

# Classes and object-oriented programming

## Chapter 15

# This chapter covers

- Defining classes
- Using instance variables and @property
- Defining methods
- Defining class variables and methods
- Inheriting from other classes
- Making variables and methods private
- Inheriting from multiple classes

# Defining classes

- A class in Python is effectively a data type.
- All the data types built into Python are classes, and Python gives you powerful tools to manipulate every aspect of a class's behavior.
- You define a class with the class statement:

```
class MyClass:  
    body
```

- Create a new object of the class type (an instance of the class) by calling the class name as a function:

```
instance = MyClass()
```

# Example class

- Class instances can be used as structures or records.
- Unlike C structures or Java classes, the data fields of an instance don't need to be declared ahead of time; they can be created on the fly.
- The following short example defines a class called Circle, creates a Circle instance, assigns a value to the radius field of the circle, and then uses that field to calculate the circumference of the circle

```
1  # define a class
2  class Circle:
3      |    def __init__(self):
4      |        self.radius = 1
5
6  # create an instance of a class
7  my_circle = Circle()
8  print(2 * 3.14 * my_circle.radius)
9
10 # set property
11 my_circle.radius = 5
12 print(2 * 3.14 * my_circle.radius)
```

# Instance variables

- Take a look at the Circle class again, radius is an instance variable of Circle instances.
- That is, each instance of the Circle class has its own copy of radius, and the value stored in that copy may be different from the values stored in the radius variable in other instances.
- In Python, you can create instance variables as necessary by assigning to a field of a class instance:

```
instance.variable = value
```

- If the variable doesn't already exist, it's created automatically, which is how `__init__` creates the radius variable.

# Methods

- A method is a function associated with a particular class.
- You've already seen the special `__init__` method, which is called on a new instance when that instance is created.

```
15 class Circle(Shape):
16     """Circle Class: inherits from Shape and has method area"""
17     pi = 3.14159
18     def __init__(self, r=1, x=0, y=0):
19         Shape.__init__(self, x, y)
20         self.radius = r
21     def area(self):
22         """Circle area method: returns the area of the circle."""
23         return self.radius * self.radius * self.pi
24     def __str__(self):
25         return "Circle of radius %s at coordinates (%d, %d)" \
26             % (self.radius, self.x, self.y)
```

# Class variables

- A class variable is a variable associated with a class, not an instance of a class, and is accessible by all instances of the class.
- A class variable might be used to keep track of some class-level information, such as how many instances of the class have been created at any point.
- A class variable is created by an assignment in the class body, not in the `__init__` function.

```
15 class Circle(Shape):
16     """Circle Class: inherits from Shape and has method area"""
17     pi = 3.14159
18     def __init__(self, r=1, x=0, y=0):
19         Shape.__init__(self, x, y)
20         self.radius = r
21     def area(self):
22         """Circle area method: returns the area of the circle."""
23         return self.radius * self.radius * self.pi
24     def __str__(self):
25         return "Circle of radius %s at coordinates (%d, %d)" \
```

# Static methods and class methods

- You can invoke static methods even though no instance of that class has been created, although you can call them by using a class instance.
- To create a static method, use the @staticmethod decorator.

```
@staticmethod
def total_area():
    """Static method to total the areas of all Circles """
    total = 0
    for c in Circle.all_circles:
        total = total + c.area()
    return total

>>> circle.Circle.total_area()
31.415899999999997
```



# Static methods and class methods

- Class methods are similar to static methods in that they can be invoked before an object of the class has been instantiated or by using an instance of the class.
- But class methods are implicitly passed the class they belong to as their first parameter, so you can code them more simply, as here.

```
@classmethod
def total_area(cls):
    total = 0
    for c in cls.all_circles:
        total = total + c.area()
    return total
```

```
>>> circle_cm.Circle.total_area()
15.70795
>>> c2.radius = 3
>>> circle_cm.Circle.total_area()
31.415899999999997
```

# TRY THIS

- Write a class method similar to `total_area()` that returns the total circumference of all circles.

# Inheritance

- Parent and children relationship.
- Also known as superclass and subclass.

# Square class

- Properties x and y are the coordinates of the Square object to be drawn on canvas.

```
1  class Square:
2      def __init__(self, side=1, x=0, y=0):
3          self.side = side
4          self.x = x
5          self.y = y
6
```

# Circle class

- Circle class also have coordinates x and y.

```
7  class Circle:
8      def __init__(self, radius=1, x=0, y=0):
9          self.radius = radius
10         self.x = x
11         self.y = y
```

# Shape class

- Move all common properties to a more generic class.
- This class will be the parent (superclass).

```
15  class Shape:
16      def __init__(self, x, y):
17          self.x = x
18          self.y = y
19
```

# Square class (updated)

- Make Square class inherits from Shape class by pass the superclass's name in the bracket.
- The call superclass's init method to initialize it.

```
20  class Square(Shape):  
21      def __init__(self, side=1, x=0, y=0):  
22          super().__init__(x, y)  
23          self.side = side
```

# Circle class (updated)

- Do the same thing to Circle class.

```
25  ✓ class Circle(Shape):  
26  ✓     def __init__(self, r=1, x=0, y=0):  
27  |         super().__init__(x, y)  
28  |         self.radius = r  
--
```



# TRY THIS

- Rewrite the code for a Rectangle class to inherit from Shape. Because squares and rectangles are related, would it make sense to inherit one from the other? If so, which would be the base class, and which would inherit?
- How would you write the code to add an area() method for the Square class? Should the area method be moved into the base Shape class and inherited by circle, square, and rectangle? If so, what issues would result?

# Inheritance with class and instance variables

```
31  class P:
32      z = "Hello"
33      def set_p(self):
34          self.x = "Class P"
35      def print_p(self):
36          print(self.x)
37
38  class C(P):
39      def set_c(self):
40          self.x = "Class C"
41      def print_c(self):
42          print(self.x)
```

- Inheritance allows an instance to inherit attributes of the class.
- Instance variables are associated with object instances, and only one instance variable of a given name exists for a given instance.

# Private variables and methods

- A private variable or private method is one that can't be seen outside the methods of the class in which it's defined.
- Private variables and methods are useful for two reasons:
  - They enhance security and reliability by selectively denying access to important or delicate parts of an object's implementation,
  - and they prevent name clashes that can arise from the use of inheritance.
- A class may define a private variable and inherit from a class that defines a private variable of the same name, but this doesn't cause a problem, because the fact that the variables are private ensures that separate copies of them are kept.
- Any method or instance variable whose name begins—but doesn't end—with a double underscore (\_\_) is private; anything else isn't private.

# Mine class with private variables

```
1  class Mine:
2      def __init__(self):
3          self.x = 2
4          self.__y = 3
5      def print_y(self):
6          print(self.__y)
```

# TRY THIS

- Modify the Rectangle class's code to make the dimension variables private. What restriction will this modification impose on using the class?

# Using @property for more flexible instance variables

- Python allows you as the programmer to access instance variables directly, without the extra machinery of the getter and setter methods often used in Java and other object-oriented languages.
- This lack of getters and setters makes writing Python classes cleaner and easier, but in some situations, using getter and setter methods can be handy.
- The answer is to use a property.
- A property combines the ability to pass access to an instance variable through methods like getters and setters and the straightforward access to instance variables through dot notation.

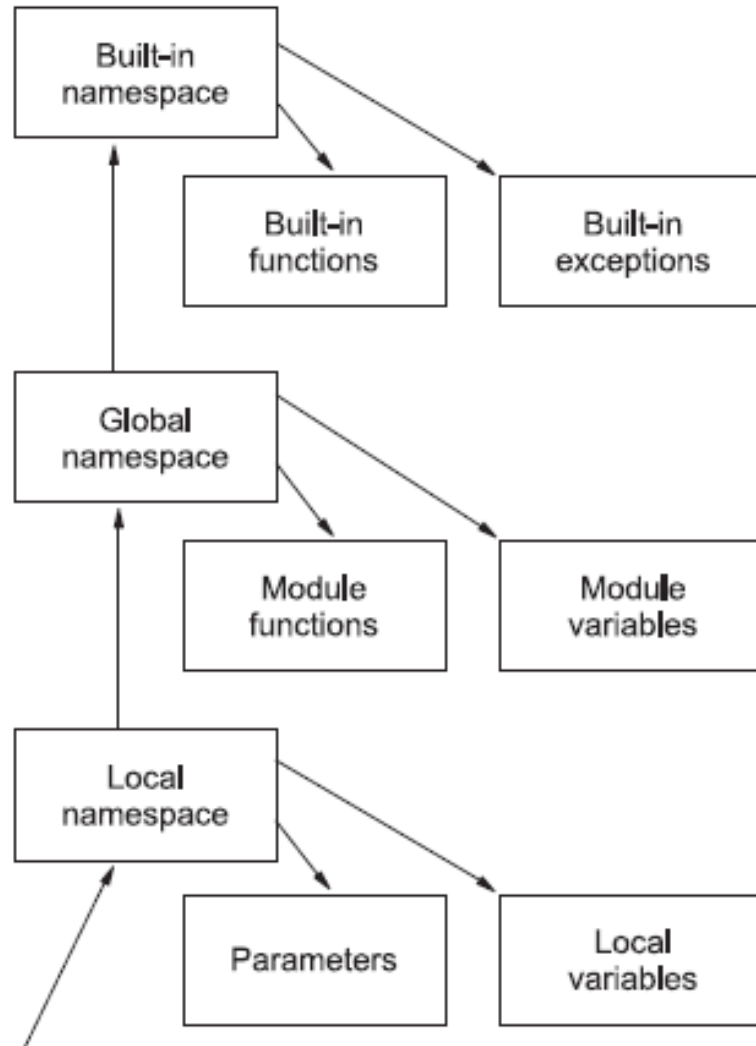
# Using @property for more flexible instance variables

- To create a property, you use the property decorator with a method that has the property's name.

```
1  class Temperature:
2      def __init__(self):
3          self._temp_fahr = 0
4
5      @property
6      def temp(self):
7          return (self._temp_fahr - 32) * 5 / 9
8
9      @temp.setter
10     def temp(self, new_temp):
11         self._temp_fahr = new_temp * 9 / 5 + 32
```

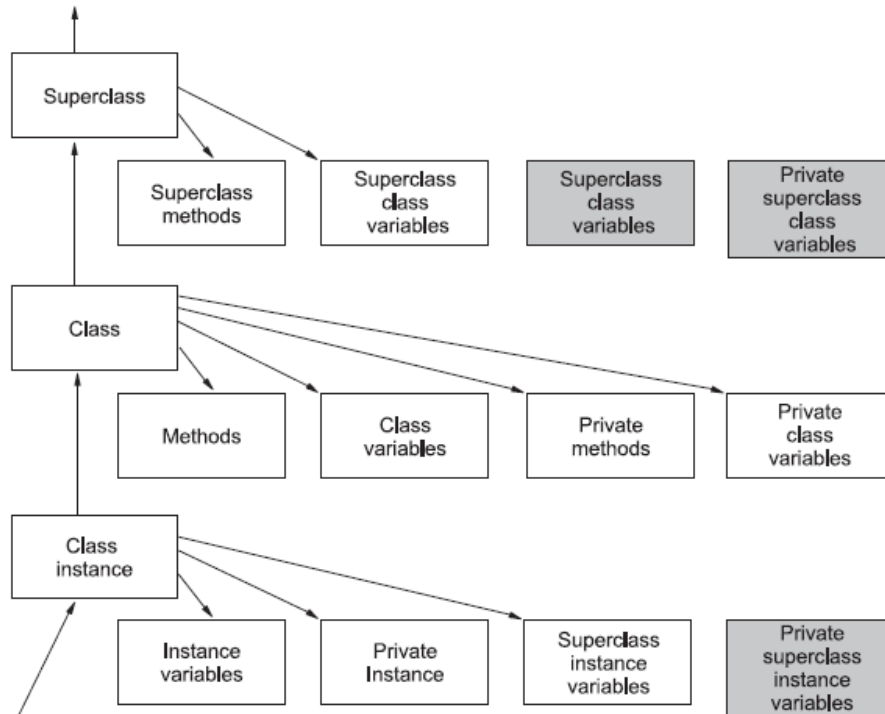
# Scoping rules and namespaces for class instances

- When you're in a method of a class, you have direct access to the local namespace (parameters and variables declared in the method), the global namespace (functions and variables declared at the module level), and the built-in namespace (built-in functions and built-in exceptions). These three namespaces are searched in the following order: local, global, and built-in.





# Scoping rules and namespaces for class instances



- You also have access through the self variable to the instance's namespace (instance variables, private instance variables, and superclass instance variables), its class's namespace (methods, class variables, private methods, and private class variables), and its superclass's namespace (superclass methods and superclass class variables).
- These three namespaces are searched in the order instance, class, and then superclass.

# Destructors and memory management

- You've already seen class initializers (the `__init__` methods).
- A destructor can be defined for a class as well.
- Python provides automatic memory management through a reference-counting mechanism.
- That is, it keeps track of the number of references to your instance; when this number reaches zero, the memory used by your instance is reclaimed, and any Python objects referenced by your instance have their reference counts decremented by one.
- You almost never need to define a destructor

# Multiple inheritance

- Python places no such restrictions on multiple inheritance.
- A class can inherit from any number of parent classes in the same way that it can inherit from a single parent class.
- In the simplest case, none of the involved classes, including those inherited indirectly through a parent class, contains instance variables or methods of the same name.

# Lab

## HTML Classes

# Summary

- Defining a class in effect creates a new data type.
- `__init__` is used to initialize data when a new instance of a class is created, but it isn't a constructor.
- The `self` parameter refers to the current instance of the class and is passed as the first parameter to methods of a class.
- Static methods can be called without creating an instance of the class, so they don't receive a `self` parameter.
- Class methods are passed a `cls` parameter, which is a reference to the class, instead of `self`.

# Summary

- All Python methods are virtual. That is, if a method isn't overridden in the subclass or private to the superclass, it's accessible by all subclasses.
- Class variables are inherited from superclasses unless they begin with two underscores (\_\_), in which case they're private and can't be seen by subclasses. Methods can be made private in the same way.
- Properties let you have attributes with defined getter and setter methods, but they still behave like plain instance attributes.
- Python allows multiple inheritance, which is often used with mixin classes.