import accounts`
    - `accounts.SavingsAccount()`
  - `from accounts import SavingsAccount`
  - `from accounts import CheckingAccount`
  - `from accounts import *`
  - `from accounts import SavingsAccount as sa`

Packages

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

```
import sound.effects.echo
sound.effects.echo.echofilter(input, output,
                                  delay=0.7, atten=4)
```

```
from sound.effects import echo
echo.echofilter(input, output, delay=0.7, atten=4)
```

```
from sound.effects.echo import echofilter
echofilter(input, output, delay=0.7, atten=4)
```

```
# cannot do
import sound.effects.echo.echofilter
```

Relative package reference

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

If you are in surround.py

```
from . import echo
from .. import formats
from ..filters import equalizer
```

See the python docs for more detail
https://docs.python.org/3/tutorial/modules.html

Docstrings

- Documenting the purpose and behavior of objects and methods
- Describe the public interface
- If someone else wants to reuse your code what do they need to know?

- Shows info when you hover over code in an IDE
- Can automatically generate a webpage with your documentation

```
class Point:
    "Represents a point in two-dimensional geometric coordinates"


    def __init__(self, new_x=0, new_y=0):
        """Initialize the position of a new point. The x and y
        coordinates can be specified. If they are not, the
        point defaults to the origin."""

        self.x = new_x
        self.y = new_y

    def reset(self):
        "Reset the point back to the geometric origin: 0, 0"

        self.move(0, 0)

    def move(self, new_x, new_y):
        "Move the point to a new location in 2D space."

        self.x = new_x
        self.y = new_y
```

Public vs private

- Public attributes and methods
  - Usable from code outside the instance
  - “permanent” public interface
- Private attributes and methods
  - Usable within an instance (and possibly instances of a subclass)
  - No direct access from other objects
- Protected
  - Usable within a module or by subclasses

Public vs private in Python

- Python does not explicitly enforce public/private
- Just because python doesn’t force us, doesn’t mean everything should be public!
- Start private variables and methods with underscore

- Name mangling
  - if you want to be a little safer
  - Start name with double underscore

Instance vs class vs static

- Instance
  - Methods take self as an argument
  - Variable is attached to a particular instance of a class
  - Each instance has separate copies of the variable
- Class
  - Methods take cls as argument
  - Variables attached to the class instead of an instance
  - Each instance can access the class variables
- Static
  - Not associated with any class or instance state
  - Present in a class just for organizational purposes

Confusion with mutable defaults

- Careful with mutable objects as defaults
- It is created once when the def statement is run

```
class ClassA:
   def __init__(self, names=[])
      self.my_names = names
      self.my_names.append(“Tim”)
```

Confusion with mutable defaults

- Careful with mutable objects as defaults
- It is created once when the def statement is run

```
class ClassA:
    def __init__(self, names=[])
        self.my_names = names
        self.my_names.append("Tim")
```

obj1 = ClassA()

obj2 = ClassA()

obj1.my-names ⇒ ["Tim", "Tim"]

Confusion with mutable defaults

- Careful with mutable objects as defaults
- It is created once when the def statement is run

```
class ClassA:
    def __init__(self, names=[])
        self.my_names = names
        self.my_names.append("Tim")
```

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Confusion with mutable defaults

- Careful with mutable objects as defaults
- It is created once when the def statement is run

```
class ClassA:
   def __init__(self, names=None)
      if not names:
            self.my_names = []
      else:
            self.my_names = names
      self.my_names.append(“Tim”)
```

More confusion upcoming?

- Pattern matching in Python 3.10…
- read about it here
  - https://www.python.org/dev/peps/pep-0636/
- a funny take on how this will cause confusion
  - https://brennan.io/2021/02/09/so-python/

```
match status_code:
    case 200:
        print("OK!")
    case 404:
        print("HTTP Not Found")
    case _:
        print(“Something else, sorry!")
```

More confusion upcoming?

- Pattern matching in Python 3.10…
- read about it here
  - https://www.python.org/dev/peps/pep-0636/
- a funny take on how this will cause confusion
  - https://brennan.io/2021/02/09/so-python/

```
NOT_FOUND = 404
match status_code:
    case 200:
        print("OK!")
    case NOT_FOUND:
        print("HTTP Not Found")
```

NOT_FOUND = 301

Getters/Setters

- Also called accessors/mutators
- Expose private parts of an object to the public interface
- Better than using instance variables directly
  - why?
- Often a bad idea
  - Why does another object need access?
  - Could the task be achieved within the original object?

- https://www.infoworld.com/article/2073723/why-getter-and-setter-methods-are-evil.html