Fundamentals of Programming I

Introduction to Programmer-Defined Classes

Objects, Classes, and Methods

- Every data value in Python is an *object*
- Every object is an instance of a *class*
- Built in classes include **`int`**, **`float`**, **`str`**, **`tuple`**, **`list`**, **`dict`**
- A class includes operations (*methods*) for manipulating objects of that class (**`append`**, **`pop`**, **`sort`**, **`find`**, etc.)
- Operators (**`==`**, **`[]`**, **`in`**, **`+`**, etc.) are "syntactic sugar" for methods

What Do Objects and Classes Do for Us?

- An object bundles together data and operations on those data
- A computational object can model practically any object in the real (natural or artificial) world
- Some classes come with a programming language
- Any others must be defined by the programmer

Programmer-Defined Classes

- The **`EasyFrame`** class is used to create GUI windows that are easy to set up
- The **`Image`** class is used to load, process, and save images
- Like the built-in classes, these classes include operations to run with their instances

Other Examples

- A **`Student`** class represents information about a student and her test scores
- A **`Rational`** class represents rational numbers and their operations
- A **`Die`** class represents dice used in games
- **`SavingsAccount`**, **`CheckingAccount`**, **`Bank`**, and **`ATM`** are used to model a banking system
- **`Proton`**, **`Neutron`**, **`Electron`**, and **`Positron`** model subatomic particles in nuclear physics

The **`Die`** Class: Its Interface and Use

Interface

```
die.py                    # The module for the Die class

Die()              # Returns a new Die object

roll()                    # Resets the die's value

getValue()                # Returns the die's value
```

The **Die** Class: Its Interface and Use

Interface

```
die.py                    # The module for the Die class

Die()             # Returns a new Die object

roll()                    # Resets the die's value

getValue()                # Returns the die's value
```

Use

```
from die import Die

d = Die()                  # Create a new Die object

d.roll()                   # Roll it

print(d.getValue())        # Display its value

help(Die)                  # Look up the documentation
```

Specifying an Interface

- The user of a class is only concerned with learning the information included in the headers of the class’s methods
- This information includes the method name and parameters
- Collectively, this information comprises the class’s *interface*
- Docstrings describe what the methods do

Defining (Implementing) a Class

- The *definition* or *implementation* of a class includes completed descriptions of an object’s data and the methods for accessing and modifying those data
- The data are contained in *instance variables* and the methods are called *instance methods*
- Related class definitions often occur in the same module

Syntax Template for a Simple Class Definition

```
<docstring for the module>

<imports of other modules used>

class <name>(<parent class name>):
    <docstring for the class>

    <method definitions>
```

Basically a header followed by several method definitions

Defining the **`Die`** Class

```
from random import randint

class <name>(<parent class name>):
    <docstring for the class>

    <method definitions>
```

We’ll use **`random.randint`** to roll the die

The Class Header

```
from random import randint

class Die(object):
    <docstring for the class>

    <method definitions>
```

By convention, programmer-defined class names are capitalized in Python

Built-in class names, like **`str`**, **`list`**, and **`object`**, are not

Like built-in function names, built-in class names appear in purple

The Class Header

```
from random import randint

class Die(object):
    <docstring for the class>

    <method definitions>
```

All Python classes are *subclasses* of the **`object`** class

A class can *inherit* behavior from its parent class

The Class Docstring

```
from random import randint

class Die(object):
    """This class represents a six-sided die."""

    <method definitions>
```

A class’s *docstring* describes the purpose of the class

Setting the Initial State

```
from random import randint

class Die(object):
    """This class represents a six-sided die."""

    def __init__(self):
        self.value = 1
```

A method definition looks a bit like a function definition

The **`__init__`** method (also called a *constructor*) is automatically run when an object is instantiated; this method usually sets the object's initial state
(**`d = Die()`**)

The **`self`** Parameter

```
from random import randint

class Die(object):
    """This class represents a six-sided die."""

    def __init__(self):
        self.value = 1
```

The name **`self`** must appear as the first parameter in each instance method definition

Python uses this parameter to refer to the object on which the method is called

Instance Variables

```
from random import randint

class Die(object):
    """This class represents a six-sided die."""

    def __init__(self):
        self.value = 1
```

`self` must also be used with all instance method calls and instance variable references within the defining class

`self` refers to the current object (a die)

Using Instance Variables

```
from random import randint

class Die(object):
    """This class represents a six-sided die."""

    def __init__(self):
        self.value = 1

    def roll(self):
        """Resets the die's value."""
        self.value = randint(1, 6)

    def getValue(self):
        return self.value
```

`self.value` refers to this object's *instance variable*

Where Are Classes Defined?

- Like everything else, in a module
- Define the **`Die`** class in a **`die`** module
- Related classes usually go in the same module (**`SavingsAccount`** and **`Bank`** the **`bank`** module)