Python语言程序设计

Python Programming

Session 10



Tom Lotz (tom.lotz@outlook.com)

Content

	Review
01	Lists of objects
02	Objects working together
03	Refactoring and testing
04	Simple inheritance

Review

A Product Object



Product is a class. Product("tea", 12) creates an object.
Run this example first. We will unpack __init__ next.

Python code

```
class Product:
    def __init__(self, name, stock):
        self.name = name
        self.stock = stock

tea = Product("tea", 12)
print(tea.stock)
```

Expected output

```
12
```

One Class, Two Objects



Each call to Product(...) creates a new object.
Use the variable name to choose which object to read.

Python code

```
class Product:
    def __init__(self, name, stock):
        self.name = name
        self.stock = stock

tea = Product("tea", 12)
cake = Product("cake", 4)
print(tea.stock)
print(cake.stock)
```

Expected output

```
12
4
```

Keys and Attributes



Dictionary lookup: tea["stock"]

Object attribute: tea.stock

Both expressions retrieve the stock for tea.
An attribute is a named value on an object.

A Method Changes an Object



A method is a function defined in a class.
Keep restock indented at the same level as __init__.

Python code

```
class Product:
    def __init__(self, name, stock):
        self.name = name
        self.stock = stock

    def restock(self, amount):
        self.stock = self.stock + amount

tea = Product("tea", 12)
tea.restock(3)
print(tea.stock)
```

Lists of objects

A Team of Players

Last time, each Player kept its own score.

How could we print every score without writing a print statement for each player?

Today you will store objects in lists, give each class a clear job, and test what happens when objects work together.

The Player Class

Keep this class at the top of your IDLE file.
We will use it in the next examples.

Python code

```
class Player:
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def add_points(self, amount):
        self.score = self.score + amount
```

A List Can Hold Objects

Below Player, create two objects and put them in a list.
The index selects an object. The dot selects its attribute.

Python code

```
mina = Player("Mina", 5)
kai = Player("Kai", 2)
players = [mina, kai]

print(players[0].name)
print(players[1].score)
```

Expected output

```
Mina
2
```

A Loop Over Player Objects

Keep the same list. Each time around the loop, player refers to one Player object.

Python code

```
for player in players:
    print(player.name, player.score)
```

Expected output

```
Mina 5
Kai 2
```

Practice: Updating an Object in a List

CORE EXERCISE

Use the same players list.
Predict both lines before running.

Python code

```
players[1].add_points(4)
for p in players:
    print(p.name, p.score)
```

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Add a third Player to the list. Run the same loop again without editing it.

Object Updates: Solution

Index 1 selects Kai. The method changes Kai’s score. Mina stays at 5.

Python code

```
players[1].add_points(4)
for p in players:
    print(p.name, p.score)
```

Expected output

```
Mina 5
Kai 6
```

Objects working together

What Each Class Stores

Player stores a name and a score.

Team stores a list of Player objects.

When we call team.add_player(mina), team is self. mina is the argument that goes into the list.

A Team has players. It does not inherit from Player.

A Team Stores Players

Put Team below Player. Each new team starts with its own list. add_player receives an existing Player object.

Python code

```
class Team:
    def __init__(self):
        self.players = []

    def add_player(self, player):
        self.players.append(player)
```

Passing a Player to a Team

Put these lines below both classes. The list holds the same Player object that the variable mina refers to.

Python code

```
team = Team()
mina = Player("Mina", 5)
team.add_player(mina)
mina.add_points(3)

print(team.players[0].score)
```

Expected output

```
8
```

Practice: Two Teams

CORE EXERCISE

Use Player and Team.

Put Mina (5) and Kai (2) on one team. Put Noor (9) on another.

Print the names on each team using two loops.

Expected: Mina, Kai, then Noor.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Add another player to the first team. Check that the second team still has one player.

B. Change Noor’s score. Check it through the second team’s list.

Two Teams: Solution

Keep both class definitions above this code.

Python code

```
first = Team()
second = Team()
first.add_player(Player("Mina", 5))
first.add_player(Player("Kai", 2))
second.add_player(Player("Noor", 9))

for player in first.players:
    print(player.name)
for player in second.players:
    print(player.name)
```

Expected output

```
Mina
Kai
Noor
```

Refactoring and testing

A Total Outside the Class

This works, but every part of the program that needs a total would need to repeat the loop. Can the team do that work?

Python code

```
team = Team()
team.add_player(Player("Mina", 5))
team.add_player(Player("Kai", 2))

total = 0
for player in team.players:
    total = total + player.score
print(total)
```

Expected output

```
7
```

A Method for the Total

Add this inside Team, aligned with add_player.
Return the result so the calling code can use or print it.

Python code

```
    def total_score(self):
        total = 0
        for player in self.players:
            total = total + player.score
        return total
```

The Same Result After Refactoring

Use the updated Team. Refactoring changes the structure of the code while keeping its behavior the same.

Python code

```
team = Team()
team.add_player(Player("Mina", 5))
team.add_player(Player("Kai", 2))

print(team.total_score())
```

Expected output

```
7
```

Three Tests for total_score

Empty team: expected 0.

Two players with scores 5 and 2: expected 7.

Add 3 to the first player: expected 10.

Pick the expected result before running the code.
If it differs, trace which values the method actually reads.

Independent Practice: Team Totals

CORE EXERCISE

Add total_score() inside Team.

Print the total for an empty team.
Add Mina (5) and Kai (2). Print again.

Add 3 to Mina and print the new total.

Expected: 0, then 7, then 10.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Add a third player with score 0.
Predict and verify the total.

B. Write size() to return how many players are on the team.

Team Totals: Test Solution

Keep Player and the updated Team above these lines.

Python code

```
team = Team()
print(team.total_score())
mina = Player("Mina", 5)
team.add_player(mina)
team.add_player(Player("Kai", 2))
print(team.total_score())
mina.add_points(3)
print(team.total_score())
```

Expected output

```
0
7
10
```

Classes and Running Code

Put class definitions first.

Below them, create the objects and call their methods.

Keep printing and user input outside methods that just calculate a result. That makes those methods easier to test.

Use one file for today’s practice.

Simple inheritance

Has a Player or Is a Counter?

A Team has Player objects. It stores them in a list.

A ResettableCounter is a kind of Counter. It can use the original Counter methods and add a reset method.

Use inheritance when the new class is a kind of the old one.

The Base Class: Counter

Use this familiar class first. It stores a count and can add to it.

Python code

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

    def add(self, amount):
        self.count = self.count + amount
```

A Subclass Adds reset

Place this below Counter. ResettableCounter inherits __init__ and add from Counter, and defines reset itself.

Python code

```
class ResettableCounter(Counter):
    def reset(self):
        self.count = 0

c = ResettableCounter(5)
c.add(2)
print(c.count)
c.reset()
print(c.count)
```

Expected output

```
7
0
```

Practice: Which Counter Changes?

CORE EXERCISE

Use both counter classes.
Predict both lines before running.

Python code

```
first = Counter(8)
second = ResettableCounter(4)
second.reset()
print(first.count)
print(second.count)
```

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Call second.add(3) after reset.
Predict the two counts. Which class defines add?

Counter Prediction: Solution

reset acts only on second. first stays at 8.
The extra add call would change second from 0 to 3.

Python code

```
first = Counter(8)
second = ResettableCounter(4)
second.reset()
print(first.count)
print(second.count)
```

Expected output

```
8
0
```

Independent Check

1. What does team.players[0] select?
2. If team.add_player(mina) runs, which object is self?
3. What should total_score() return for an empty team?
4. Where does a ResettableCounter get its add method?

Recap

A list can hold objects. A loop can read each object.

One object can store or use other objects.

Move repeated calculations into a method and test the result.

A subclass can reuse methods from its base class.

Homework: A Reading List

CORE EXERCISE

Make a Book with title and pages.

Make a ReadingList that starts empty. Give it add_book(book) and total_pages().

Test the empty list, then add books of 80 and 120 pages. Submit your .py file.

Expected totals: 0, then 200.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Add a method that prints all titles.

B. Create a second reading list. Add a book to it and check that the first total stays the same.