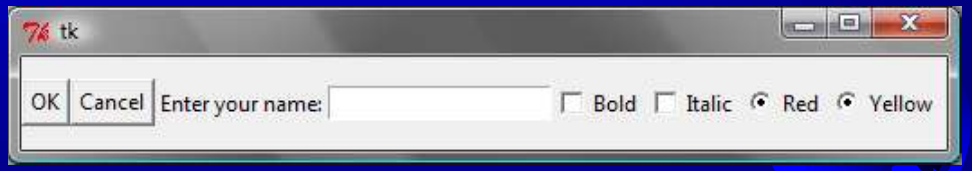
Chapter 7 Object-Oriented Programming

Motivations

After learning the preceding chapters, you are capable of solving many programming problems using selections, loops, and functions. However, these Python features are not sufficient for developing graphical user interfaces and large scale software systems. Suppose you want to develop a graphical user interface as shown below. How do you program it?



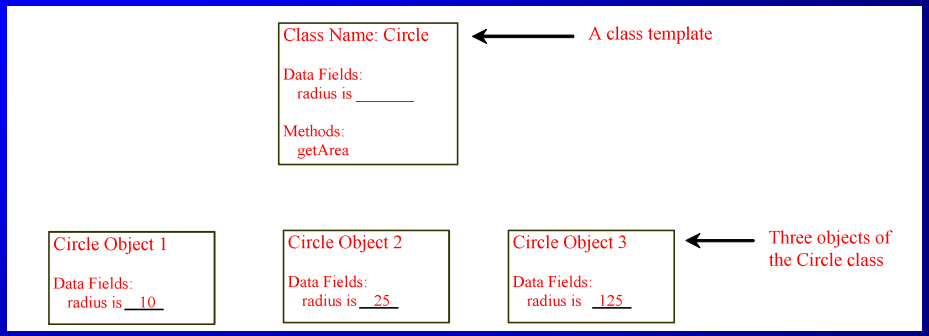
Objectives

- To describe objects and classes, and use classes to model objects (§7.2).
- To define classes (§7.2.1).
- To construct an object using a constructor that invokes the initializer to create and initialize data fields (§7.2.2).
- To access the members of objects using the dot operator (.) (§7.2.3).
- To reference an object itself with the self parameter (§7.2.4).
- To use UML graphical notation to describe classes and objects (§7.3).
- To distinguish between immutable and mutable (§7.4).
- To hide data fields to prevent data corruption and make classes easy to maintain (§7.5).
- To apply class abstraction and encapsulation to software development (§7.6).
- To explore the differences between the procedural paradigm and the object-oriented paradigm (§7.7).

OO Programming Concepts

Object-oriented programming (OOP) involves programming using objects. An *object* represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects. An object has a unique identity, state, and behaviors. The *state* of an object consists of a set of *data fields* (also known as *properties*) with their current values. The *behavior* of an object is defined by a set of methods.

Objects



An object has both a state and behavior. The state defines the object, and the behavior defines what the object does.

Classes

A Python class uses variables to store data fields and defines methods to perform actions. Additionally, a class provides a special type method, known as *initializer*, which is invoked to create a new object. An initializer can perform any action, but initializer is designed to perform initializing actions, such as creating the data fields of objects.

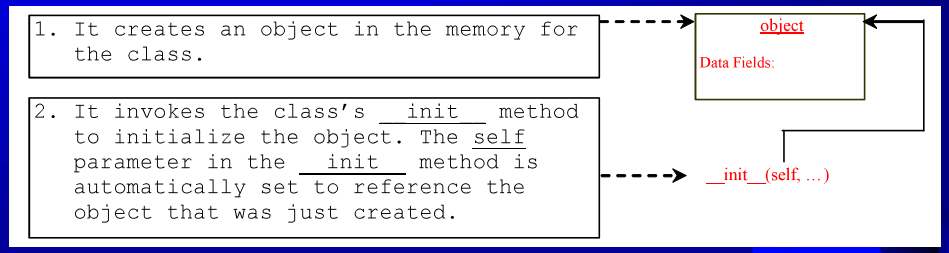
```
class ClassName:
    initializer
    methods
```

Circle | TestCircle | Run

Constructing Objects

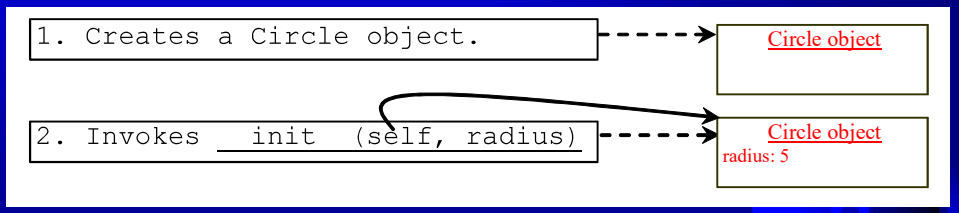
Once a class is defined, you can create objects from the class by using the following syntax, called a *constructor*:

```
className(arguments)
```



Constructing Objects

The effect of constructing a Circle object using Circle(5) is shown below:



Instance Methods

Methods are functions defined inside a class. They are invoked by objects to perform actions on the objects. For this reason, the methods are also called *instance methods* in Python. You probably noticed that all the methods including the constructor have the first parameter **self**, which refers to the object that invokes the method. You can use any name for this parameter. But by convention, **self** is used.

Accessing Objects

After an object is created, you can access its data fields and invoke its methods using the dot operator (.), also known as the *object member access operator*. For example, the following code accesses the radius data field and invokes the getPerimeter and getArea methods.

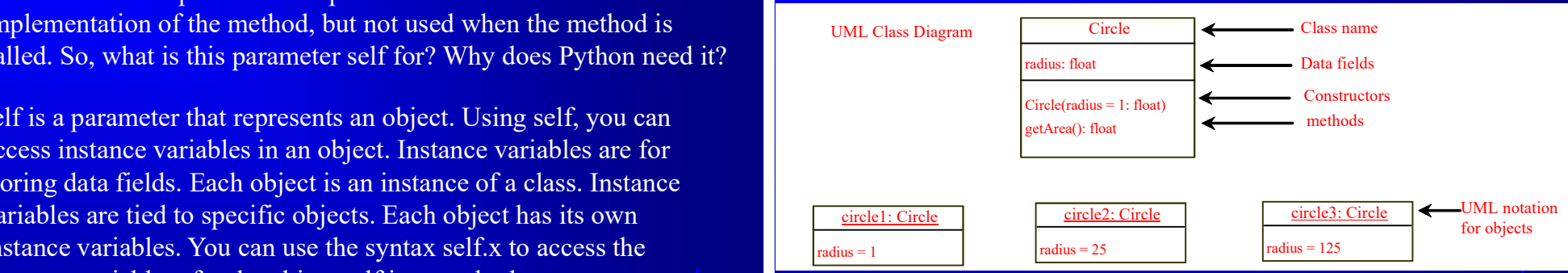
```
>>> from Circle import Circle
>>> c = Circle(5)
>>> c.getPerimeter()
31.41592653589793
>>> c.radius = 10
>>> c.getArea()
314.1592653589793
```


Why self?

Note that the first parameter is special. It is used in the implementation of the method, but not used when the method is called. So, what is this parameter self for? Why does Python need it?

self is a parameter that represents an object. Using self, you can access instance variables in an object. Instance variables are for storing data fields. Each object is an instance of a class. Instance variables are tied to specific objects. Each object has its own instance variables. You can use the syntax self.x to access the instance variable x for the object self in a method.

UML Class Diagram



Example: Defining Classes and Creating Objects

TV	
channel: int	The current channel (1 to 120) of this TV.
volumeLevel: int	The current volume level (1 to 7) of this TV.
on: bool	Indicates whether this TV is on/off.
TV()	Constructs a default TV object.
turnOn(): None	Turns on this TV.
turnOff(): None	Turns off this TV.
getChannel(): int	Returns the channel for this TV.
setChannel(channel: int): None	Sets a new channel for this TV.
getVolume(): int	Gets the volume level for this TV.
setVolume(volumeLevel: int): None	Sets a new volume level for this TV.
channelUp(): None	Increases the channel number by 1.
channelDown(): None	Decreases the channel number by 1.
volumeUp(): None	Increases the volume level by 1.
volumeDown(): None	Decreases the volume level by 1.

TV

TestTV

Run

animation

Trace Code

```
myCircle = Circle(5.0)

yourCircle = Circle()

yourCircle.radius = 100
```

Assign object reference to myCircle

myCircle reference value

: Circle

radius: 5.0

animation

Trace Code

```
myCircle = Circle(5.0)

yourCircle = Circle()

yourCircle.radius = 100
```

myCircle reference value

: Circle

radius: 5.0

Assign object reference to yourCircle

yourCircle reference value

: Circle

radius: 1.0

animation

Trace Code

```
myCircle = Circle(5.0)

yourCircle = Circle()

yourCircle.radius = 100
```

myCircle reference value

: Circle

radius: 5.0

Modify radius in yourCircle

yourCircle reference value

: Circle

radius: 100

The datetime Class

```
from datetime import datetime
d = datetime.now()
print("Current year is " + str(d.year))
print("Current month is " + str(d.month))
print("Current day of month is " + str(d.day))
print("Current hour is " + str(d.hour))
print("Current minute is " + str(d.minute))
print("Current second is " + str(d.second))
```


Data Field Encapsulation

To protect data.

To make class easy to maintain.

To prevent direct modifications of data fields, don't let the client directly access data fields. This is known as *data field encapsulation*. This can be done by defining private data fields. In Python, the private data fields are defined with two leading underscores. You can also define a private method named with two leading underscores.

CircleWithPrivateDataRadius

Data Field Encapsulation

CircleWithPrivateDataRadius

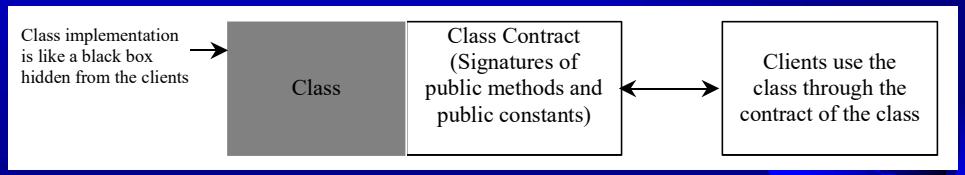
```
>>> from CircleWithPrivateRadius import Circle
>>> c = Circle(5)
>>> c.__radius
AttributeError: 'Circle' object has no attribute
'__radius'
>>> c.getRadius()
 5
```


Design Guide

If a class is designed for other programs to use, to prevent data from being tampered with and to make the class easy to maintain, define data fields private. If a class is only used internally by your own program, there is no need to encapsulate the data fields.

Class Abstraction and Encapsulation

Class abstraction means to separate class implementation from the use of the class. The creator of the class provides a description of the class and let the user know how the class can be used. The user of the class does not need to know how the class is implemented. The detail of implementation is encapsulated and hidden from the user.



Designing the Loan Class

The – sign denotes a private data field.

The get methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

Loan
-annualInterestRate: float -numberOfYears: int -loanAmount: float -borrower: str
Loan(annualInterestRate: float, numberOfYear: int, loanAmount: float, borrower: str)

The annual interest rate of the loan (default: 2.5).
The number of years for the loan (default: 1)
The loan amount (default: 1000).
The borrower of this loan.

Constructs a Loan object with the specified annual interest rate, number of years, loan amount, and borrower.

Loan TestLoanClass Run

Object-Oriented Thinking

This book's approach is to teach problem solving and fundamental programming techniques before object-oriented programming. This section will show how procedural and object-oriented programming differ. You will see the benefits of object-oriented programming and learn to use it effectively. We will use several examples in the rest of the chapter to illustrate the advantages of the object-oriented approach. The examples involve designing new classes and using them in applications.

The BMI Class

The get methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

BMI
-name: str -age: int -weight: float -height: float
BMI(name: str, age: int, weight: float, height: float) getBMI(): float getStatus(): str

The name of the person.
The age of the person.
The weight of the person in pounds.
The height of the person in inches.

Creates a BMI object with the specified name, weight, height, and a default age 20.
Returns the BMI
Returns the BMI status (e.g., normal, overweight, etc.)

BMI UseBMIClass Run

Procedural vs. Object-Oriented

In procedural programming, data and operations on the data are separate, and this methodology requires sending data to methods. Object-oriented programming places data and the operations that pertain to them in an object. This approach solves many of the problems inherent in procedural programming.

Procedural vs. Object-Oriented

The object-oriented programming approach organizes programs in a way that mirrors the real world, in which all objects are associated with both attributes and activities. Using objects improves software reusability and makes programs easier to develop and easier to maintain. Programming in Python involves thinking in terms of objects; a Python program can be viewed as a collection of cooperating objects.