Chapter 14

How to define and use your own classes

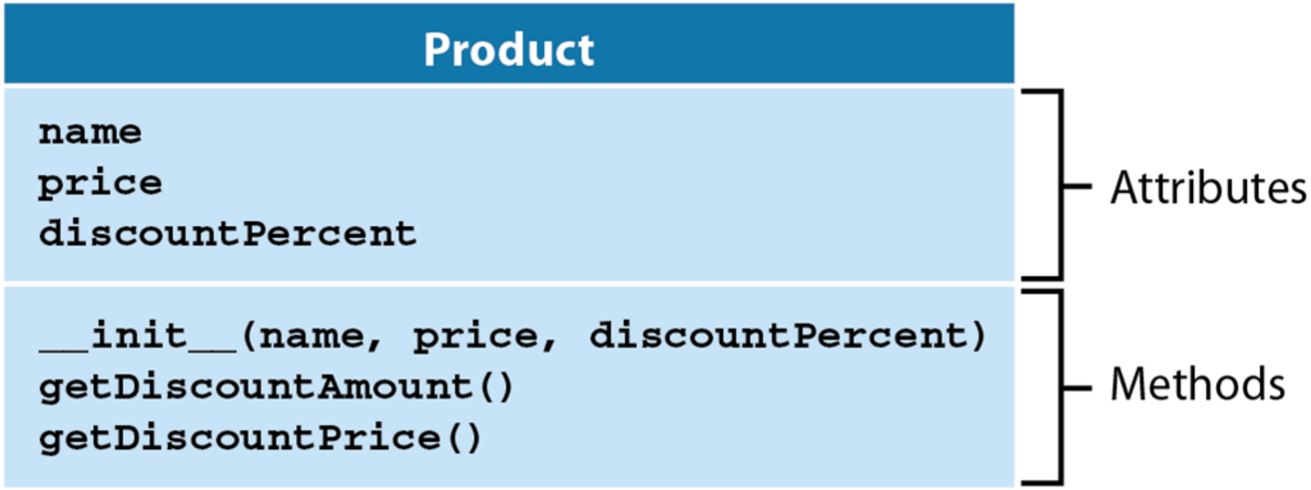
Applied objectives

1. Code a data class that has attributes and methods.
2. Code the constructor for a class that has attributes.
3. Import a class, create objects from it, access the attributes of the objects, and call the methods of the objects.
4. Use object composition to combine simple objects into more complex data structures.
5. Use encapsulation to hide the data attributes of an object, and use methods or properties to access the hidden attributes.

Knowledge objectives

1. Describe a UML class diagram.
2. Describe the relationship between a class and an object.
3. In general terms, describe the identity, state, and behavior of an object.
4. In general terms, describe the way Python code is used to define a data class with attributes and methods.
5. Explain when you may need to code an __init__() or __post_init__() method.
6. In general terms, describe the way Python code is used to create an object from a class.
7. Describe the concept of object composition.
8. Describe the concept of encapsulation.
9. Distinguish between public and private attributes.
10. Describe the use of getter and setter methods.

A diagram of the Product class



Product

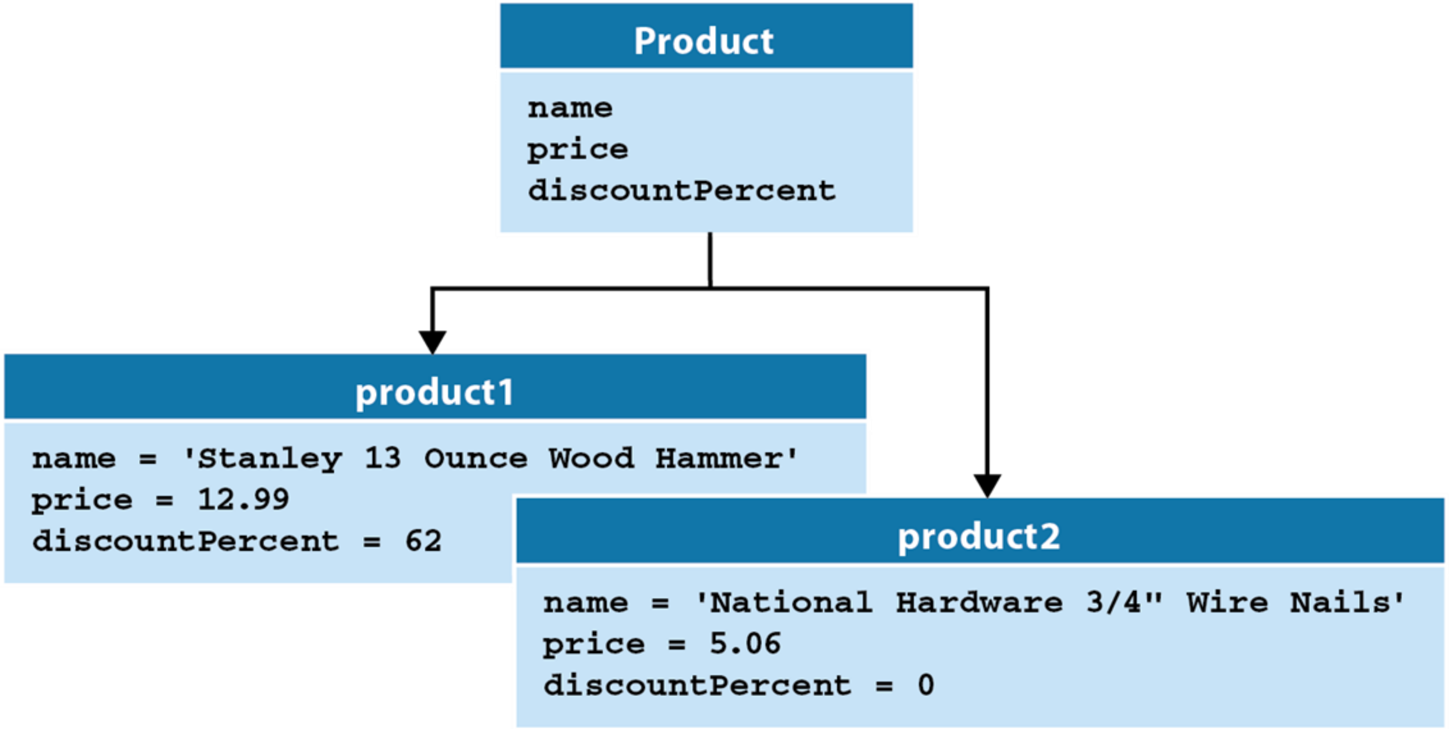
```
name
price
discountPercent
```

Attributes

```
__init__(name, price, discountPercent)
getDiscountAmount()
getDiscountPrice()
```

Methods

The relationship between a class and its objects



```
name
price
discountPercent
```

```
name = 'Stanley 13 Ounce Wood Hammer'
price = 12.99
discountPercent = 62
```

```
name = 'National Hardware 3/4" Wire Nails'
price = 5.06
discountPercent = 0
```

UML diagramming notes

- *UML* (*Unified Modeling Language*) is the industry standard used to describe the classes and objects of an object-oriented application.
- A UML *class diagram* describes the attributes and methods of one or more classes.

The Product class in the module named objects

```
from dataclasses import dataclass

# a class with three attributes and two methods
@dataclass                           # dataclass decorator
class Product:
    name:str                         # attribute 1
    price:float                      # attribute 2
    discountPercent:int              # attribute 3

    # a method that uses two attributes
    def getDiscountAmount(self):
        return self.price * self.discountPercent / 100

    # a method that calls another method
    def getDiscountPrice(self):
        return self.price - self.getDiscountAmount()
```

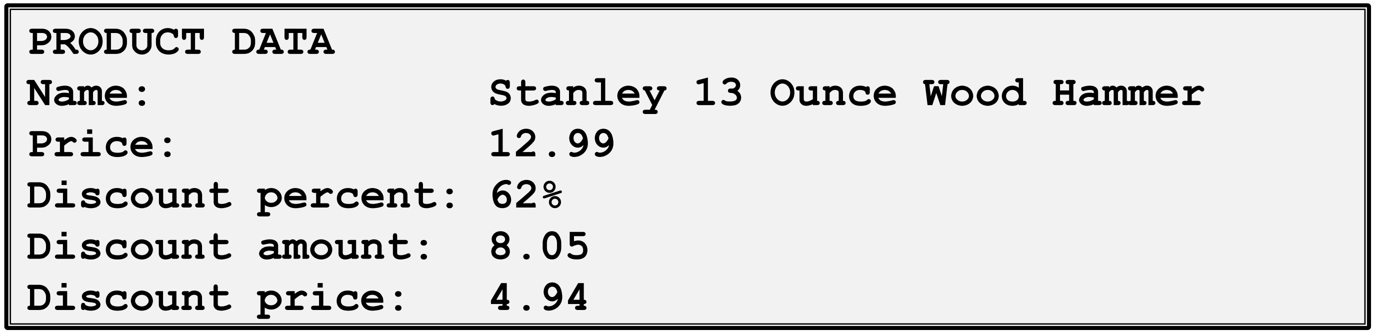
A script that creates and uses a Product object

```
from objects import Product

# create two product objects
product1 = Product("Stanley 13 Ounce Wood Hammer", 12.99, 62)
product2 = Product('National Hardware 3/4" Wire Nails', 5.06, 0)

# print data for product1 to console
print("PRODUCT DATA")
print(f"Name:             {product1.name}")
print(f"Price:            {product1.price:.2f}")
print(f"Discount percent: {product1.discountPercent:d}%")
print(f"Discount amount:  {product1.getDiscountAmount():.2f}")
print(f"Discount price:   {product1.getDiscountPrice():.2f}")
```

The console display when the script is run



```
PRODUCT DATA
Name:                  Stanley 13 Ounce Wood Hammer
Price:                 12.99
Discount percent:      62%
Discount amount:       8.05
Discount price:        4.94
```

How to import a class

The syntax

```
from module_name import ClassName1[, ClassName2]...
```

Import the Product class from the objects module

```
from objects import Product
```

How to create an object

The syntax

```
objectName = ClassName([parameters])
```

Create two Product objects

```
product1 = Product('Stanley 13 Ounce Wood Hammer',
                   12.99, 62)
product2 = Product('National Hardware 3/4" Wire Nails',
                   5.06, 0)
```

How to access the attributes of an object

The syntax

```
objectName.attributeName
```

Set an attribute

```
product1.discountPercent = 40
```

Get an attribute

```
percent = product1.discountPercent     # percent = 40
```

How to call the methods of an object

The syntax

```
objectName.methodName([parameters])
```

Call the getDiscountAmount() method

```
discount = product1.getDiscountAmount()
```

Call the getDiscountPrice() method

```
salePrice = product1.getDiscountPrice()
```

The syntax of a class with attributes

With a dataclass decorator

```
@dataclass                                 # dataclass decorator
class ClassName:
    attrName1:type [= default_value]       # first attribute
    attrName2:type [= default_value]       # second attribute
    ...
```

With a constructor

```
class ClassName:
    def __init__(self[, parameters]):      # the constructor
        self.attrName1 = attrValue1        # first attribute
        self.attrName2 = attrValue2        # second attribute
        ...
```

A data class that has three attributes with default values (3.7 and later)

```
# import dataclass module
from dataclasses import dataclass

@dataclass
class Product:
    name:str =  ""
    price:float = 0.0
    discountPercent:int = 0
```

A class that has three attributes with default values (3.6 and earlier)

```
class Product:
    def __init__(self, name = "", price = 0.0,
                 discount_percent = 0):
        self.name = name
        self.price = price
        self.discountPercent = discount_percent
```

Code that uses the constructor to create an object

```
product = Product()
```

Code that supplies all three parameters

```
product = Product("Stanley 13 Ounce Wood Hammer", 12.99, 62)
```

Code that supplies just two parameters

```
product = Product(name="Stanley 13 Ounce Wood Hammer",
                  price=12.99)
```

The syntax for coding a method

```
def methodName(self[, parameters]):
    statements
```

A method that returns a value

```
def getDiscountAmount(self):
    discountAmount = self.price *
                     self.discountPercent / 100
    return discountAmount
```

Code that calls this method

```
discountAmount = product.getDiscountAmount()
```

A more concise way to code this method

```
def getDiscountAmount(self):
    return self.price * self.discountPercent / 100
```

Code that calls this method

```
discountAmount = product.getDiscountAmount()
```

A method that calls another method of the class

```
def getDiscountPrice(self):
    return self.price - self.getDiscountAmount()
```

Code that calls this method

```
discountPrice = product.getDiscountPrice()
```

A method that accepts a parameter

```
def getPriceStr(self, country):
    priceStr = f"{self.price:.2f}"
    if country == "US":
        priceStr += " USD"
    elif country == "DE":
        priceStr = priceStr += " EUR"
    return priceStr
```

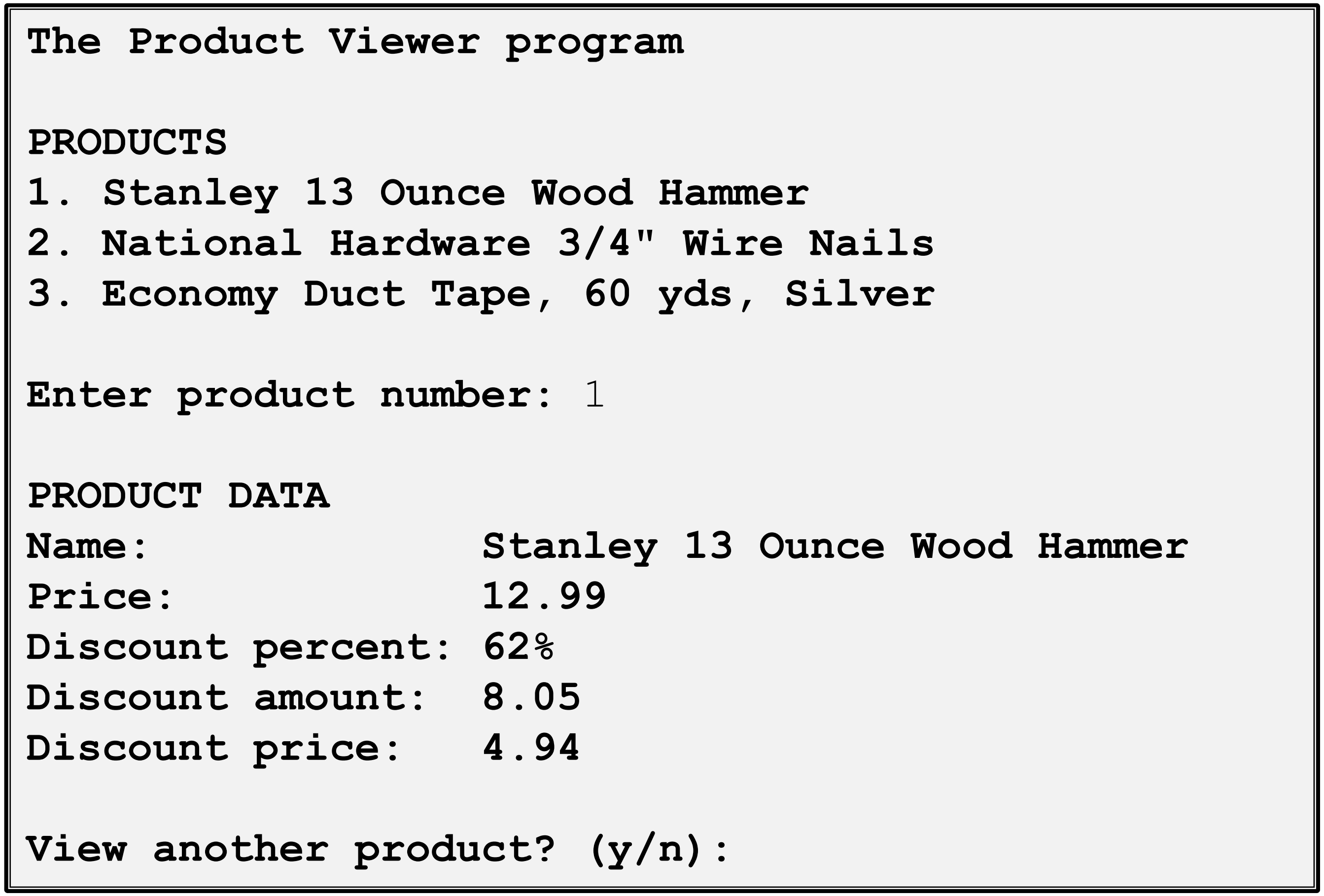
Code that calls this method

```
print(f"Price: {product.getPriceStr('US')}")
```

The error that’s displayed if you forget to code the self parameter

```
TypeError: getPriceStr() takes 1 positional argument
but 2 were given
```

The console for the Product Viewer



```
The Product Viewer program

PRODUCTS
1. Stanley 13 Ounce Wood Hammer
2. National Hardware 3/4" Wire Nails
3. Economy Duct Tape, 60 yds, Silver

Enter product number: 1

PRODUCT DATA
Name:             Stanley 13 Ounce Wood Hammer
Price:            12.99
Discount percent: 62%
Discount amount:  8.05
Discount price:   4.94

View another product? (y/n):
```

The objects module

```
from dataclasses import dataclass

@dataclass
class Product:
    name:str = ""
    price:float = 0.0
    discountPercent:float = 0

    def getDiscountAmount(self):
        return self.price * self.discountPercent / 100

    def getDiscountPrice(self):
        return self.price - self.getDiscountAmount()
```

The product_viewer module (part 1)

```
from objects import Product

def show_products(products):
    print("PRODUCTS")
    for i, product in enumerate(products, start=1):
        print(f"{i}. {product.name}")
    print()

def show_product(product):
    w = 18
    print(f"{'Name:':{w}}{product.name}")
    print(f"{'Price:':{w}}{product.price:.2f}")
    print(f"{'Discount percent:':{w}}{
        product.discountPercent:d}%")
    print(f"{'Discount amount:':{w}}{
        product.getDiscountAmount():.2f}")
    print(f"{'Discount price:':{w}}{
        product.getDiscountPrice():.2f}")
    print()
```

The product_viewer module (part 2)

```
def get_products():
    # return a tuple of Product objects
    return (Product("Stanley 13 Ounce Wood Hammer", 12.99, 62),
            Product('National Hardware 3/4" Wire Nails', 5.06, 0),
            Product("Economy Duct Tape, 60 yds, Silver", 7.24, 0))

def get_product(products):
    while True:
        try:
            number = int(input("Enter product number: "))
            if number < 1 or number > len(products):
                print("Product number out of range. "
                      "Please try again.")
            else:
                return products[number-1]
        except ValueError:
            print("Invalid number. Please try again.")
        print()
```

The product_viewer module (part 3)

```
def main():
    print("The Product Viewer program")
    print()

    products = get_products()
    show_products(products)

    choice = "y"
    while choice.lower() == "y":
        product = get_product(products)
        show_product(product)

        choice = input("View another product? (y/n): ")
        print()

    print("Bye!")

if __name__ == "__main__":
    main()
```