香 港 中 文 大 學（深 圳）
The Chinese University of Hong Kong, Shenzhen

Introduction to Computer Science: Programming Methodology

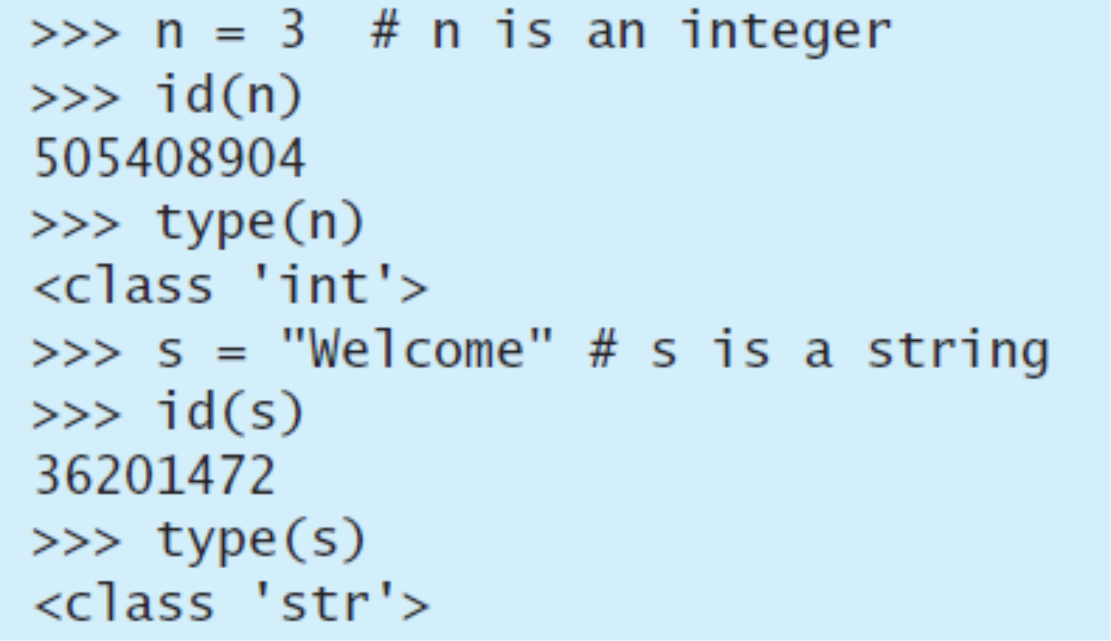
Lecture 6 Object Oriented Programming

Tongxin Li

School of Data Science

Object

- In Python, everything is an object (number, string, etc)
- You can use the id() function and type() function to get information about an object



```
>>> n = 3  # n is an integer
>>> id(n)
505408904
>>> type(n)
<class 'int'>
>>> s = "Welcome" # s is a string
>>> id(s)
36201472
>>> type(s)
<class 'str'>
```

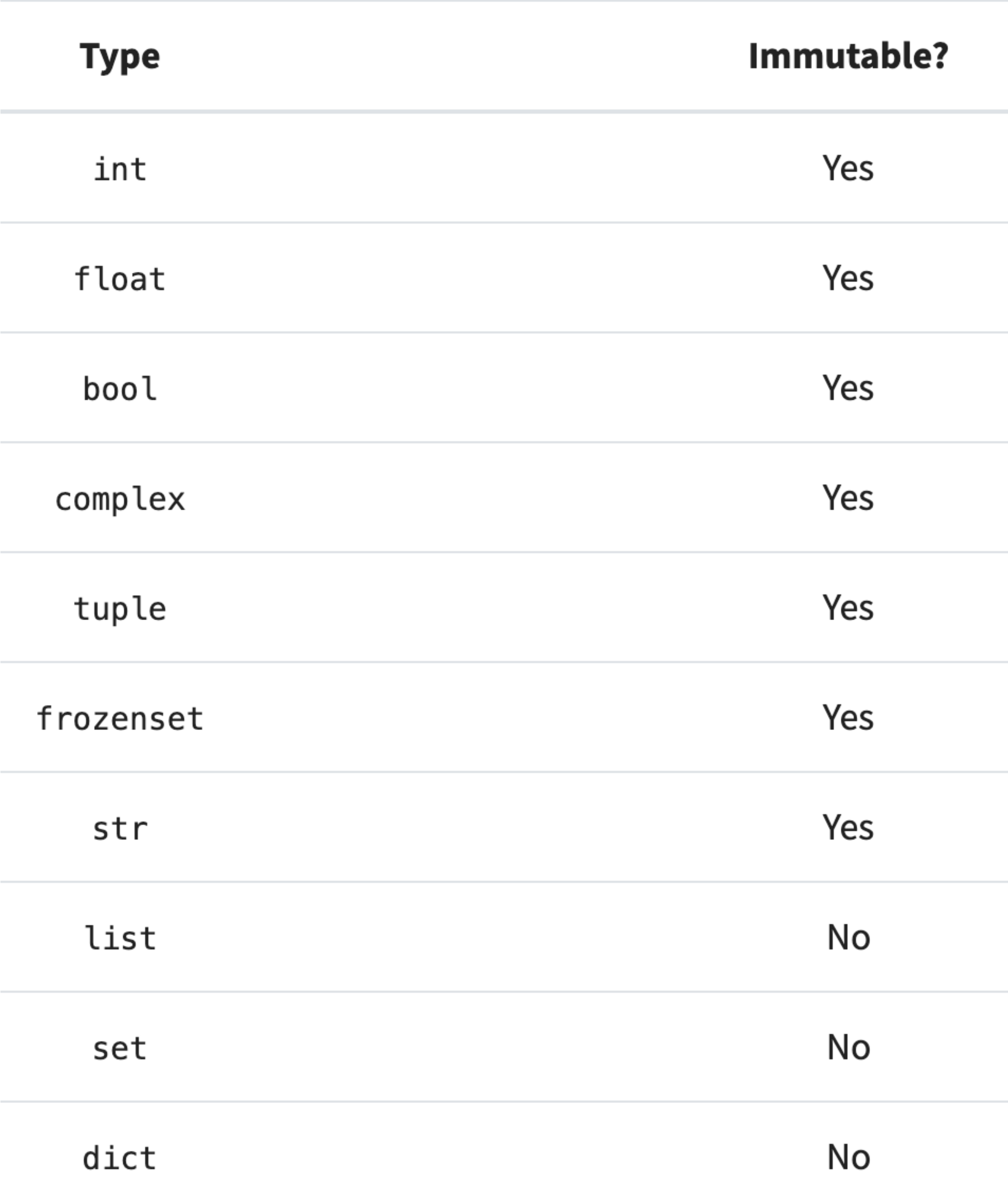
ID and type

- The id of an object is automatically assigned a unique integer by Python when the program is executed

- The id for the object will not be changed during the execution of the program

- The type for the object is determined by Python according to the value of the object

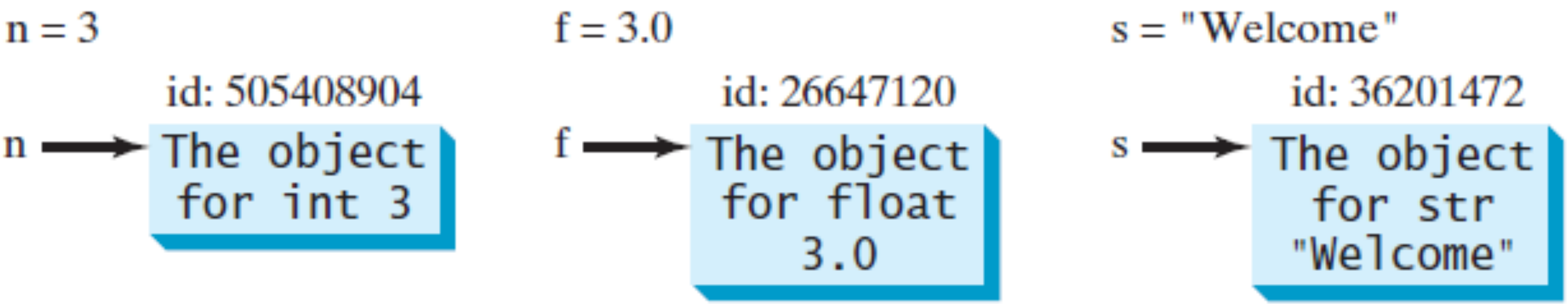
ID and type



Type	Immutable?
`int`	Yes
`float`	Yes
`bool`	Yes
`complex`	Yes
`tuple`	Yes
`frozenset`	Yes
`str`	Yes
`list`	No
`set`	No
`dict`	No

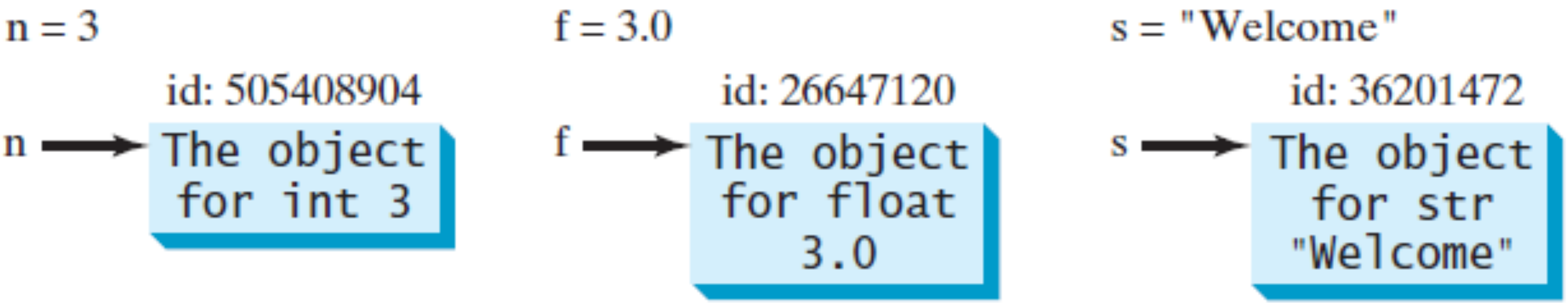
Variable is actually only a reference

- A variable in Python is actually a reference to an object.



Variable is actually only a reference

- A variable in Python is actually a reference to an object.



n = 3

id: 505408904

n

The object for int 3

f = 3.0

id: 26647120

f

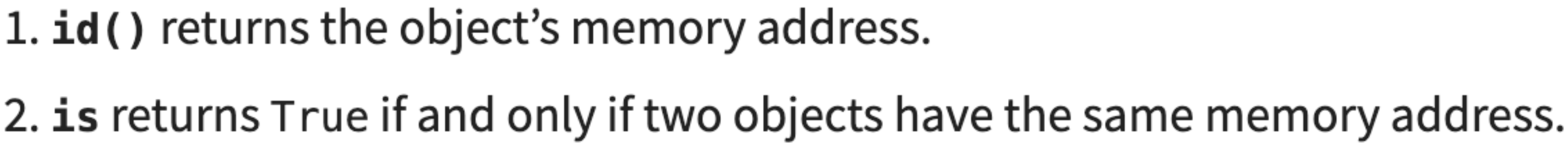
The object for float 3.0

s = "Welcome"

id: 36201472

s

The object for str "Welcome"

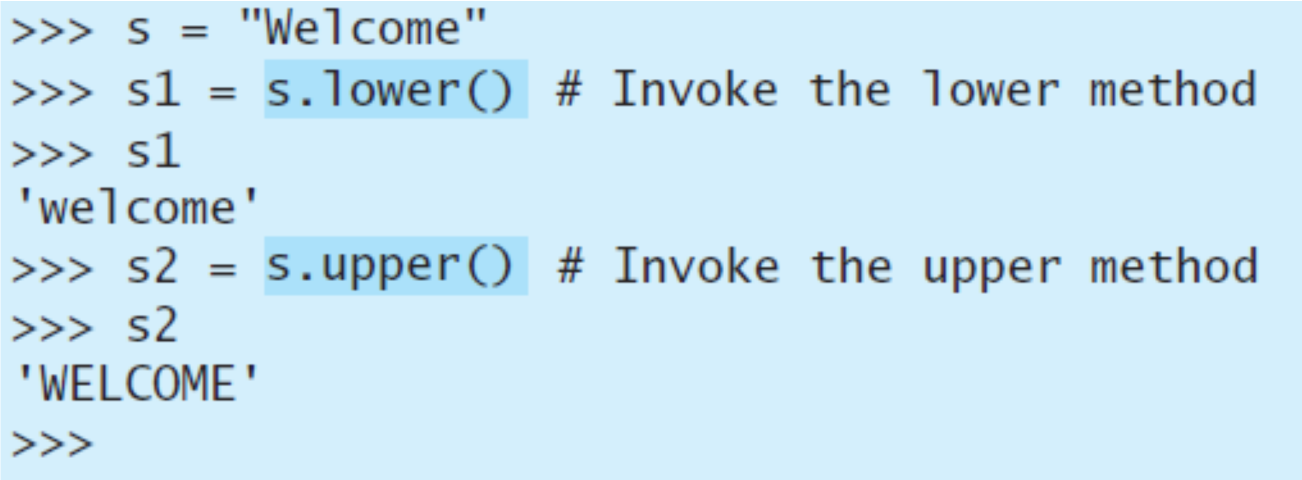


1. **id()** returns the object's memory address.

2. **is** returns `True` if and only if two objects have the same memory address.

Methods

- You can perform operations on an object
- The operations are defined using functions
- The functions for the objects are called methods in Python
- Methods can only be invoked from a specific object



```
>>> s = "Welcome"
>>> s1 = s.lower() # Invoke the lower method
>>> s1
'welcome'
>>> s2 = s.upper() # Invoke the upper method
>>> s2
'WELCOME'
>>>
```

Why we need object-oriented programming?

- Writing a real software is a complicated process

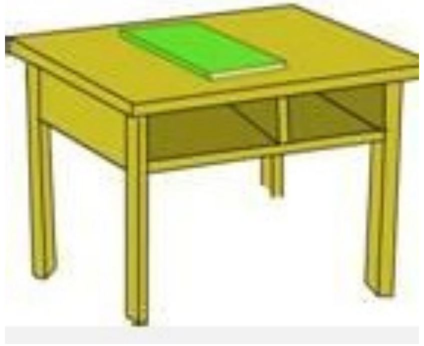
- A sub-field in computer science called software engineering is invented to help with the development of large-scale software systems

- People are always trying to invent new ways of writing programs so that software development can be more efficient – structural programming, OO programming, service-oriented architecture, etc

- Object oriented programing allows us to write program in a way that naturally match the problem that we are trying to solve

Object

- An object represents an entity in the real world that can be distinctly identified.
- Examples: a student, a desk, a circle, a button, and even a loan
- An object has a unique identity, state, and behaviours



Key elements of an object

- An object’s identity is like a person’s ID. Python automatically assigns each object a unique id for identifying the object at runtime.

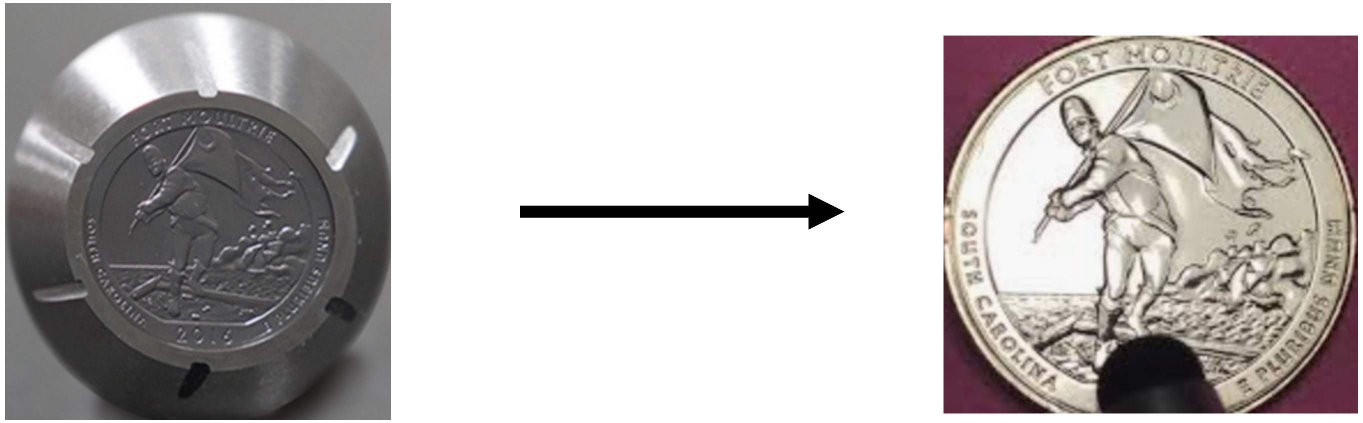
- An object’s state (also known as its properties or attributes) is represented by variables, called data fields.

- Python uses methods to define an object’s behavior (also known as its actions). Recall that methods are defined as functions. You make an object perform an action by invoking a method on that object.

How to create an object?

How to create an object?

- In Python, we use a template, called a **class** to create objects



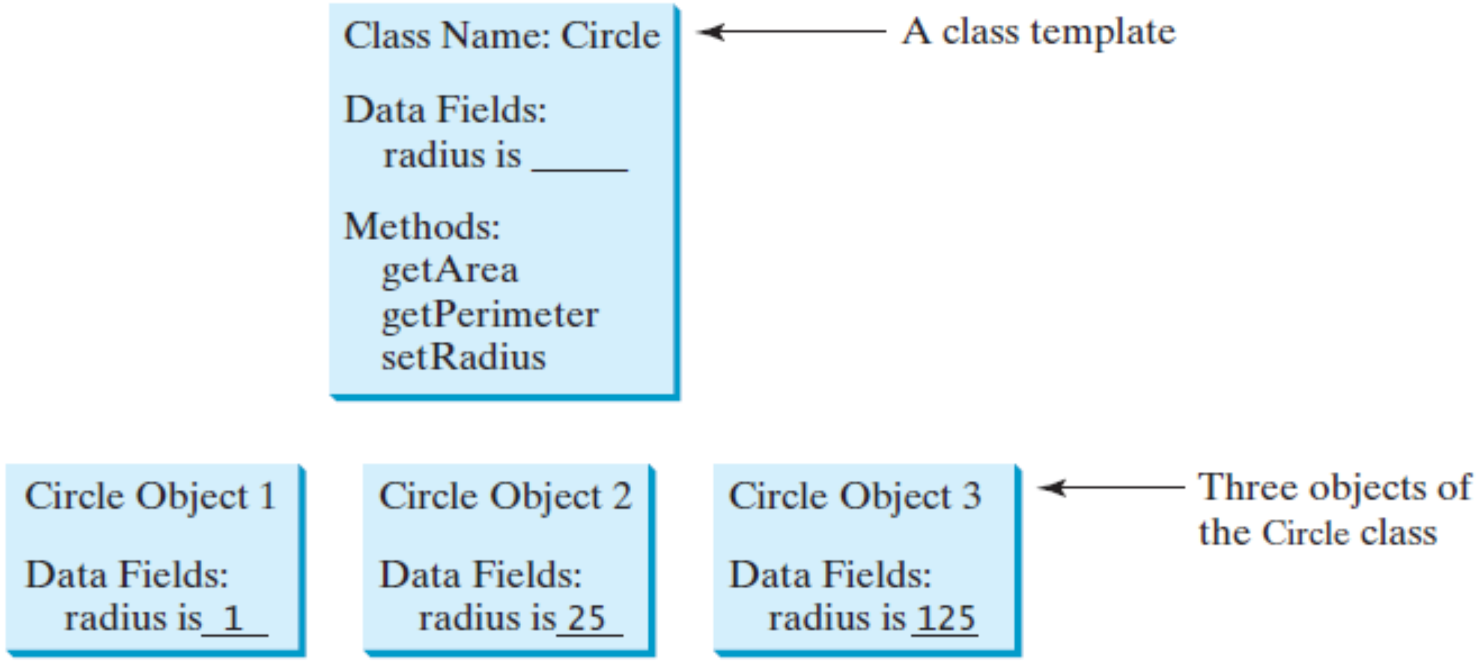
Example: Use hub to make coins

Class

- Objects of the same kind are defined by using a common class
- The relationship between Classes and objects is analogous to that between an apple-pie recipe and apple pies
- A Python class uses variables to store data fields and defines methods to perform actions
- A class is a contract—also sometimes called a template or blueprint

Object v.s. class

- An object is an instance of a class, and you can create many instances of a class
- Creating an instance of a class is referred to as instantiation
- The terms object and instance are often used interchangeably



Class Name: Circle

Data Fields:
radius is ______

Methods:
getArea
getPerimeter
setRadius

A class template

Circle Object 1

Data Fields:
radius is 1

Circle Object 2

Data Fields:
radius is 25

Circle Object 3

Data Fields:
radius is 125

Three objects of the Circle class

Class Example

- Class name: Human
- Data fields: Height, body weight, IQ, EQ, education level …
- Methods:

  Eat()

  Sleep()

  Marry()

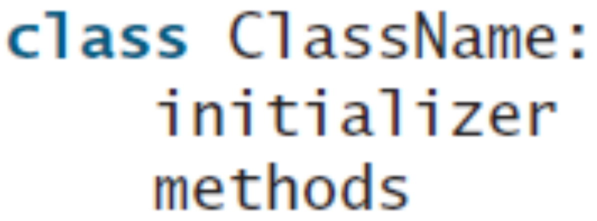
  Work()

  ……

Define class

- Python uses the following syntax to define a class

- a class provides a special method, __init__(). This method, known as an initializer, is invoked to initialize a new object’s state when it is created

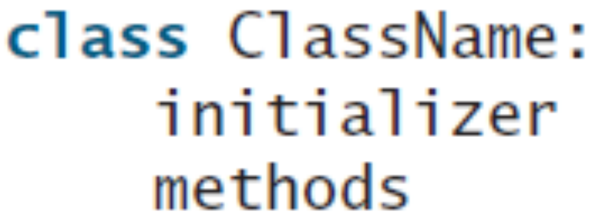


```
class ClassName:
    initializer
    methods
```

Define class

- Python uses the following syntax to define a class

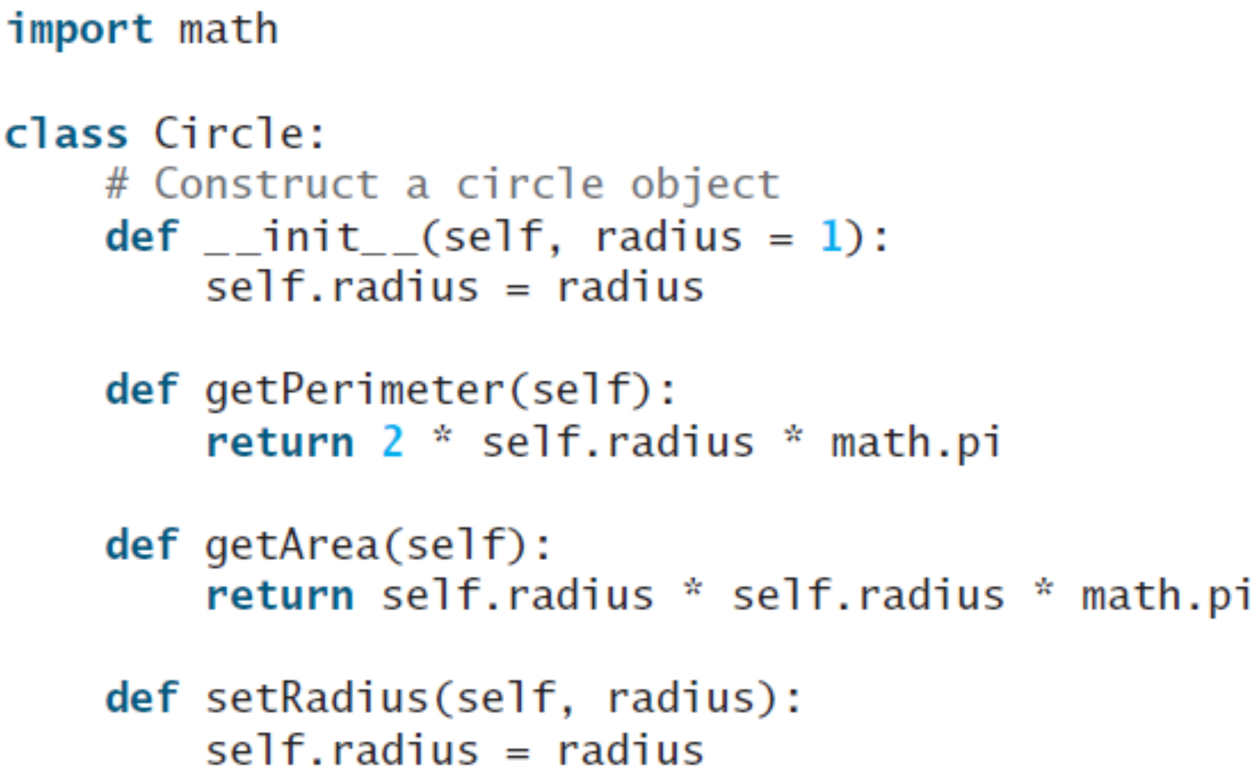
- a class provides a special method, __init__(). This method, known as an initializer, is invoked to initialize a new object's state when it is created



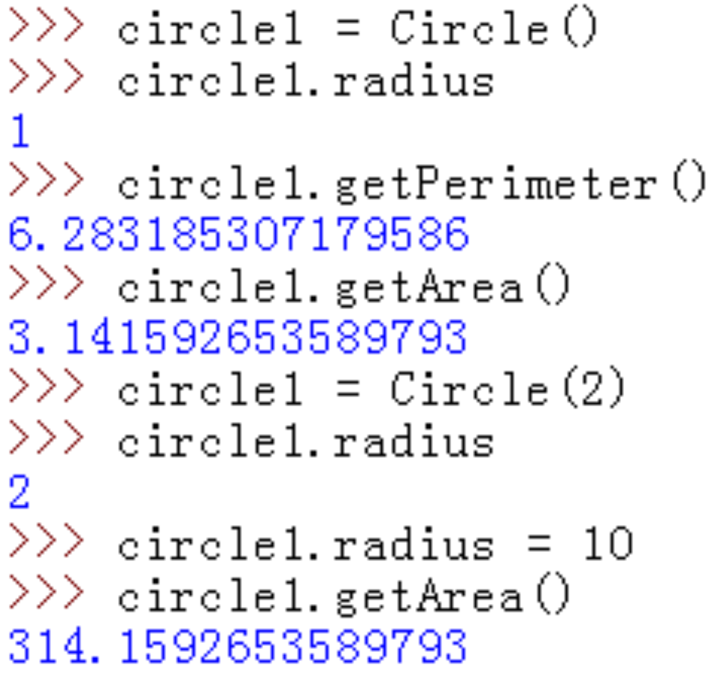
```
class ClassName:
    initializer
    methods
```

__init__() is a dunder method. Dunder methods have specific roles and are used to interact with Python's built-in behaviors and protocols.

Example



```
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
```



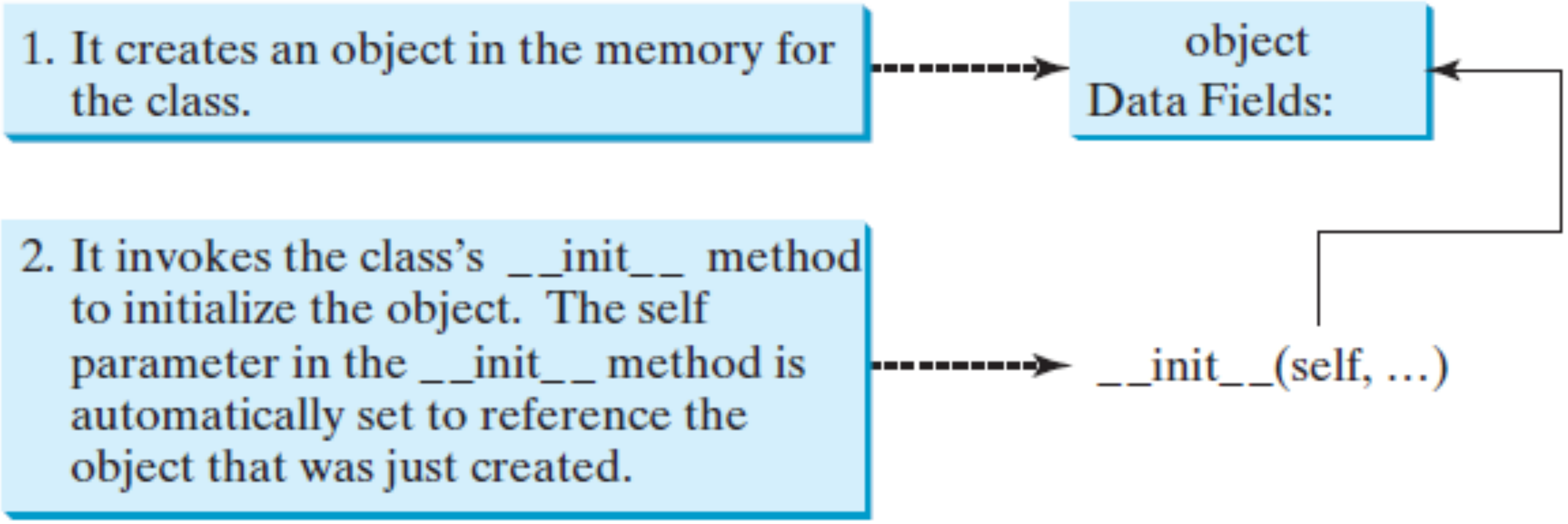
```
>>> circle1 = Circle()
>>> circle1.radius
1
>>> circle1.getPerimeter()
6.283185307179586
>>> circle1.getArea()
3.141592653589793
>>> circle1 = Circle(2)
>>> circle1.radius
2
>>> circle1.radius = 10
>>> circle1.getArea()
314.1592653589793
```

Constructing objects

- Once a class is defined, you can create objects from the class with a constructor. The constructor does two things:

✓It creates an object in the memory for the class

✓It invokes the class's __init__() method to initialize the object

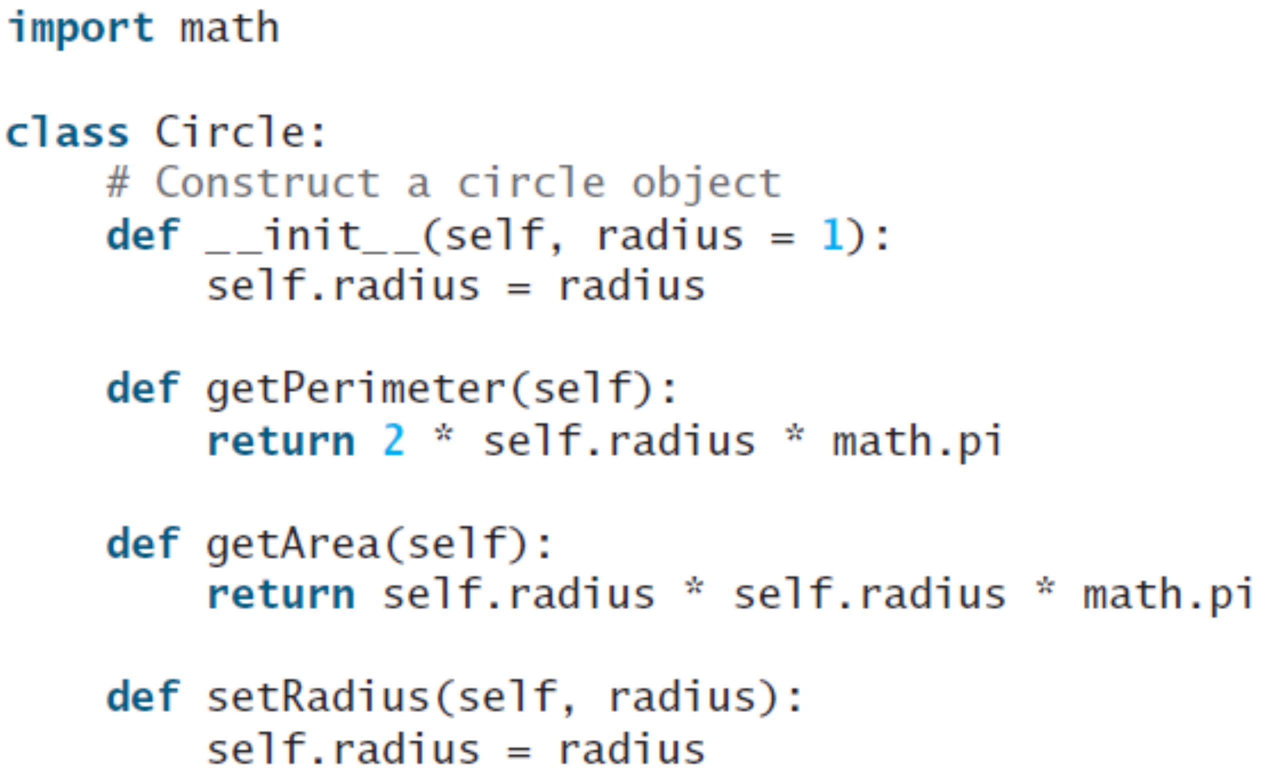


1. It creates an object in the memory for the class.

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.

Self

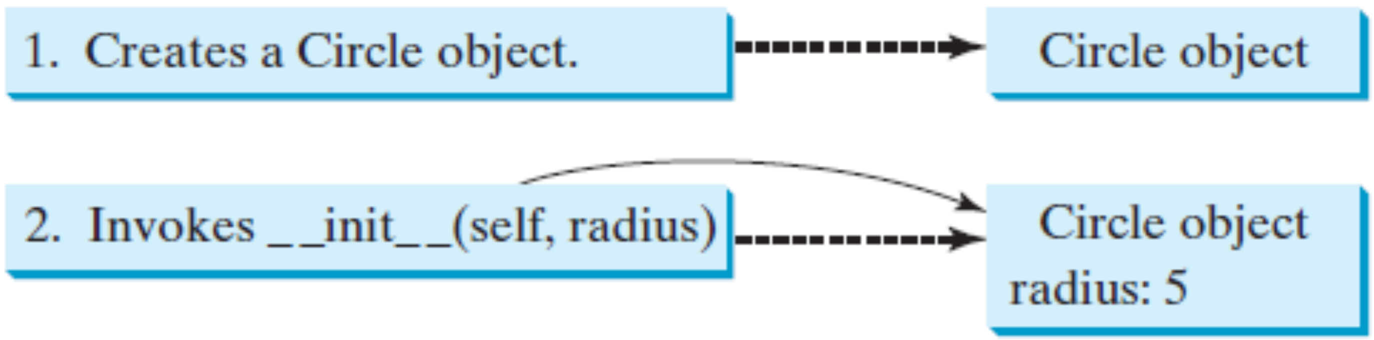
- All methods, including the initializer, have the first parameter self
- This parameter refers to the object that invokes the method.
- The self parameter in the __init__() method is automatically set to reference the object that was just created



```
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
```

Constructor arguments

- The arguments of the constructor match the parameters in the __init__() method without self


1. Creates a Circle object.
Circle object
2. Invokes __init__(self, radius)
Circle object
radius: 5

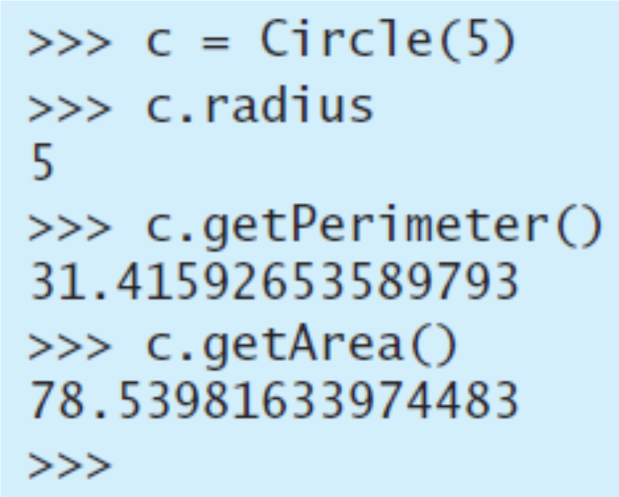
- The initializer in the Circle class has a default radius value, then the constructor without arguments will assign the default values to data fields

Accessing member of objects

- Data fields are also called instance variables, because each object (instance) has a specific value for a data field

- Methods are also called instance methods, because a method which is invoked by an object (instance) will perform actions based on the data fields of that object

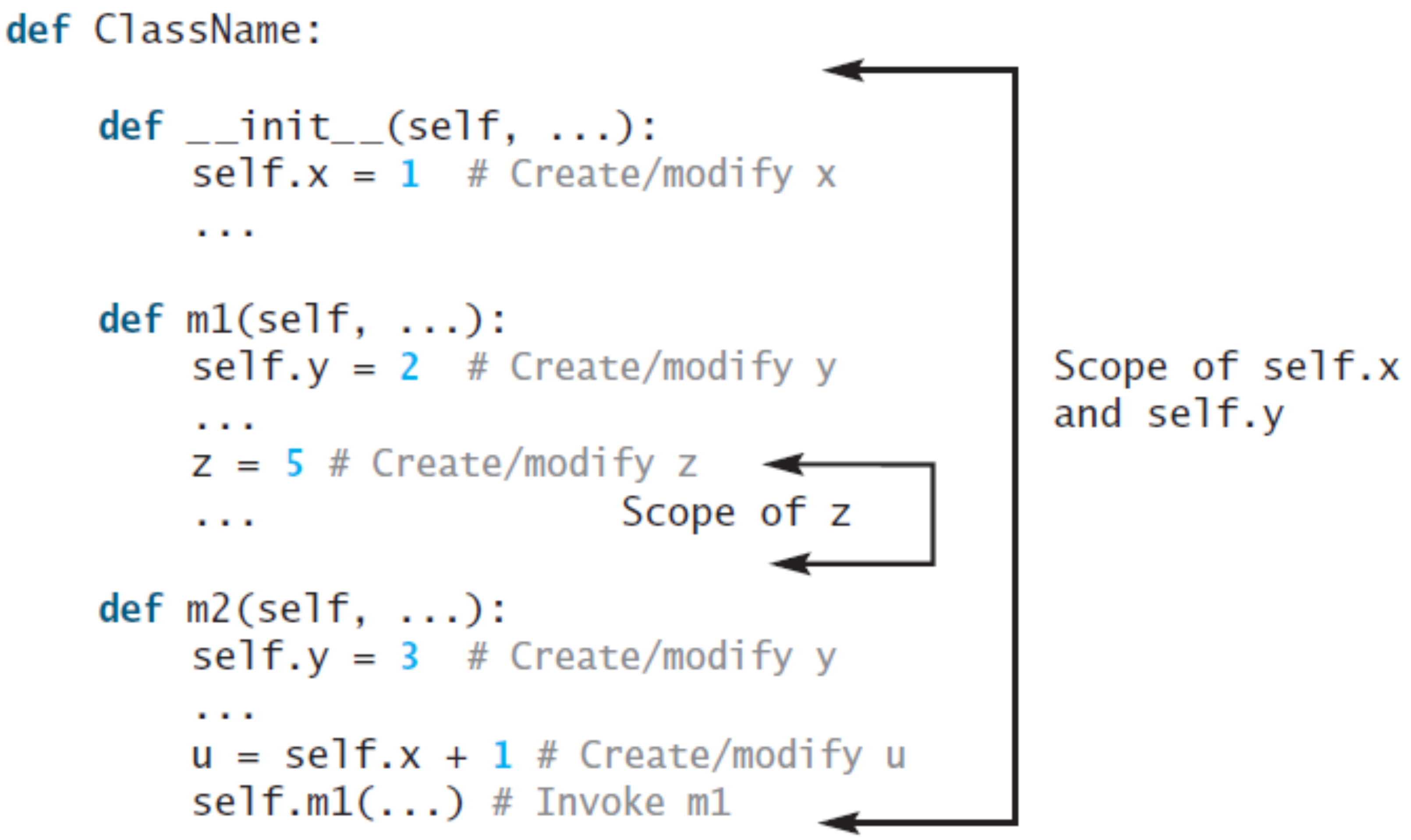
- You can access the object's data fields and invoke its methods by using the dot operator (.), also known as the object member access operator (Does this look familiar?)



```
>>> c = Circle(5)
>>> c.radius
5
>>> c.getPerimeter()
31.41592653589793
>>> c.getArea()
78.53981633974483
>>>
```

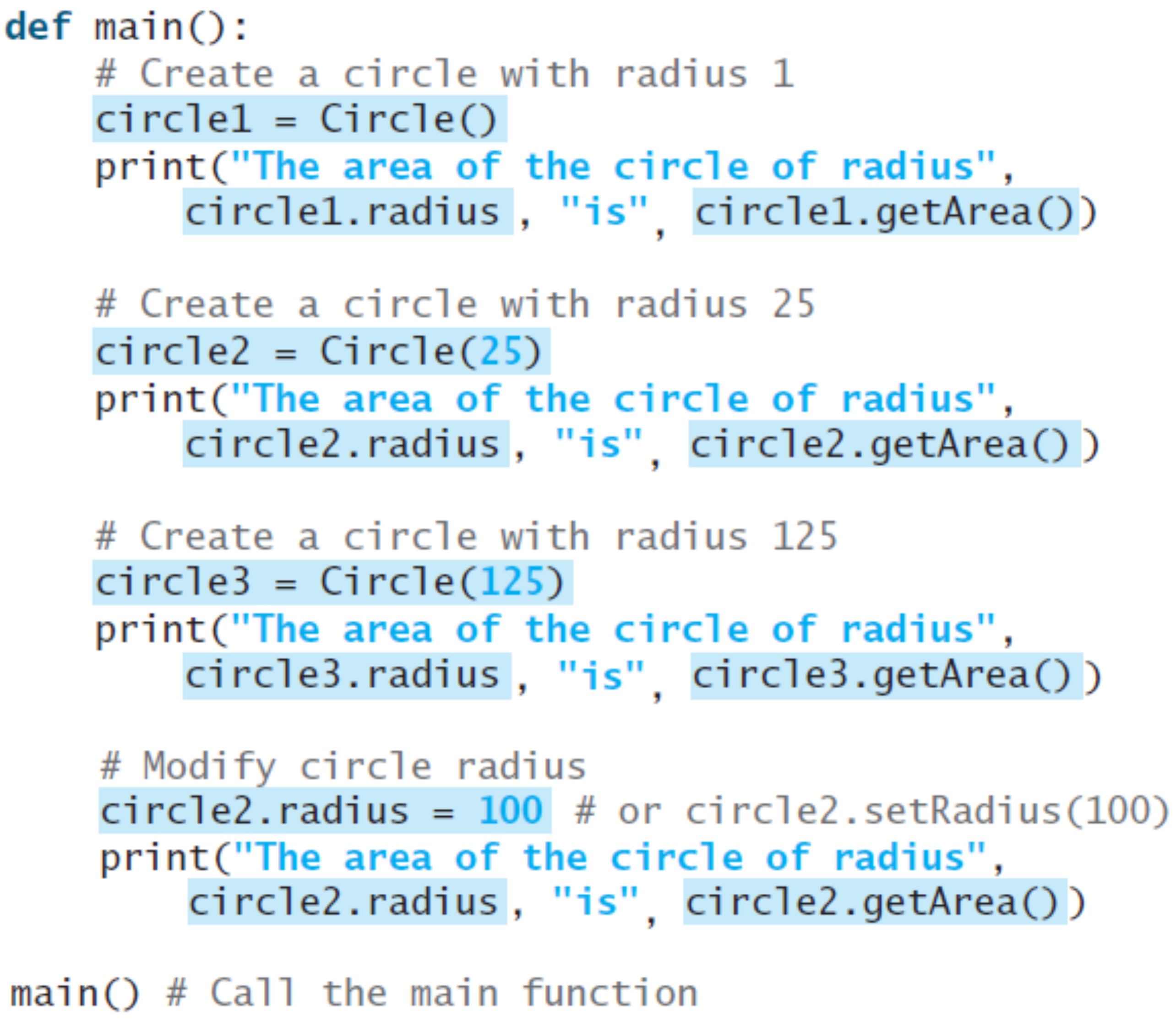
Scope of self

- The scope of an instance variable is the entire class once it is created
- You can also create local variables in a method
- The scope of a local variable is within the method



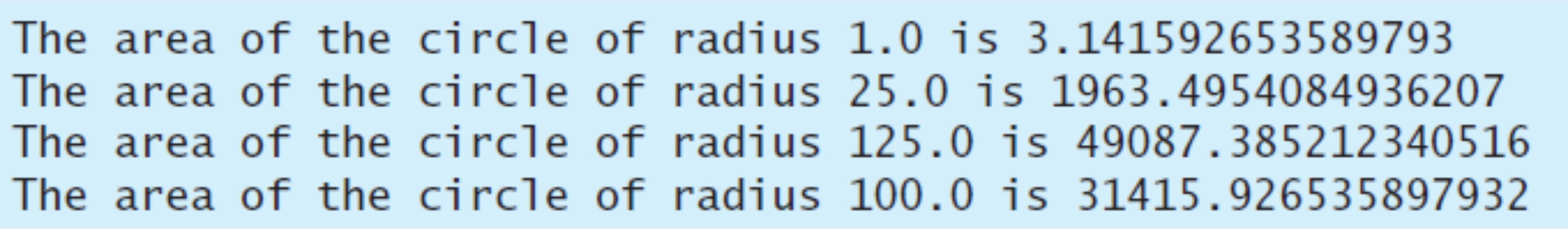
```
def ClassName:

    def __init__(self, ...):
        self.x = 1  # Create/modify x
        ...

    def m1(self, ...):
        self.y = 2  # Create/modify y
        ...
        z = 5 # Create/modify z
        ...

    def m2(self, ...):
        self.y = 3  # Create/modify y
        ...
        u = self.x + 1 # Create/modify u
        self.m1(...) # Invoke m1
```

Example: Use the objects we’ve just defined



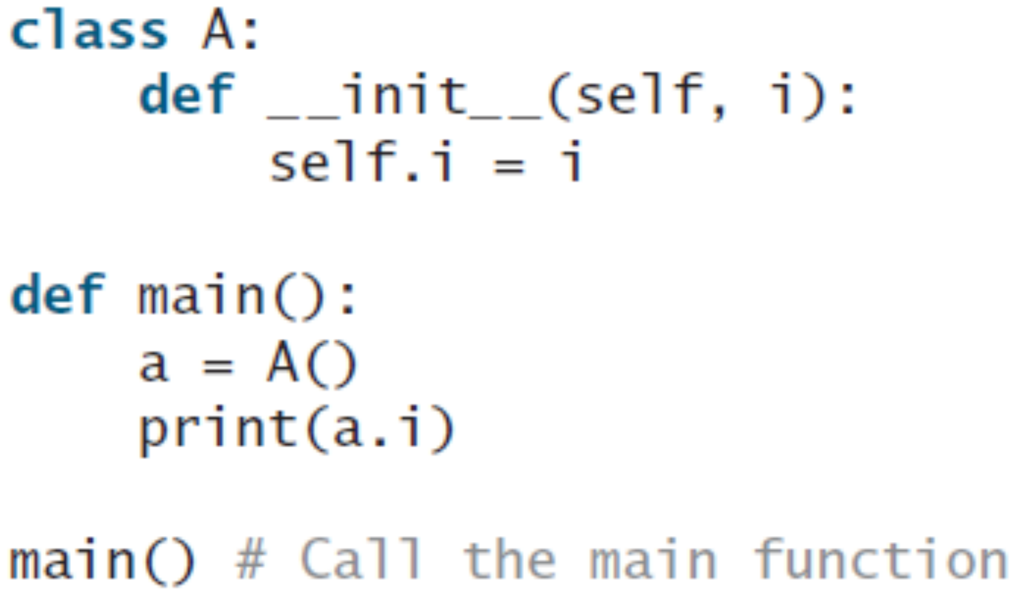
```
def main():
    # Create a circle with radius 1
    circle1 = Circle()
    print("The area of the circle of radius",
        circle1.radius , "is", circle1.getArea())

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

main() # Call the main function
```

Result



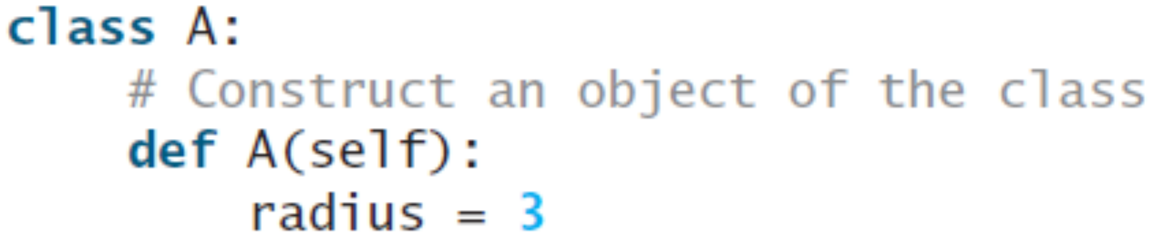
```
The area of the circle of radius 1.0 is 3.141592653589793
The area of the circle of radius 25.0 is 1963.4954084936207
The area of the circle of radius 125.0 is 49087.385212340516
The area of the circle of radius 100.0 is 31415.926535897932
```

What is wrong with this program?

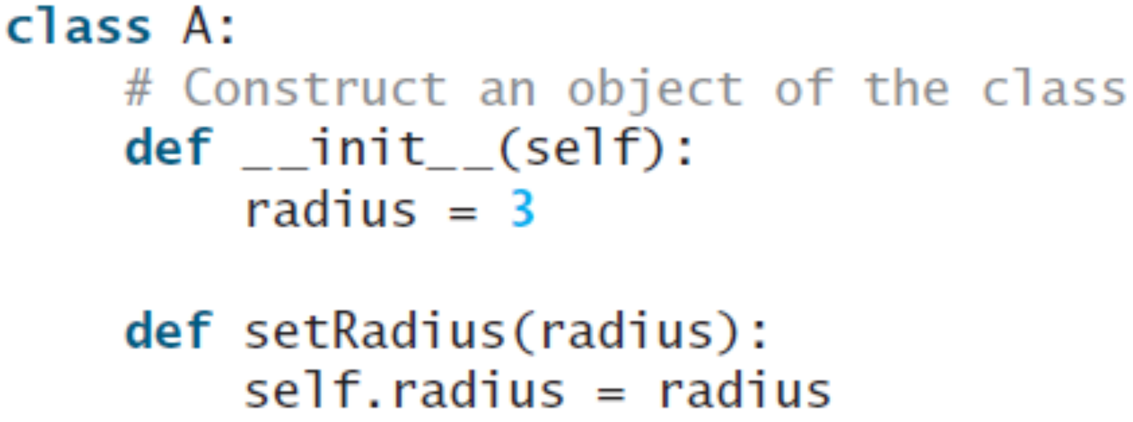


```
class A:
    def __init__(self, i):
        self.i = i

def main():
    a = A()
    print(a.i)

main() # Call the main function
```

What is wrong with these program?

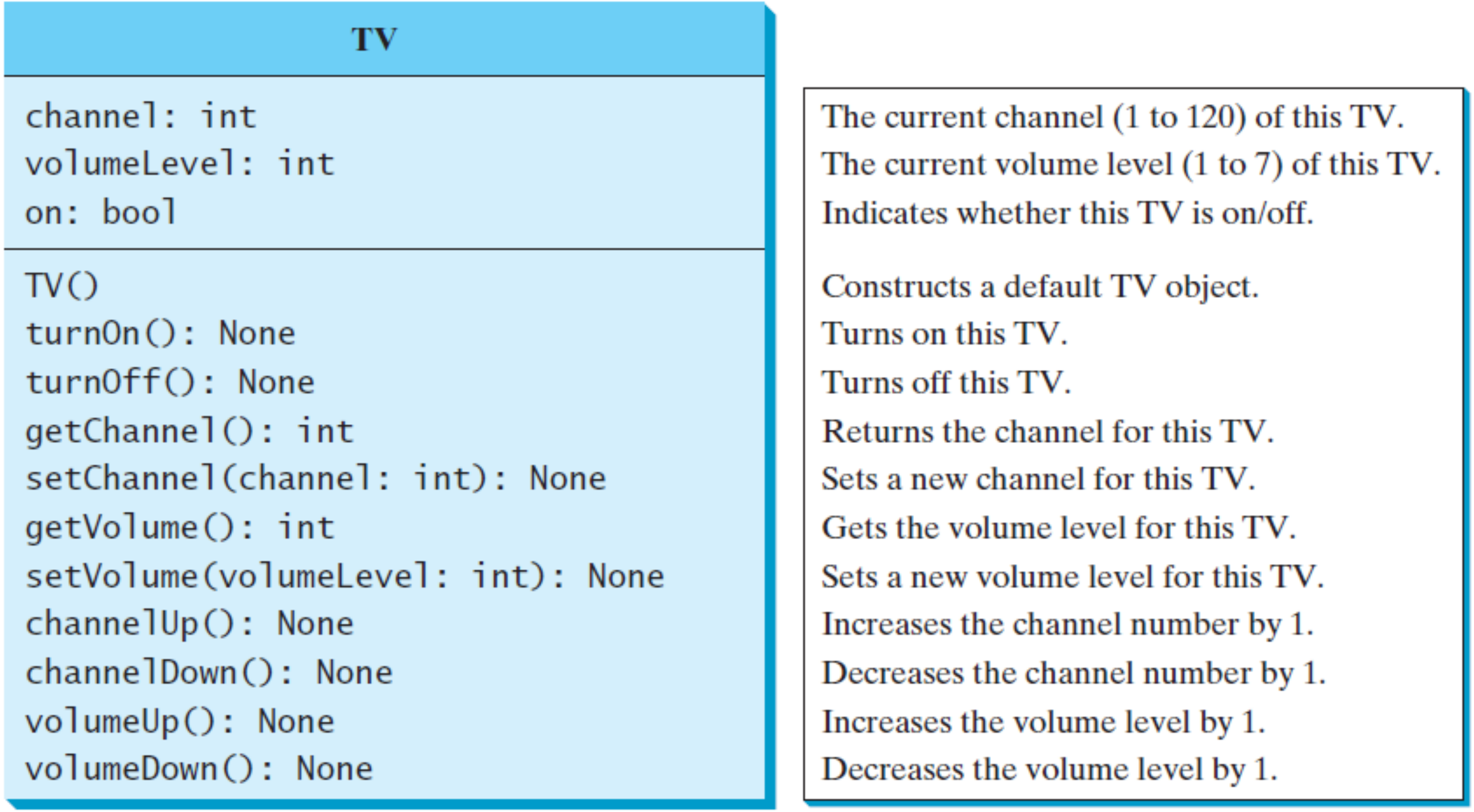


```
class A:
    # Construct an object of the class
    def A(self):
        radius = 3
```



```
class A:
    # Construct an object of the class
    def __init__(self):
        radius = 3

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

Example: TV class

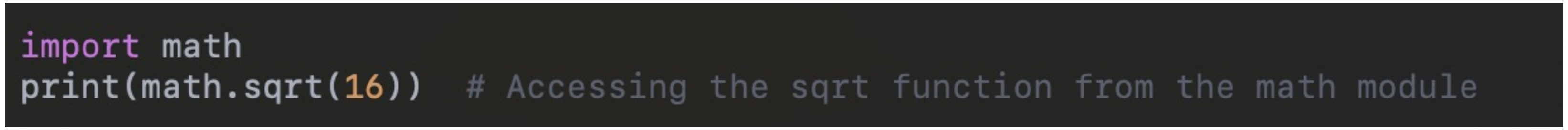


TV	
channel: int	The current channel (1 to 120) of this TV.
volumeLevel: int	The current volume level (1 to 7) of this TV.
on: bool	Indicates whether this TV is on/off.
TV()	Constructs a default TV object.
turnOn(): None	Turns on this TV.
turnOff(): None	Turns off this TV.
getChannel(): int	Returns the 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.

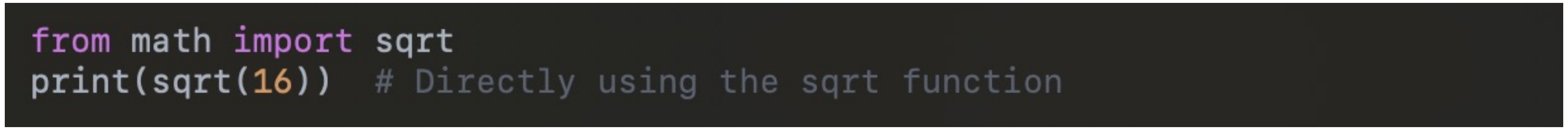
```python
class TV:
    def __init__(self):
        self.channel = 1  # Default channel is 1
        self.volumeLevel = 1  # Default volume level is 1
        self.on = False  # Initially, TV is off

    def turnOn(self):
        self.on = True

    def turnOff(self):
        self.on = False

    def getChannel(self):
        return self.channel

    def setChannel(self, channel):
        if self.on and 1 <= self.channel <= 120:
            self.channel = channel

    def getVolumeLevel(self):
        return self.volumeLevel

    def setVolume(self, volumeLevel):
        if self.on and \
              1 <= self.volumeLevel <= 7:
            self.volumeLevel = volumeLevel

    def channelUp(self):
        if self.on and self.channel < 120:
            self.channel += 1

    def channelDown(self):
        if self.on and self.channel > 1:
            self.channel -= 1

    def volumeUp(self):
        if self.on and self.volumeLevel < 7:
            self.volumeLevel += 1

    def volumeDown(self):
        if self.on and self.volumeLevel > 1:
            self.volumeLevel -= 1
```

Example: the code to use TV class

In Python, import and from … import are two ways to bring modules and their contents (like functions, classes, or variables) into your current script or notebook.



```
import math
print(math.sqrt(16))  # Accessing the sqrt function from the math module
```

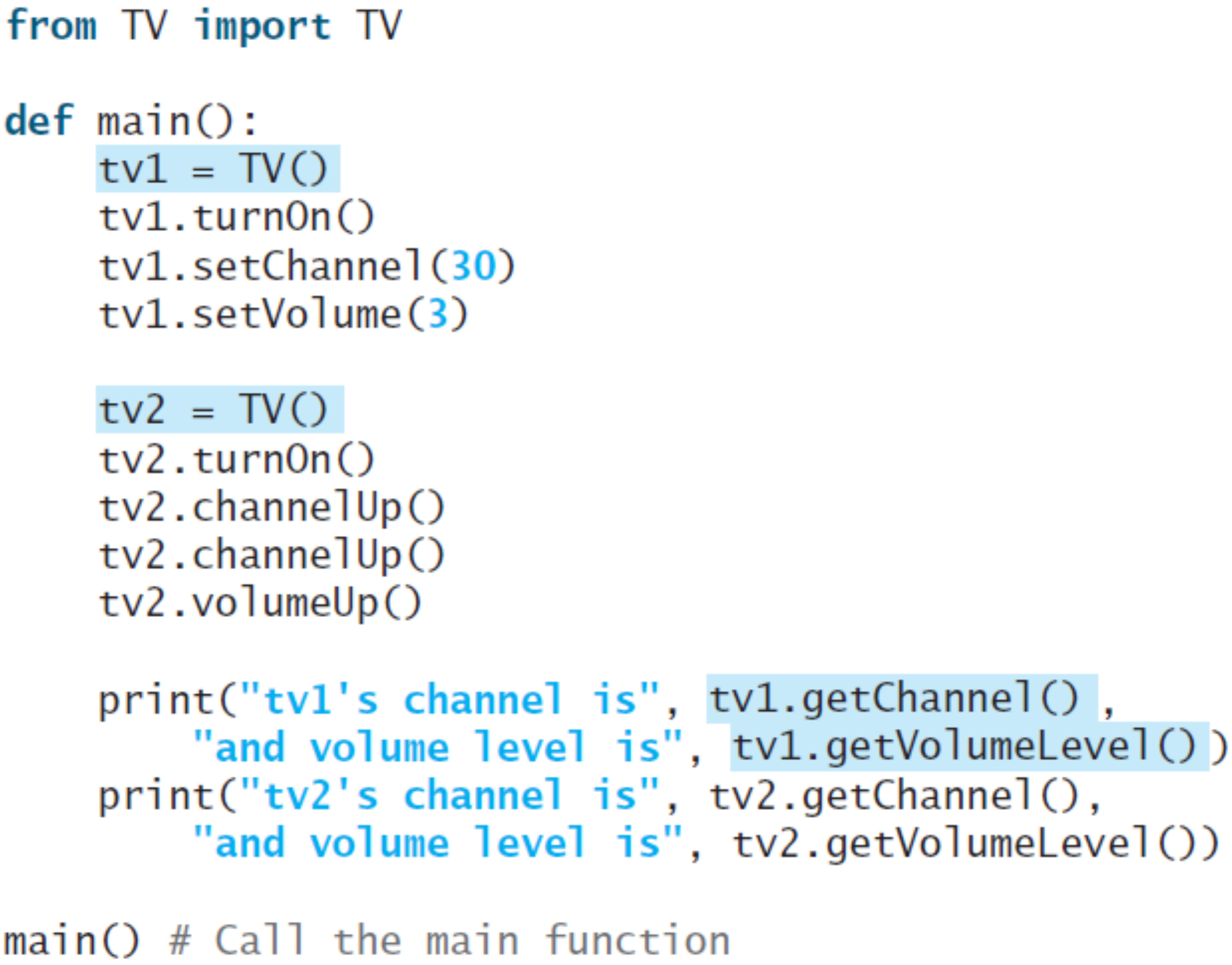


```
from math import sqrt
print(sqrt(16))  # Directly using the sqrt function
```

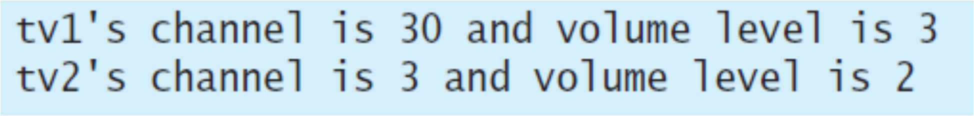
Importing everything: from module import *

Be careful when using from … import * (Risk of Overwriting Names and Namespace Pollution)

Example: the code to use TV class

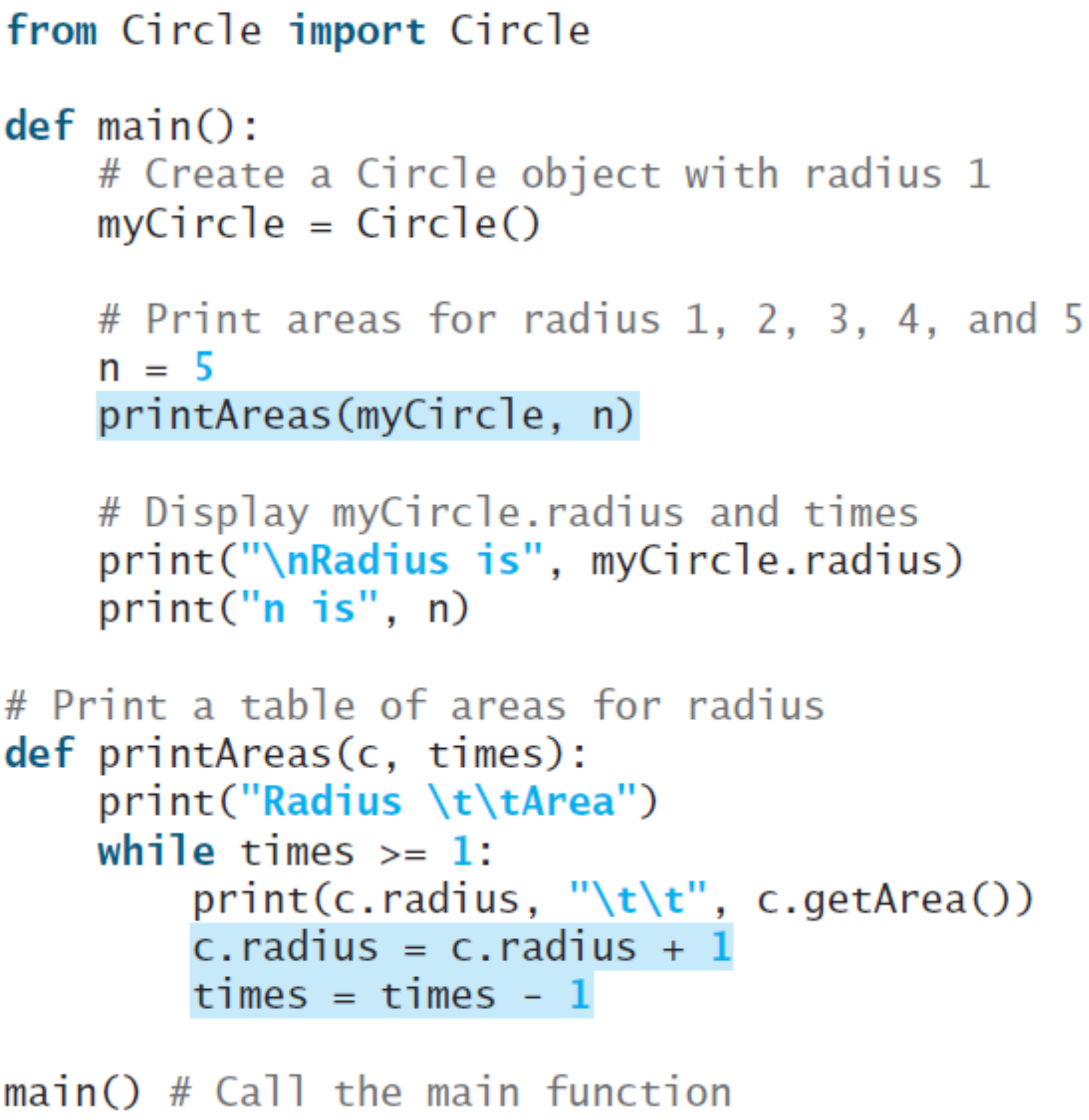


```python
from TV import TV

def main():
    tv1 = TV()
    tv1.turnOn()
    tv1.setChannel(30)
    tv1.setVolume(3)

    tv2 = TV()
    tv2.turnOn()
    tv2.channelUp()
    tv2.channelUp()
    tv2.volumeUp()

    print("tv1's channel is", tv1.getChannel() ,
        "and volume level is", tv1.getVolumeLevel() )
    print("tv2's channel is", tv2.getChannel(),
        "and volume level is", tv2.getVolumeLevel())

main() # Call the main function
```

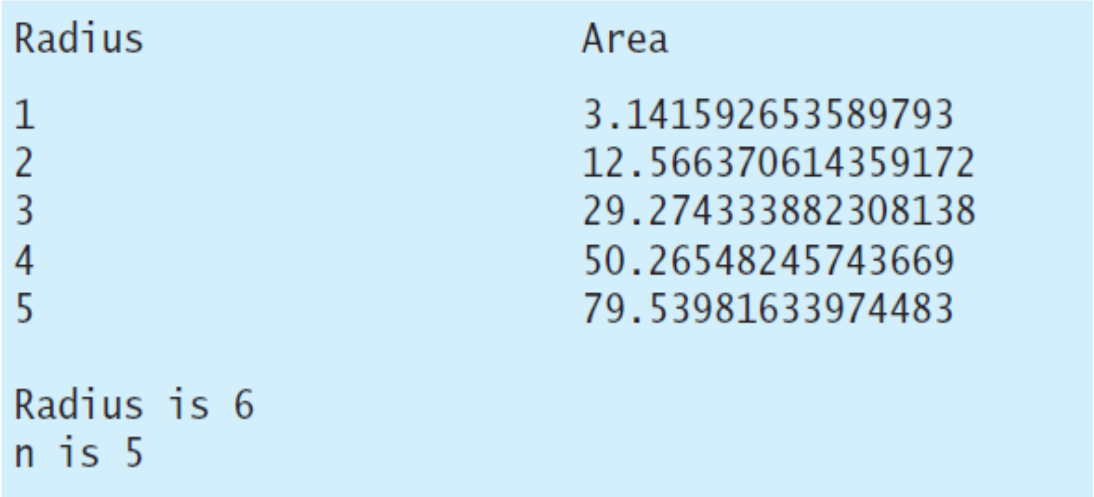


```
tv1's channel is 30 and volume level is 3
tv2's channel is 3 and volume level is 2
```

Mutable objects

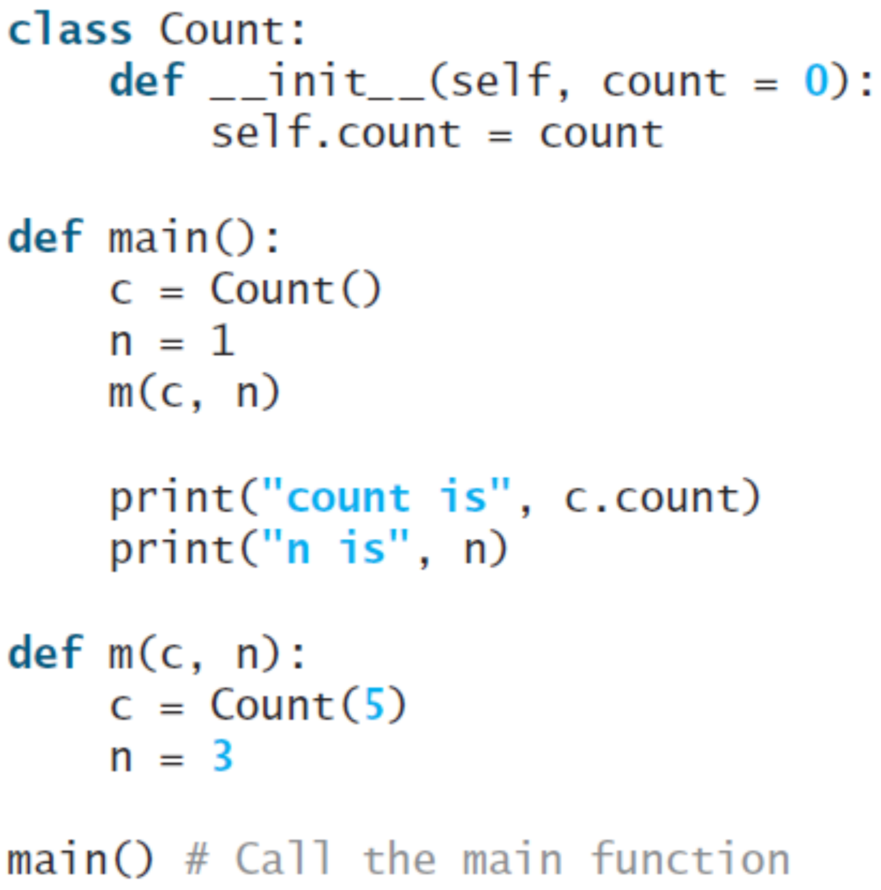


```
from Circle import Circle

def main():
    # Create a Circle object with radius 1
    myCircle = Circle()

    # Print areas for radius 1, 2, 3, 4, and 5
    n = 5
    printAreas(myCircle, n)

    # Display myCircle.radius and times
    print("\nRadius is", myCircle.radius)
    print("n is", n)

# Print a table of areas for radius
def printAreas(c, times):
    print("Radius \t\tArea")
    while times >= 1:
        print(c.radius, "\t\t", c.getArea())
        c.radius = c.radius + 1
        times = times - 1

main() # Call the main function
```



```
Radius          Area

1               3.141592653589793
2               12.566370614359172
3               29.274333882308138
4               50.26548245743669
5               79.53981633974483

Radius is 6
n is 5
```

Practice



```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```

- What would be the output of the above program?

Hiding data fields

- Direct access of a data field in an object is not good !!

- First, data may be tampered with

- Second, the class becomes difficult to maintain and vulnerable to bugs

Private data fields

- Prevent other programmers from directly accessing the data fields of your class is a common industrial practice

- This is known as data hiding

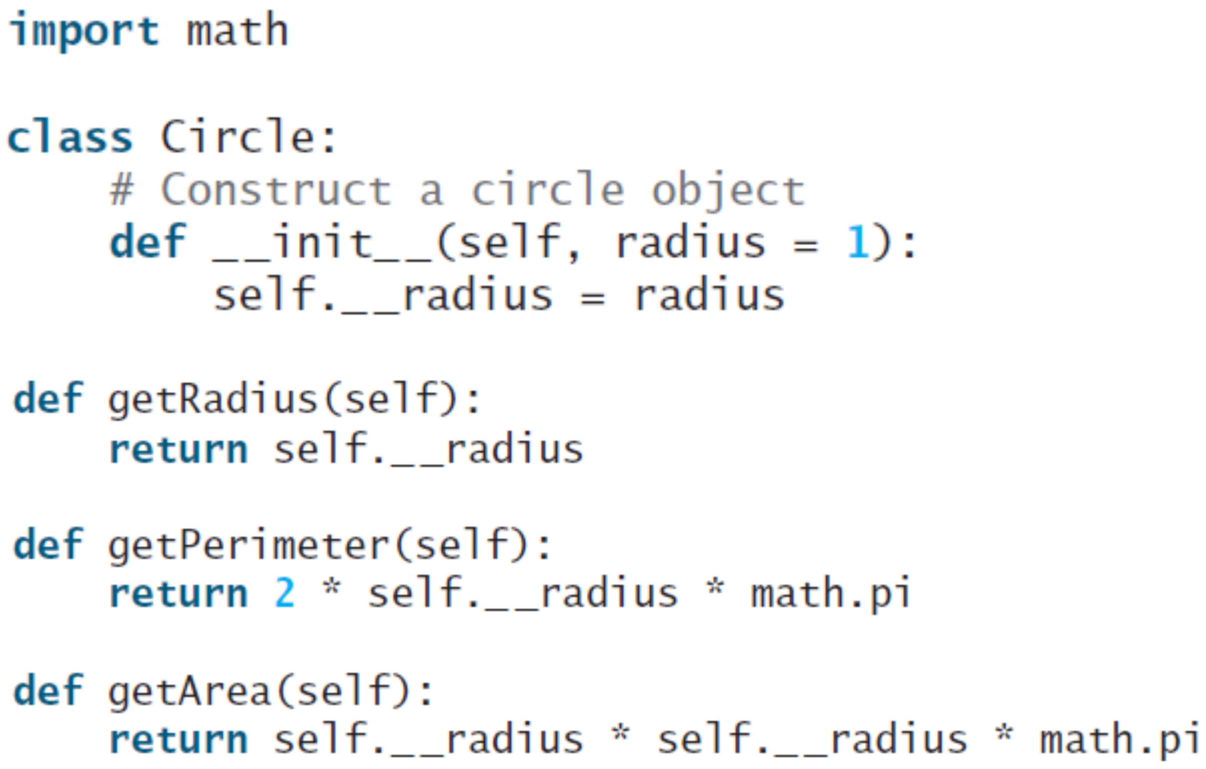
- This can be done by defining private data fields

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

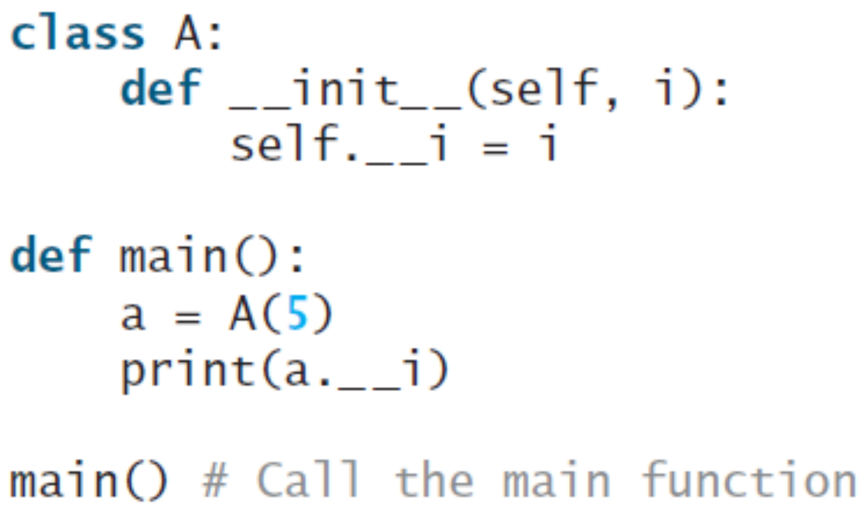
- Private data fields and methods can be accessed within a class, but they cannot be accessed outside the class

- Define some methods to allow access to private data fields



```
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
```

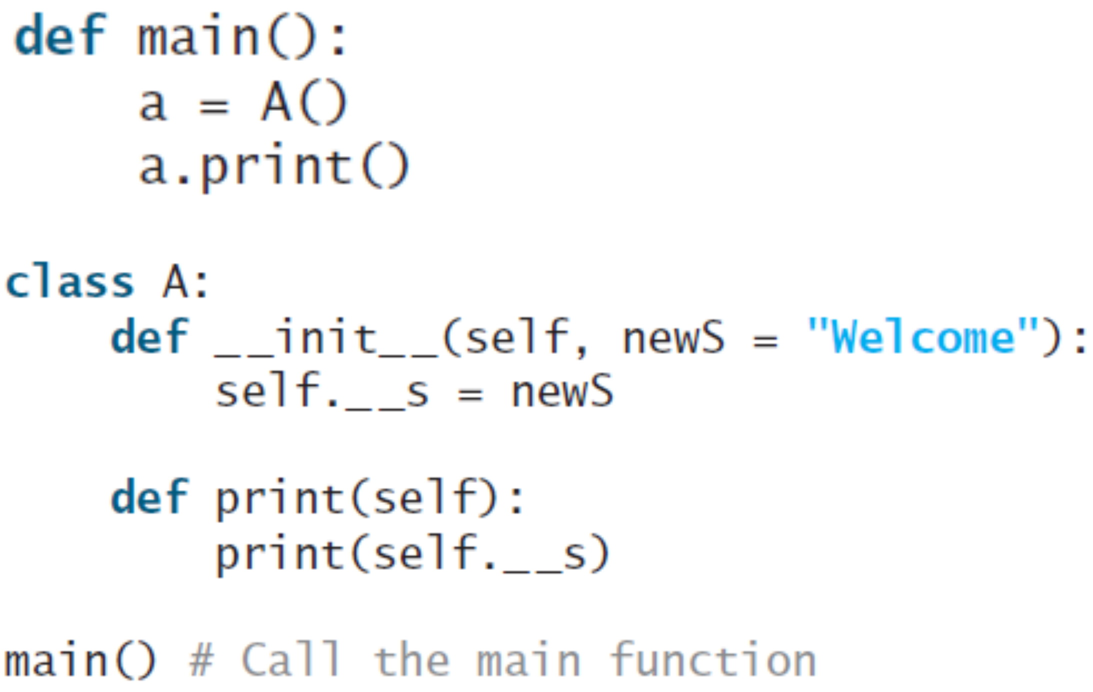
Practice



```
class A:
    def __init__(self, i):
        self.__i = i

def main():
    a = A(5)
    print(a.__i)

main() # Call the main function
```

- What is the problem with this program?

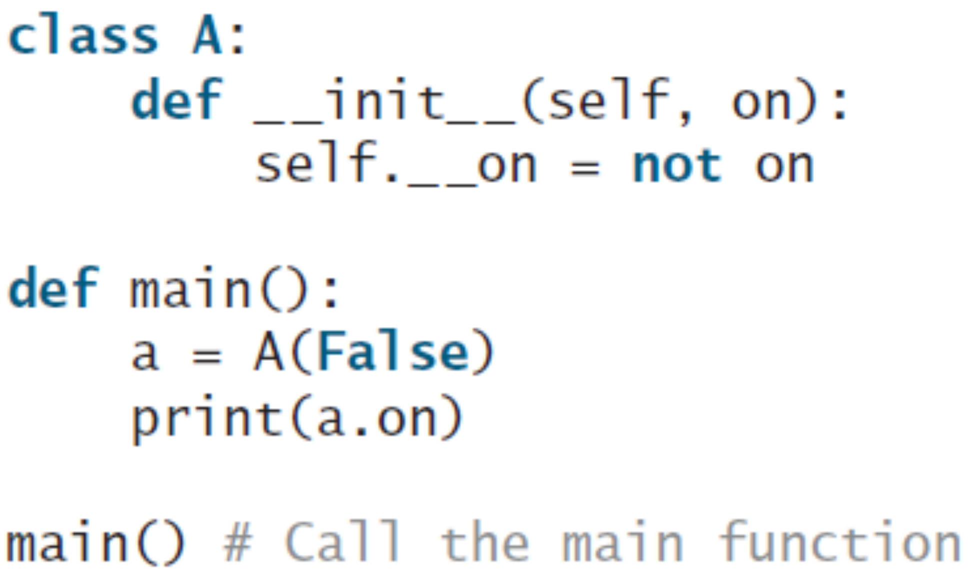
Practice



```
def main():
    a = A()
    a.print()

class A:
    def __init__(self, newS = "Welcome"):
        self.__s = newS

    def print(self):
        print(self.__s)

main() # Call the main function
```

- Is the above code correct? If yes, what would be the output?

Practice



```
class A:
    def __init__(self, on):
        self.__on = not on

def main():
    a = A(False)
    print(a.on)

main() # Call the main function
```

- Is the above code correct? If not, how do we fix it?

Abstraction

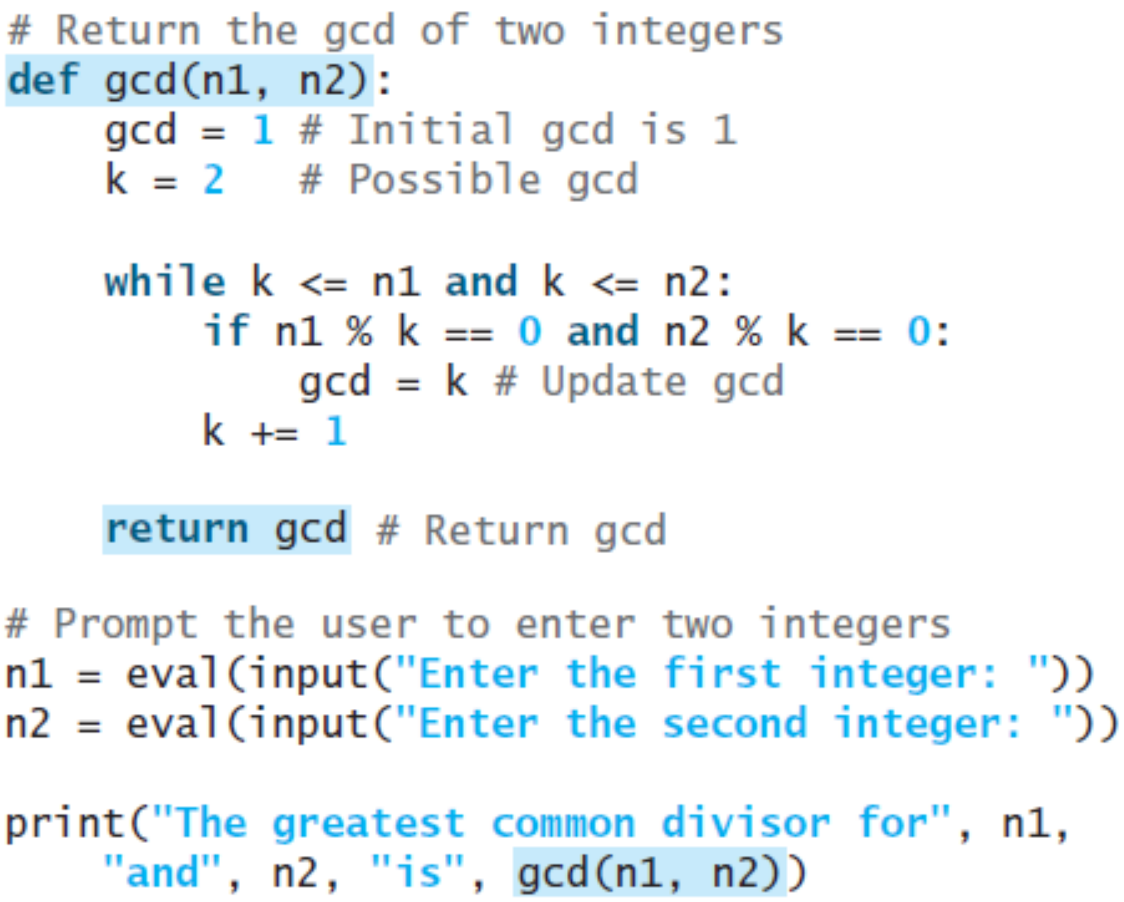
- Abstraction means separate the implementation of a part of code from the usage of that code

- In software engineering, there are many levels of abstraction, a commonly used one is called function abstraction

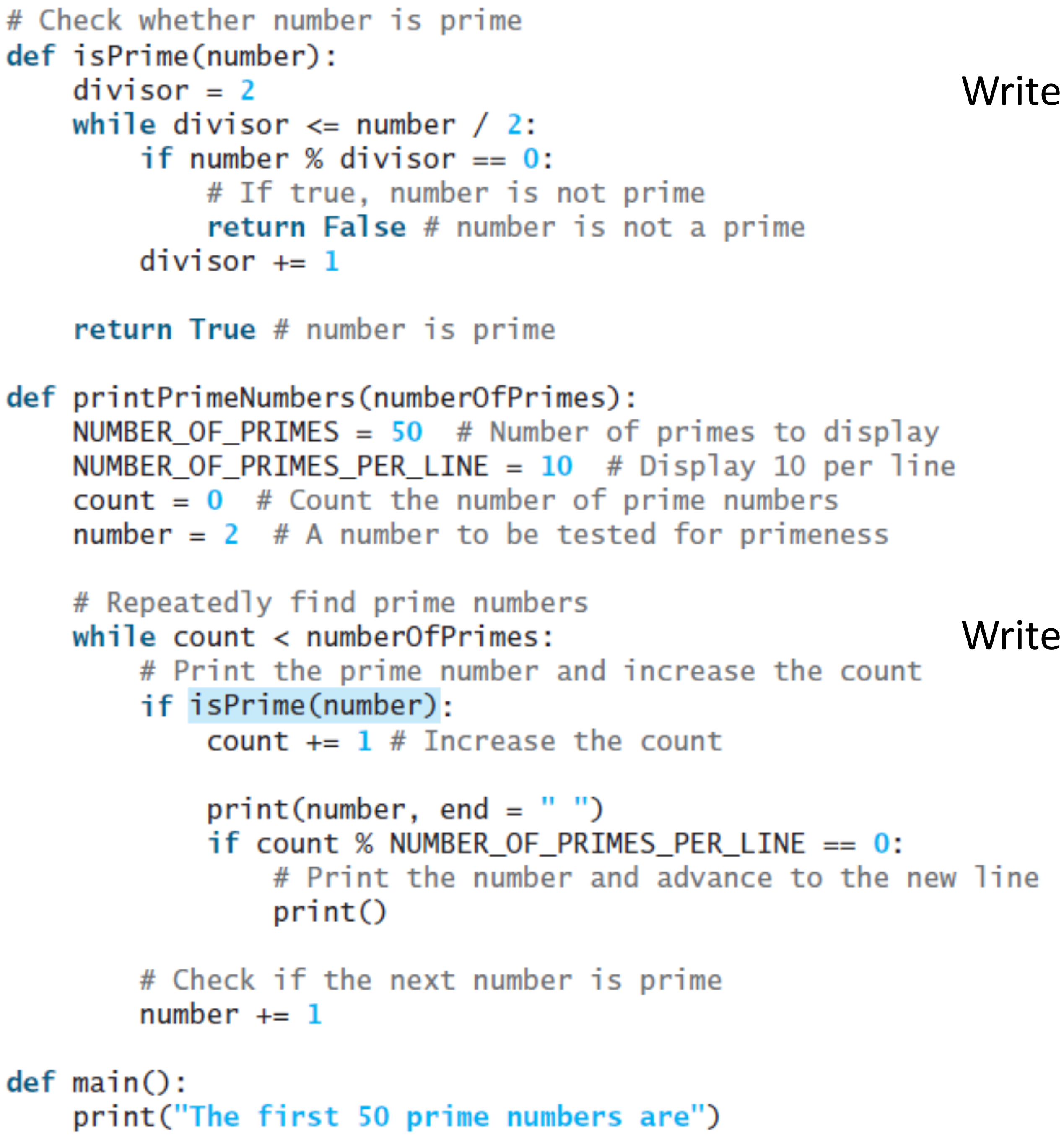
- Function abstraction means separating the implementation of a function from its usage

- Abstraction makes your code easy to maintain, debug and reuse

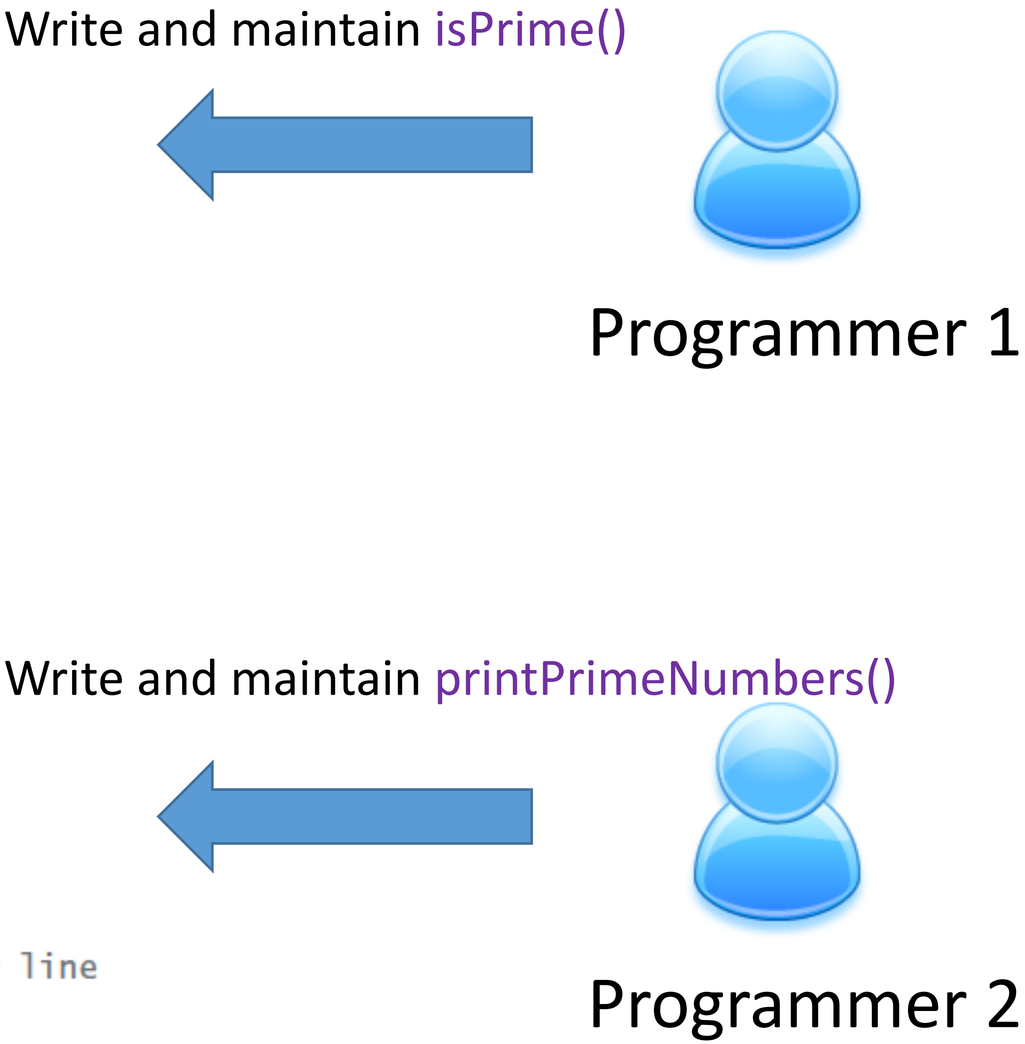
Example



```
# Return the gcd of two integers
def gcd(n1, n2):
    gcd = 1 # Initial gcd is 1
    k = 2   # Possible gcd

    while k <= n1 and k <= n2:
        if n1 % k == 0 and n2 % k == 0:
            gcd = k # Update gcd
        k += 1

    return gcd # Return gcd

# Prompt the user to enter two integers
n1 = eval(input("Enter the first integer: "))
n2 = eval(input("Enter the second integer: "))

print("The greatest common divisor for", n1,
    "and", n2, "is", gcd(n1, n2))
```



```python
# Check whether number is prime
def isPrime(number):
    divisor = 2
    while divisor <= number / 2:
        if number % divisor == 0:
            # If true, number is not prime
            return False # number is not a prime
        divisor += 1

    return True # number is prime

def printPrimeNumbers(numberOfPrimes):
    NUMBER_OF_PRIMES = 50  # Number of primes to display
    NUMBER_OF_PRIMES_PER_LINE = 10  # Display 10 per line
    count = 0  # Count the number of prime numbers
    number = 2  # A number to be tested for primeness

    # Repeatedly find prime numbers
    while count < numberOfPrimes:
        # Print the prime number and increase the count
        if isPrime(number):
            count += 1 # Increase the count

            print(number, end = " ")
            if count % NUMBER_OF_PRIMES_PER_LINE == 0:
                # Print the number and advance to the new line
                print()

        # Check if the next number is prime
        number += 1

def main():
    print("The first 50 prime numbers are")
```



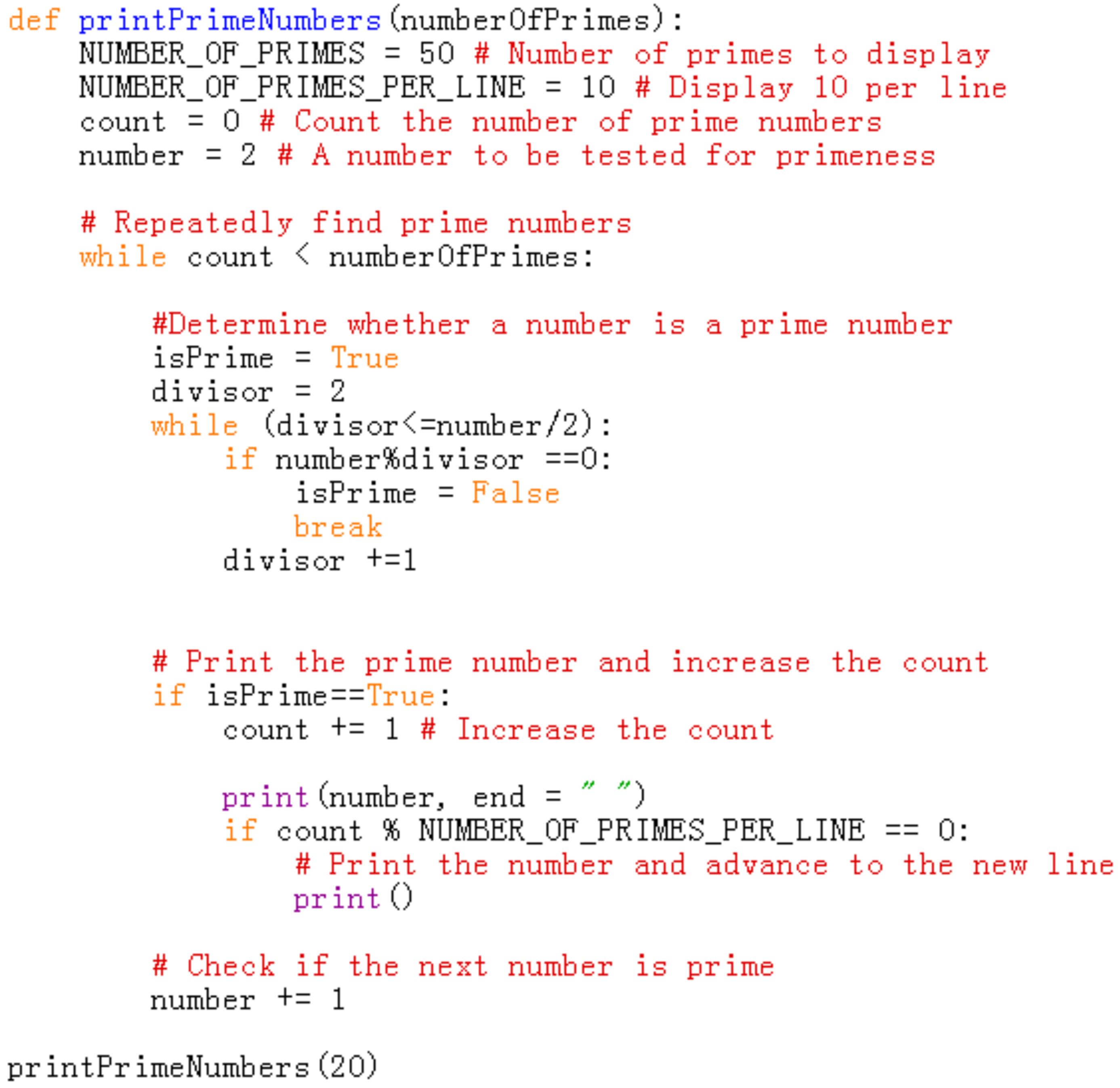
Write and maintain isPrime()

Programmer 1

Write and maintain printPrimeNumbers()

Programmer 2

If we write everything together…



```
def printPrimeNumbers(numberOfPrimes):
    NUMBER_OF_PRIMES = 50 # Number of primes to display
    NUMBER_OF_PRIMES_PER_LINE = 10 # Display 10 per line
    count = 0 # Count the number of prime numbers
    number = 2 # A number to be tested for primeness

    # Repeatedly find prime numbers
    while count < numberOfPrimes:

        #Determine whether a number is a prime number
        isPrime = True
        divisor = 2
        while (divisor<=number/2):
            if number%divisor ==0:
                isPrime = False
                break
            divisor +=1


        # Print the prime number and increase the count
        if isPrime==True:
            count += 1 # Increase the count

            print(number, end = " ")
            if count % NUMBER_OF_PRIMES_PER_LINE == 0:
                # Print the number and advance to the new line
                print()

        # Check if the next number is prime
        number += 1

printPrimeNumbers(20)
```

Class abstraction and encapsulation

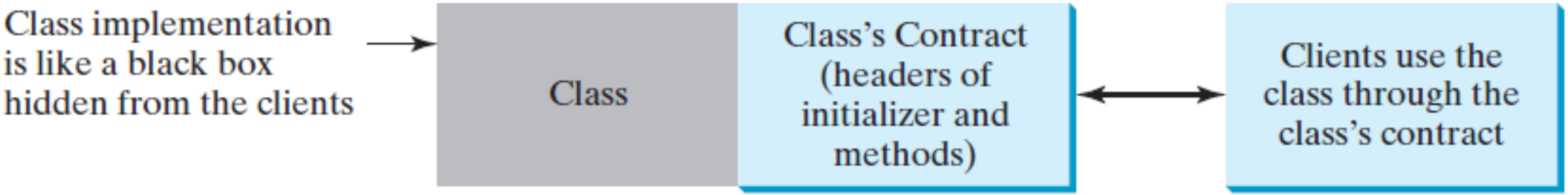
- Class abstraction means separating class implementation from the use of a class

- The class implementation details are invisible from the user

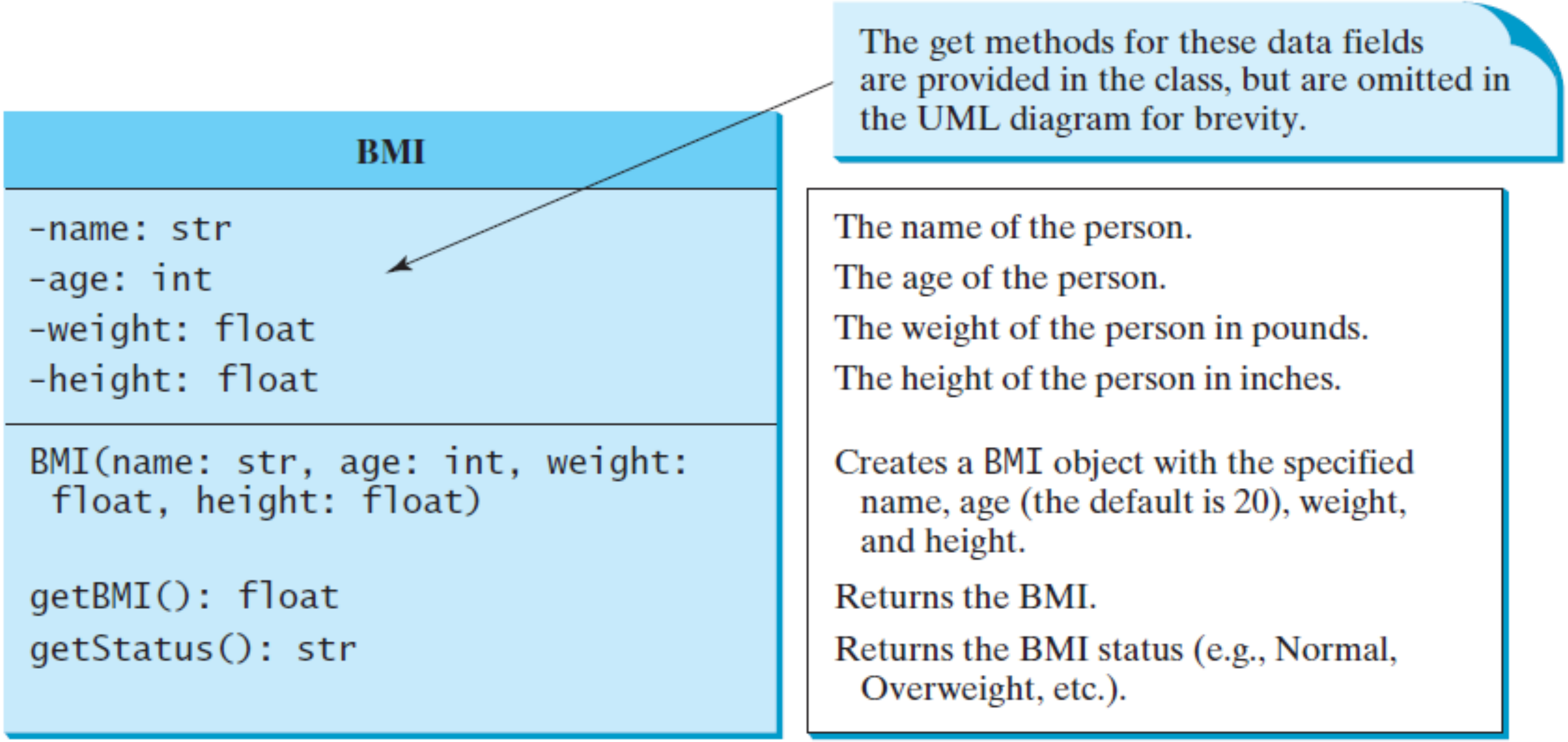
- The class’s collection of methods, together with the description of how these methods are expected to behave, serves as the class’s contract with the client

Class abstraction and encapsulation

- The user of the class does not need to know how the class is implemented. The details of implementation are encapsulated and hidden from the user.
- This is known as class encapsulation
- In essence, encapsulation combines data and methods into a single object and hides the data fields and method implementation from the user


Class implementation is like a black box hidden from the clients
Class
Class's Contract (headers of initializer and methods)
Clients use the class through the class's contract

Example – BMI calculation



The get methods for these data fields are provided in the class, but are 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.

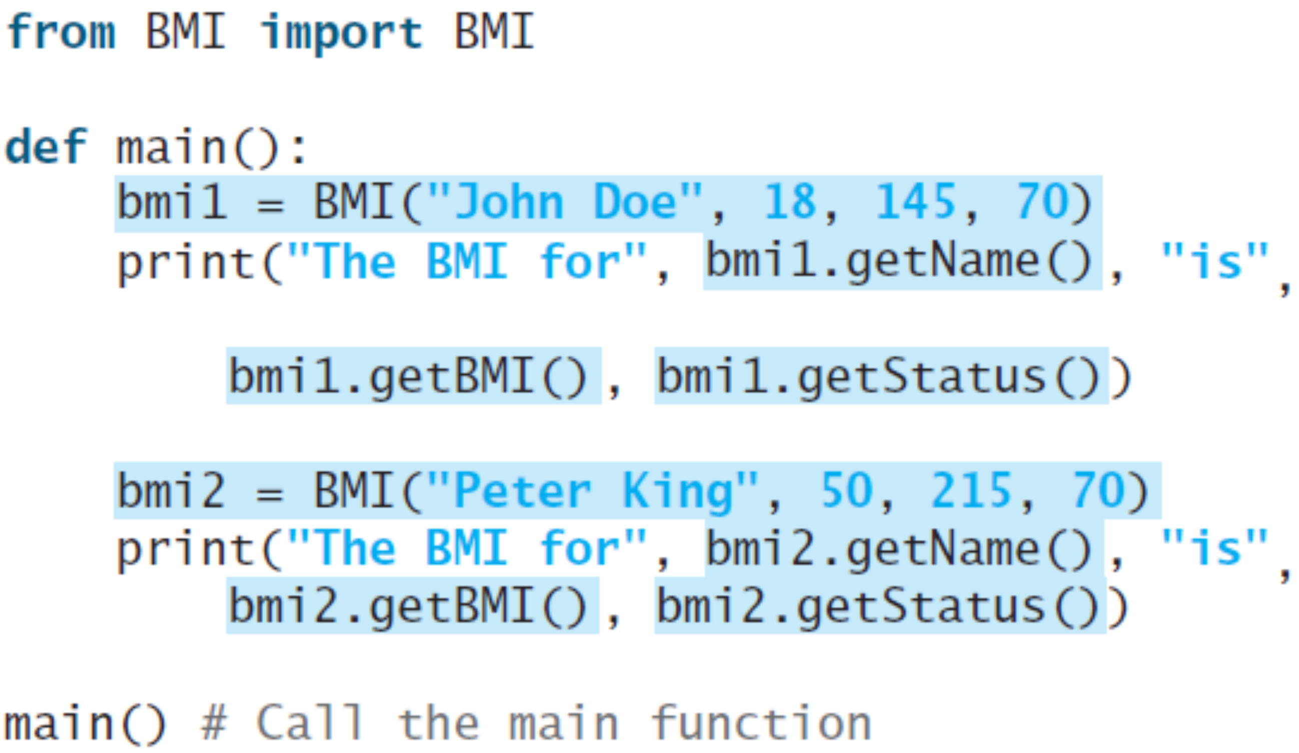
The height of the person in inches.

Creates a BMI object with the specified name, age (the default is 20), weight, and height.

Returns the BMI.

Returns the BMI status (e.g., Normal, Overweight, etc.).

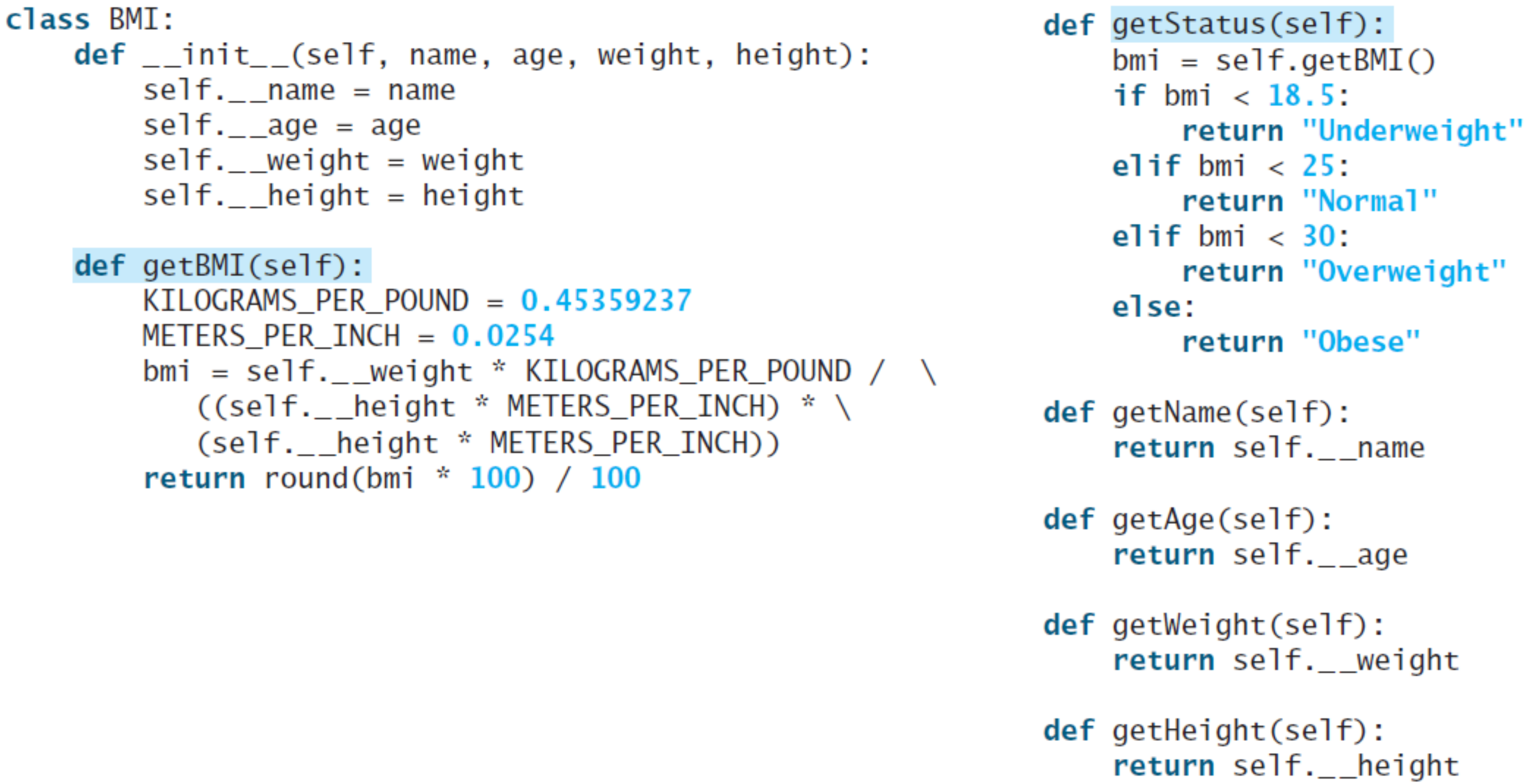
The code to use BMI class



```
from BMI import BMI

def main():
    bmi1 = BMI("John Doe", 18, 145, 70)
    print("The BMI for", bmi1.getName(), "is",

        bmi1.getBMI(), bmi1.getStatus())

    bmi2 = BMI("Peter King", 50, 215, 70)
    print("The BMI for", bmi2.getName(), "is",
        bmi2.getBMI(), bmi2.getStatus())

main() # Call the main function
```

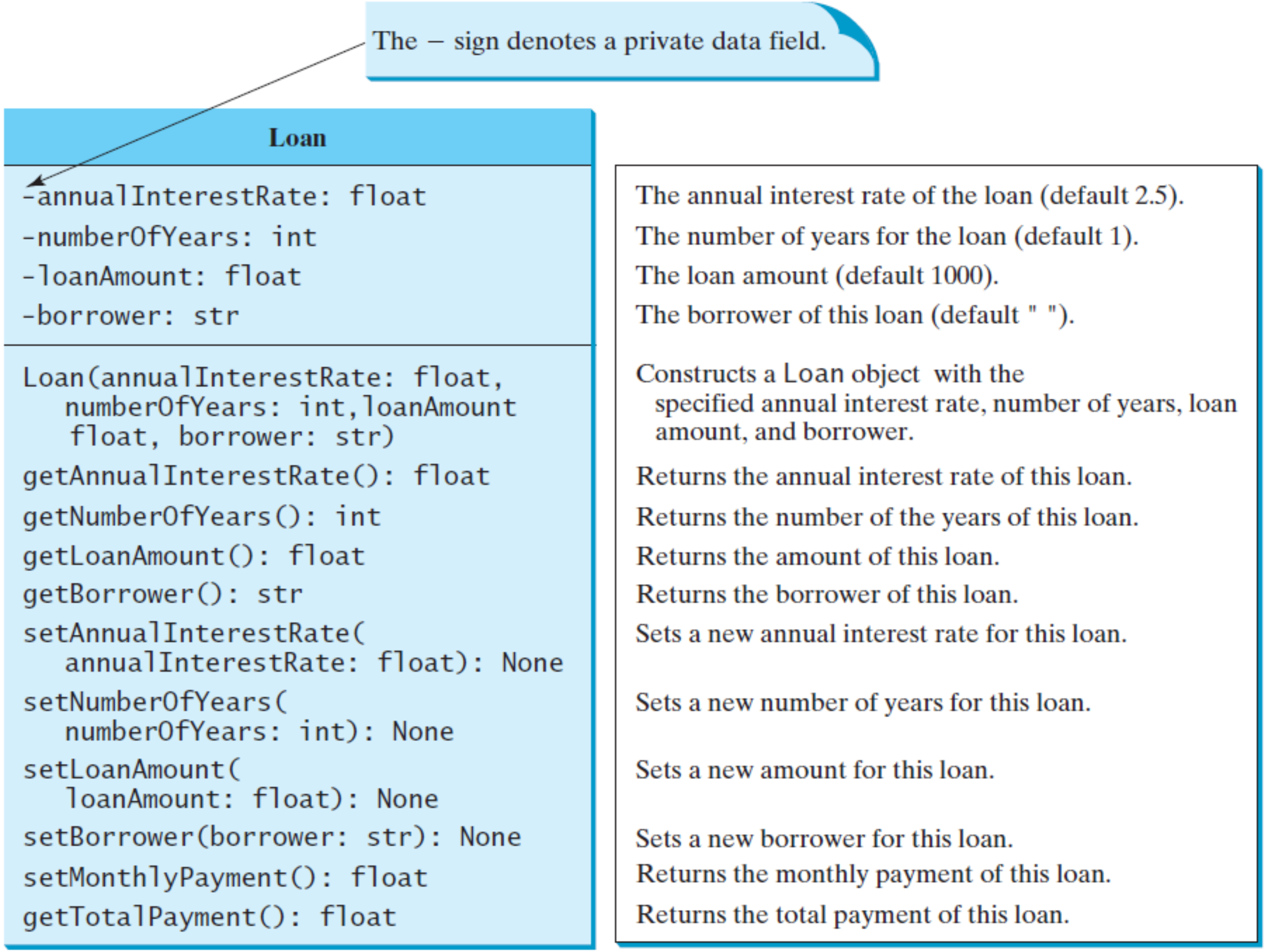
- We can use the BMI class if you have its contract
- You don’t need to know the details about how it is implemented!!

The BMI class



```
class BMI:
    def __init__(self, name, age, weight, height):
        self.__name = name
        self.__age = age
        self.__weight = weight
        self.__height = height

    def getBMI(self):
        KILOGRAMS_PER_POUND = 0.45359237
        METERS_PER_INCH = 0.0254
        bmi = self.__weight * KILOGRAMS_PER_POUND /  \
           ((self.__height * METERS_PER_INCH) * \
           (self.__height * METERS_PER_INCH))
        return round(bmi * 100) / 100

    def getStatus(self):
        bmi = self.getBMI()
        if bmi < 18.5:
            return "Underweight"
        elif bmi < 25:
            return "Normal"
        elif bmi < 30:
            return "Overweight"
        else:
            return "Obese"

    def getName(self):
        return self.__name

    def getAge(self):
        return self.__age

    def getWeight(self):
        return self.__weight

    def getHeight(self):
        return self.__height
```

Example - Loan



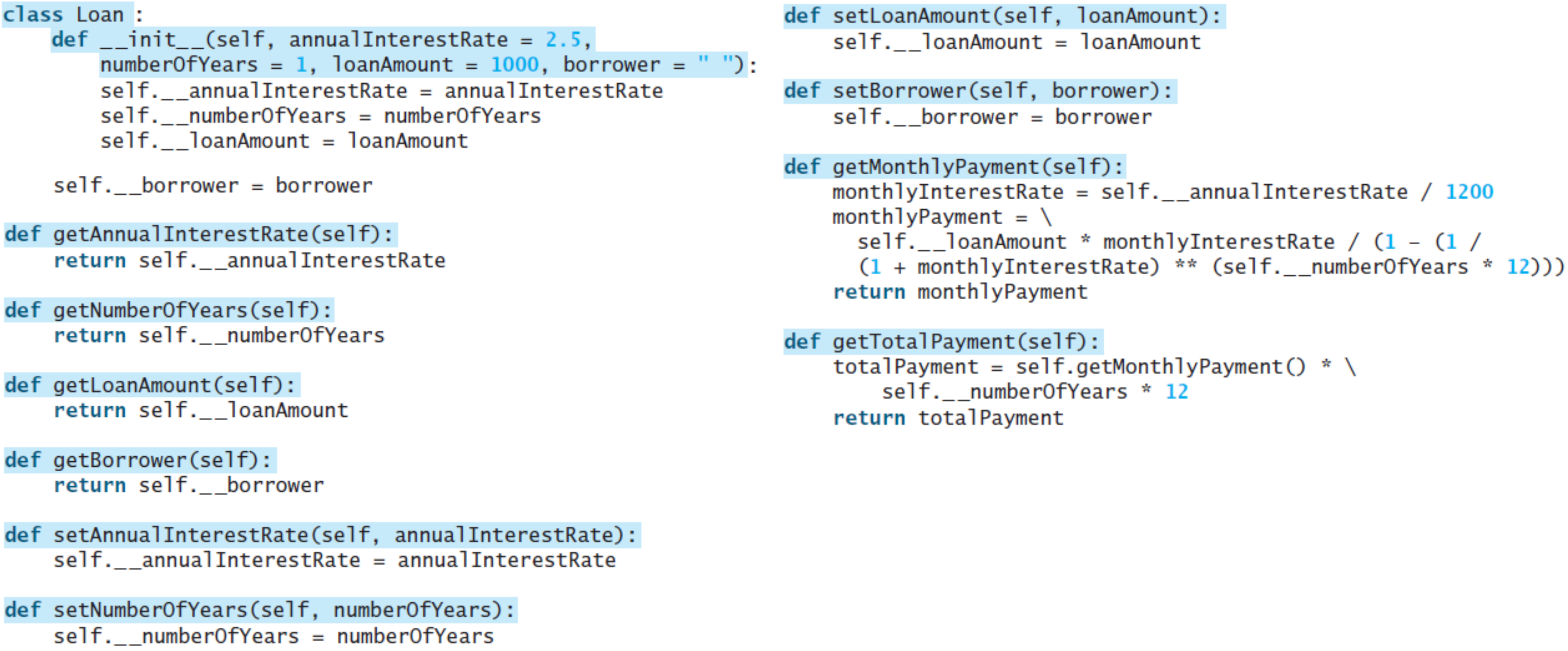
The – sign denotes a private data field.

Loan	
-annualInterestRate: float	The annual interest rate of the loan (default 2.5).
-numberOfYears: int	The number of years for the loan (default 1).
-loanAmount: float	The loan amount (default 1000).
-borrower: str	The borrower of this loan (default " ").
Loan(annualInterestRate: float, numberOfYears: int,loanAmount float, borrower: str)	Constructs a Loan object with the specified annual interest rate, number of years, loan amount, and borrower.
getAnnualInterestRate(): float	Returns the annual interest rate of this loan.
getNumberOfYears(): int	Returns the number of the years of this loan.
getLoanAmount(): float	Returns the amount of this loan.
getBorrower(): str	Returns the borrower of this loan.
setAnnualInterestRate(annualInterestRate: float): None	Sets a new annual interest rate for this loan.
setNumberOfYears(numberOfYears: int): None	Sets a new number of years for this loan.
setLoanAmount(loanAmount: float): None	Sets a new amount for this loan.
setBorrower(borrower: str): None	Sets a new borrower for this loan.
setMonthlyPayment(): float	Returns the monthly payment of this loan.
getTotalPayment(): float	Returns the total payment of this loan.

```python
from Loan import Loan

def main():
    # Enter yearly interest rate
    annualInterestRate = eval(input
        ("Enter yearly interest rate, for example, 7.25: "))

    # Enter number of years
    numberOfYears = eval(input(
        "Enter number of years as an integer: "))

    # Enter loan amount
    loanAmount = eval(input(
        "Enter loan amount, for example, 120000.95: "))

    # Enter a borrower
    borrower = input("Enter a borrower's name: ")

    # Create a Loan object
    loan = Loan(annualInterestRate, numberOfYears,
        loanAmount, borrower)

    # Display loan date, monthly payment, and total payment
    print("The loan is for", loan.getBorrower() )
    print("The monthly payment is",
        format(loan.getMonthlyPayment() , ".2f"))
    print("The total payment is",
        format(loan.getTotalPayment() , ".2f"))

main() # Call the main function
```

```
Enter yearly interest rate, for example, 7.25: 2.5 ↵Enter
Enter number of years as an integer: 5 ↵Enter
Enter loan amount, for example, 120000.95: 1000 ↵Enter
Enter a borrower's name: John Jones ↵Enter
The loan is for John Jones
The monthly payment is 17.75
The total payment is 1064.84
```

Example of loan class



```
class Loan :
    def __init__(self, annualInterestRate = 2.5,
        numberOfYears = 1, loanAmount = 1000, borrower = " "):
        self.__annualInterestRate = annualInterestRate
        self.__numberOfYears = numberOfYears
        self.__loanAmount = loanAmount

    self.__borrower = borrower

def getAnnualInterestRate(self):
    return self.__annualInterestRate

def getNumberOfYears(self):
    return self.__numberOfYears

def getLoanAmount(self):
    return self.__loanAmount

def getBorrower(self):
    return self.__borrower

def setAnnualInterestRate(self, annualInterestRate):
    self.__annualInterestRate = annualInterestRate

def setNumberOfYears(self, numberOfYears):
    self.__numberOfYears = numberOfYears
```

```
def setLoanAmount(self, loanAmount):
    self.__loanAmount = loanAmount

def setBorrower(self, borrower):
    self.__borrower = borrower

def getMonthlyPayment(self):
    monthlyInterestRate = self.__annualInterestRate / 1200
    monthlyPayment = \
     self.__loanAmount * monthlyInterestRate / (1 - (1 /
     (1 + monthlyInterestRate) ** (self.__numberOfYears * 12)))
    return monthlyPayment

def getTotalPayment(self):
    totalPayment = self.getMonthlyPayment() * \
        self.__numberOfYears * 12
    return totalPayment
```

Practice

(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.

Practice

(The Stock class) 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 for getting/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.