CITY UNIVERSITY
LONDON

Introduction to programming using Python

Session 7

Matthieu Choplin

matthieu.choplin1@city.ac.uk

http://moodle.city.ac.uk/

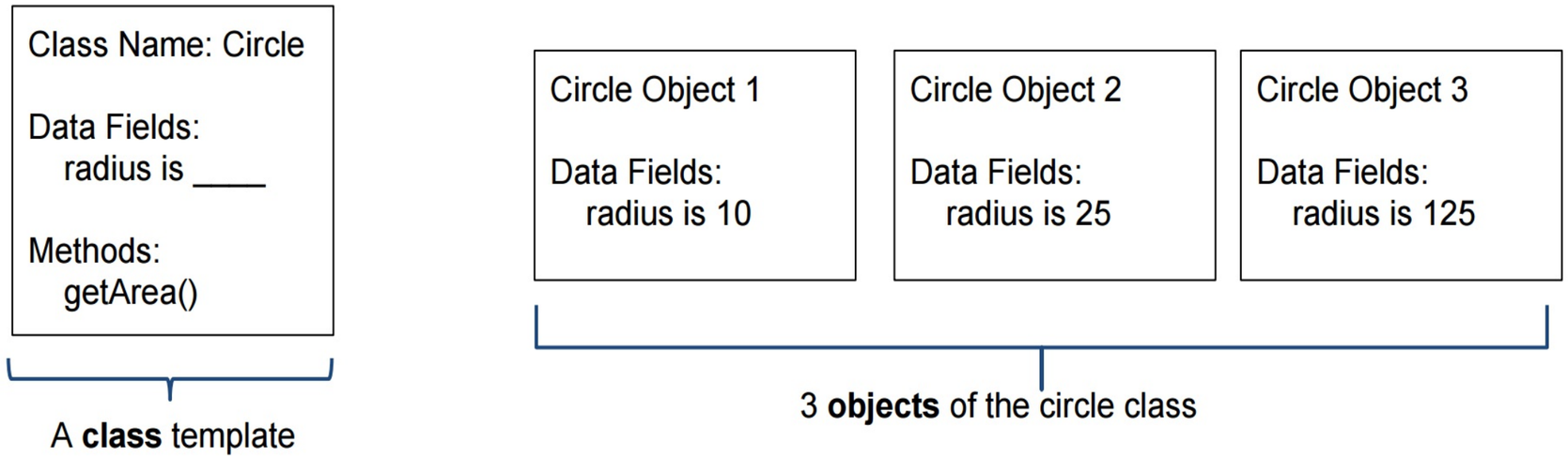
Objectives

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

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 initialising actions, such as creating the data fields of objects.

```
class ClassName:
 # initializer
 # methods
```

Example for creating a class

```
import math

class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.radius = radius

    def getPerimeter(self):
        return 2 * self.radius * math.pi

    def getArea(self):
        return self.radius * self.radius * math.pi

    def setRadius(self, radius):
        self.radius = radius
```

Example for using a class and creating objects

```
from Circle import Circle

def main():
    # Create a circle with radius 1
    circle1 = Circle()
    print("The area of the circle of radius",
        circle1.radius, "is", circle1.getArea())

    # Create a circle with radius 25
    circle2 = Circle(25)
    print("The area of the circle of radius",
        circle2.radius, "is", circle2.getArea())

    # Create a circle with radius 125
    circle3 = Circle(125)
    print("The area of the circle of radius",
        circle3.radius, "is", circle3.getArea())

    # Modify circle radius
    circle2.radius = 100
    print("The area of the circle of radius",
        circle2.radius, "is", circle2.getArea())

main() # Call the main function
```

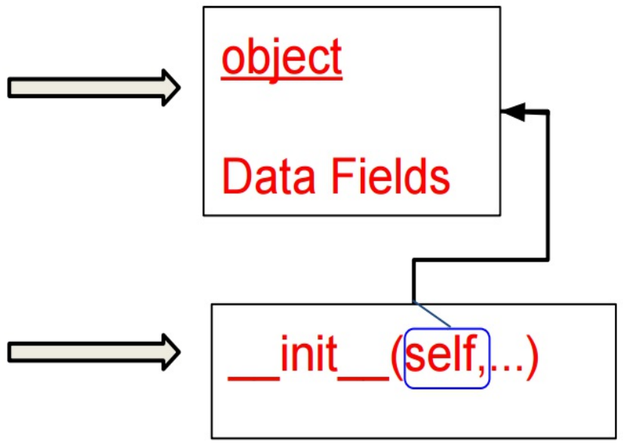
Constructing Objects

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

```
className(arguments)
```

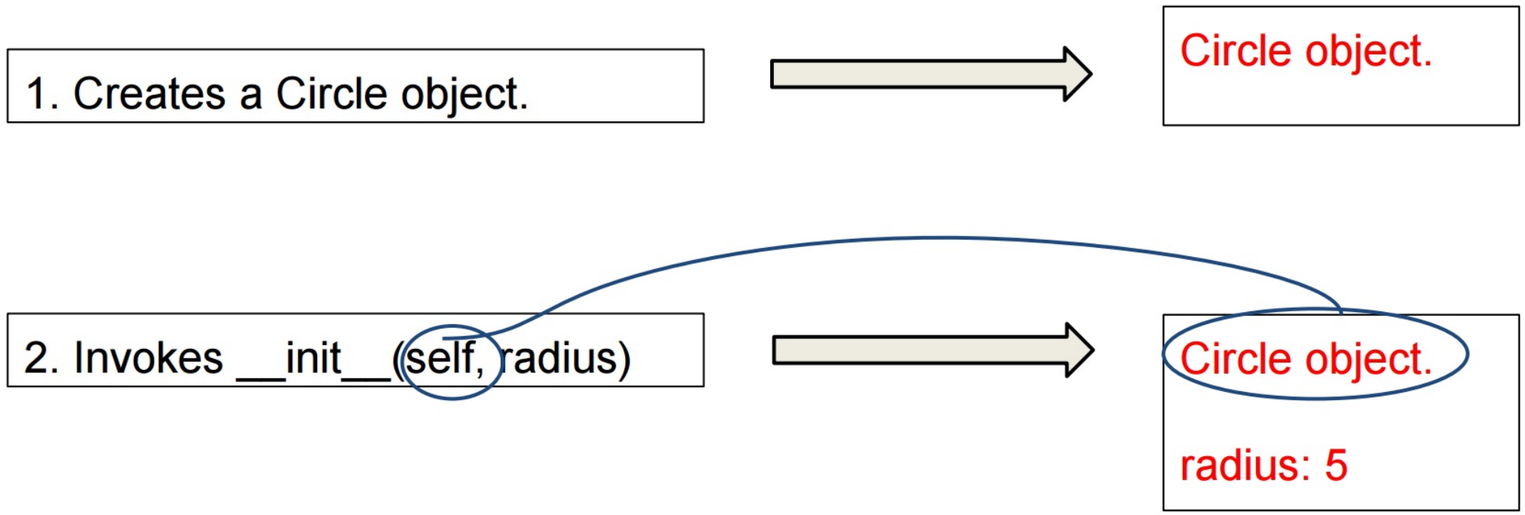
1. It creates an object in the memory for the class.

2. It invokes the class's __init__ method to initialize the object. The self parameter in the __init__ method is automatically set to reference the object that was just created.



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

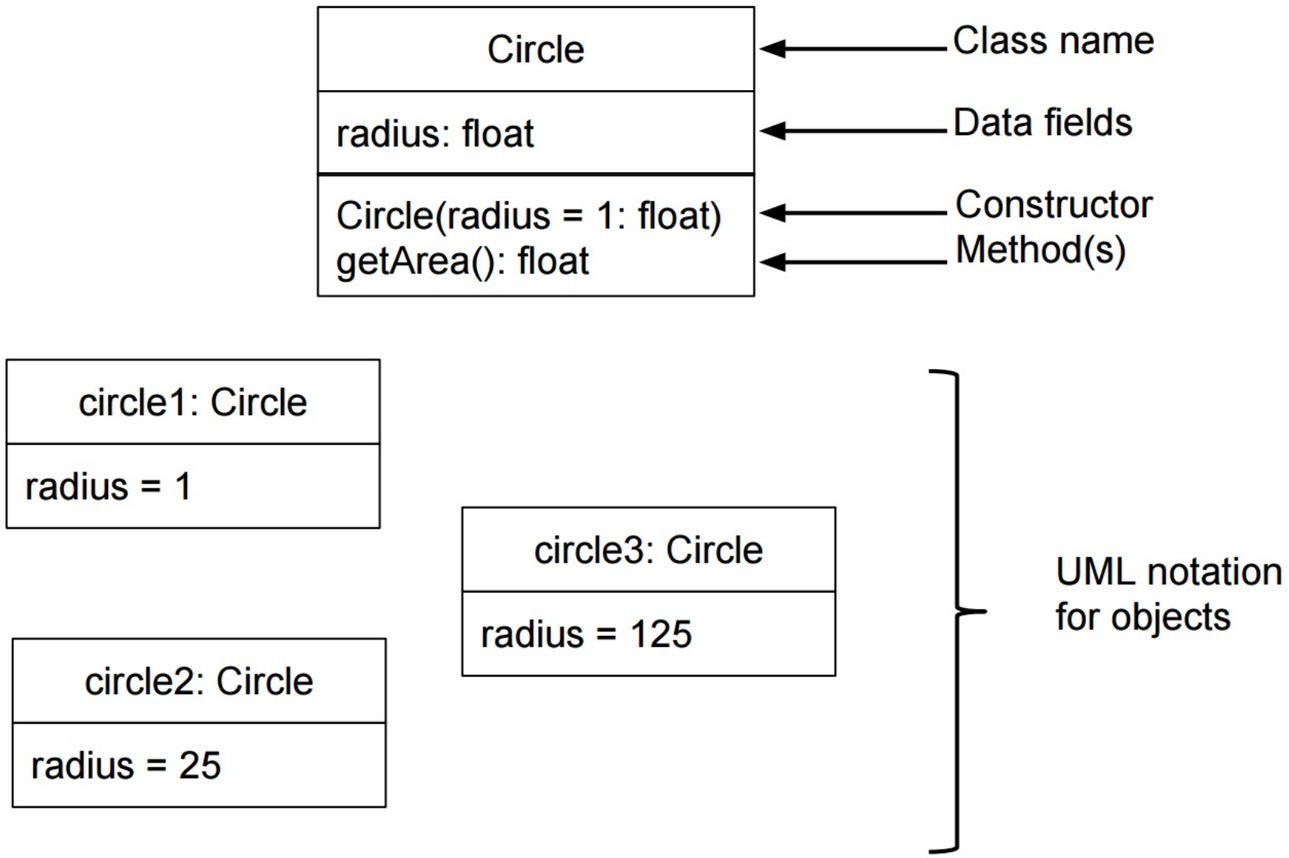
- After an object is created, you can access its data fields and invoke its methods using the dot operator (.), also known as the object member access operator. For example, the following code accesses the radius data field and invokes the getPerimeter and getArea methods.

```
[In 1]from Circle import Circle
[In 2]c = Circle(5)
[In 3]c.getPerimeter()
[Out 3]31.41592653589793
[In 4]c.radius = 10
[In 4]c.getArea()
[Out 4]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

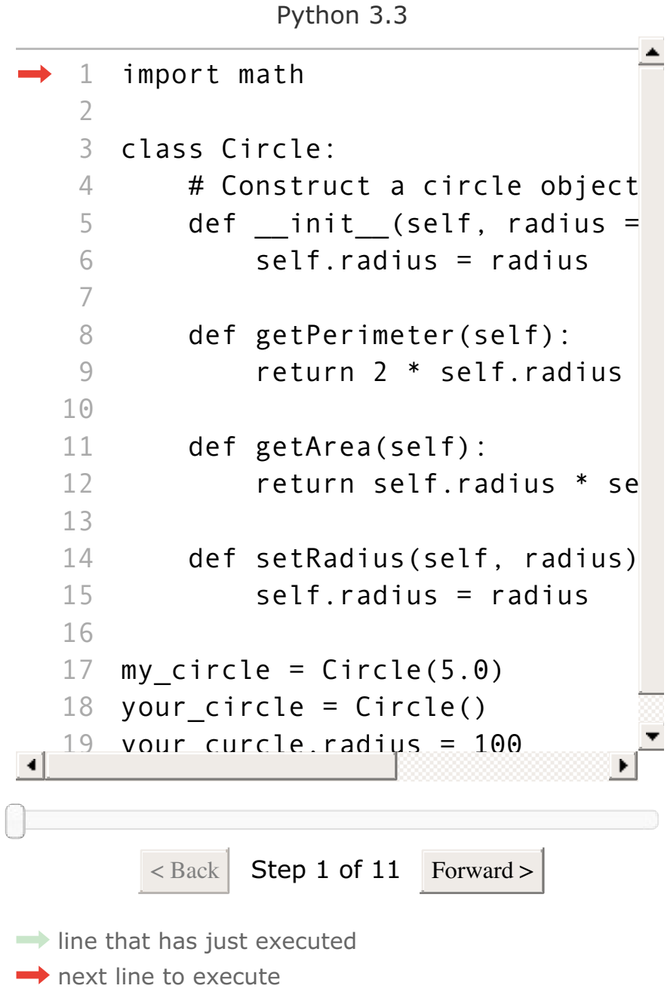
TV	
channel: int volumeLevel: int on: boolData fields	The current channel (1 to 120) of this TV. The current volume level (1 to 7) of this TV. Indicates whether this TV is on/off.
TV() turnOn(): None turnOff(): None getChannel(): int setChannel(channel: int): None getVolume(): int setVolume(volumeLevel: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

Example: Defining Classes and Creating Objects

- See TV.py
- See TestTV.py

CITY UNIVERSITY LONDON

Trace execution



Exercise - The Rectangle class

Following the example of the Circle class, design a class named Rectangle to represent a rectangle. The class contains:

- Two data fields named width and height.
- A constructor that creates a rectangle with the specified width and height. The default values are 1 and 2 for the width and height, respectively.
- A method named getArea()that returns the area of this rectangle
- A method named getPerimeter()that returns the perimeter

Draw the UML diagram for the class, and then implement the class. Write a test program that creates two Rectangle objects—one with width 4 and height 40 and the other with width 3.5 and height 35.7. Display the width, height, area, and perimeter of each rectangle in this order.

Data Field Encapsulation

- To protect data.
- To make classes 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.

Example for making fields private

```
import math

class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.__radius = radius

    def getRadius(self):
        return self.__radius

    def getPerimeter(self):
        return 2 * self.__radius * math.pi

    def getArea(self):
        return self.__radius * self.__radius * math.pi
```

Data Field Encapsulation

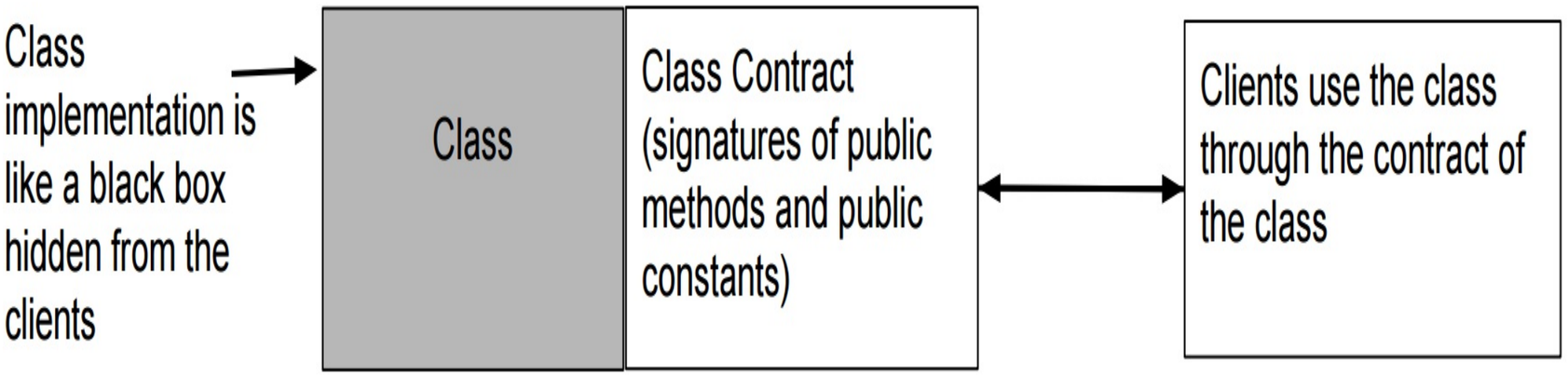
```
>> from CircleWithPrivateRadius import Circle
>>> c = Circle(5)
>>> c.__radius
AttributeError: 'Circle' object has no attribute '__radi
>>> c.getRadius()
 5
```

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

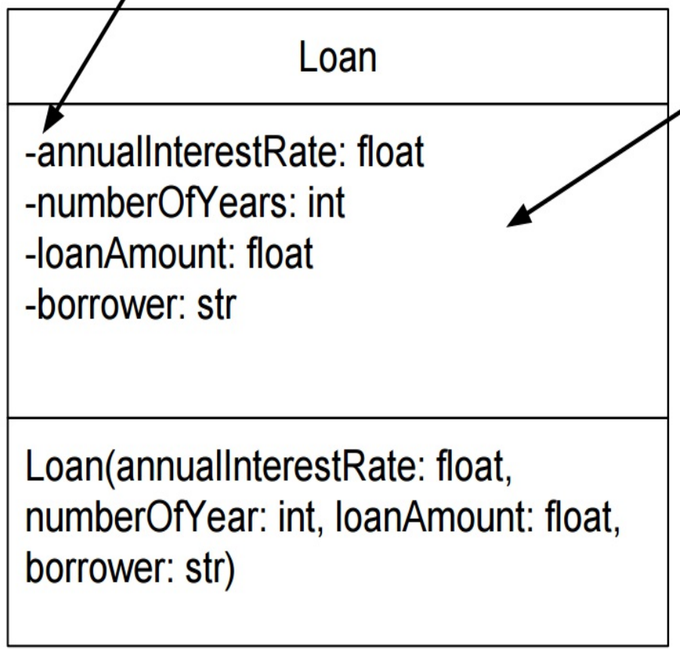
- Class abstraction means to separate class implementation from the use of the class. The creator of the class provides a description of the class and let the user know how the class can be used. The user of the class does not need to know how the class is implemented. The detail of implementation is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



Designing the Loan Class

The _ sign denotes a private data fields

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



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.

- See the program Loan.py
- See the program TestLoanClass.py

Exercise - Stock Price

Design a class named Stock to represent a company's stock that contains:

- A private string data field named symbol for the stock's symbol.
- A private string data field named name for the stock's name.
- A private float data field named previousClosingPrice that stores the stock price for the previous day
- A private float data field named currentPrice that stores the stock price for the current time.
- A constructor that creates a stock with the specified symbol, name, previous price, and current price
- A get method for returning the stock name
- A get method for returning the stock symbol
- Get and set methods forgetting/setting the stock's previous price.
- Get and set methods for getting/setting the stock's current price.
- A method named getChangePercent() that returns the percentage changed from previousClosingPrice to currentPrice

Draw the UML diagram for the class, and then implement the class. Write a test program that creates a Stock object with the stock symbol INTC, the name Intel Corporation, the previous closing price of 20.5, and the new current price of 20.35, and display the price change percentage.

See the difference between procedural and OO thinking

Write a program that prompts the user to enter a weight in pounds and height in inches and then displays the BMI. Note that one pound is 0.45359237 kilograms and one inch is 0.0254 meters.

- See the program BMI_procedural.py

But we could also write this program using classes

The BMI Class

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.)

The BMI Class

- See the program BMI.py
- See the program UseBMIClass.py

Procedural vs. Object-Oriented (1)

- 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 (2)

- 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.

Exercise - Stopwatch

Design a class named StopWatch. The class contains:

- The private data fields startTime and endTime with get methods.
- A constructor that initializes startTime with the current time.
- A method named start() that resets the startTime to the current time.
- A method named stop() that sets the endTime to the current time.
- A method named getElapsedTime() that returns the elapsed time for the stop watch in milliseconds.

Solution

- rectangle.py
- stock.py
- stopwatch.py