

Python语言程序设计

Python Programming

Session 03



Tom Lotz (tom.lotz@outlook.com)

Content

Review

01

Lists

02

Create and access a list

03

Change, add, and remove items

04

Practice and homework

Review

Session 2 Review

- Three short problems revisit the main ideas from Session 2.
- Each problem uses a different situation.
- We will build each solution together.
- Ask whenever one step is unclear.



Review Problem 1: Library Printing Balance

- A student has 20 yuan in a printing account.
- One print job costs 6 yuan.
- A second print job costs 4 yuan.
- The program must display the remaining balance.
- Which value changes while the program runs?



Follow the Balance

- Start with a balance of 20 yuan.
- Subtract the first print job.
- Subtract the second print job.
- Display the remaining balance with a clear label.



The Balance Program

- Write the value of balance after each assignment.
- Then run the program and check the displayed balance.

```
balance = 20
first_job = 6
balance = balance - first_job
second_job = 4
balance = balance - second_job
print("Balance:", balance)
```



How the Balance Changes

- balance begins with 20.
- After the first print job, balance becomes 14.
- After the second print job, balance becomes 10.
- The program displays Balance: 10.
- Only the two subtraction lines change balance.



Review Problem 2: Club Announcement

- Stored club name: international chess club
- Stored location: student activity center
- Required output: International Chess Club meets at Student Activity Center
- Which words come from variables?
- Which words stay fixed in the sentence?



Stored Text and Displayed Text

- The program stores both values in lowercase.
- The displayed club name and location need title capitalization.
- The words "meets at" remain unchanged.
- A string method can prepare both stored values for display.



String Methods

- A string stores text inside quotation marks.
- .title() capitalizes the first letter of each word.
- .upper() makes every letter uppercase.
- .lower() makes every letter lowercase.
- The original stored value remains available.



Club Announcement: Solution

- The variables supply the changing information.
- The print instruction supplies the fixed words.

```
club_name = "international chess club"  
location = "student activity center"  
print(club_name.title(), "meets at", location.title())
```



Review Problem 3: Choose from the Café Menu

- A student has a budget of 50 yuan.
- Price list: tea 6 yuan, fruit cup 8 yuan, sandwich 15 yuan.
- The program asks which item the student wants and asks for its listed price.
- It calculates how many whole items the student can buy.
- It also displays how much money remains.



New Tools for the Calculation

- `input()` returns text, even when the user enters digits.
- `int()` converts the entered price into a whole number.
- `//` gives the number of complete items the budget can buy.
- `%` gives the money remaining after buying that number.
- For a 50 yuan budget and a 6 yuan tea: `50 // 6` is 8, and `50 % 6` is 2.



Complete the Purchase Program

- Choose an item from the price list and enter the matching price.
- Complete the two calculations before running the program.

```
budget = 50
item = input("What would you like? ")
price = int(input("Enter its price: "))
number = _____
money_left = _____
print("You can buy", number, item)
print("Money left:", money_left)
```



Purchase Program: Solution

- `number = budget // price`
- `money_left = budget % price`
- Tea: 8 items and 2 yuan left.
- Fruit cup: 6 items and 2 yuan left.
- Sandwich: 3 items and 5 yuan left.
- The same calculations work for every item on the price list.



Test a New Budget

- Change the budget from 50 yuan to 37 yuan.
- Run the program once for tea and once for sandwiches.
- Check the number of complete items and the money remaining.
- Then add one new item with a whole-number price to the menu.
- Use the same program to test your new item.



Review Checkpoint

- Can you follow a variable whose value changes?
- Can you format stored text with a string method?
- Can you perform several calculations in the required order?
- Can you store and use an answer from `input()`?
- Can you add labels that make displayed values understandable?
- Which point should we revisit before moving to lists?



Lists

Problem: Store Several Names

- A program needs to store four student names: Mina, Chen, Sara, and David.
- One possible plan uses four variables.
- What becomes difficult when the class has 20 students?
- What should stay together?

```
student_1 = "Mina"  
student_2 = "Chen"  
student_3 = "Sara"  
student_4 = "David"
```

What Is a List?

- A list stores several related values in one variable.
- Square brackets contain the items.
- Commas separate the items.
- The items remain in an order.

```
students = ["Mina", "Chen", "Sara", "David"]  
print(students)
```

Create and Display a List

- Type the code in IDLE.
- Predict the complete output before running it.
- Run the program and compare.

```
students = ["Mina", "Chen", "Sara", "David"]  
print(students)
```

Change and Explain

- Replace one name with another name.
- Predict the complete list before running the program.
- Run it and compare.
- Explain the purpose of the brackets, commas, and quotation marks.

Problem: Display One Name

- The complete list is stored in students.
- The program should display only Chen.
- How can Python identify the required item?

```
students = ["Mina", "Chen", "Sara", "David"]
```

List Indexes Start at Zero

- An index is the position number of an item.
- The first item has index 0.
- The second item has index 1.
- The last item can also be accessed with index -1.

```
students = ["Mina", "Chen", "Sara", "David"]  
print(students[1])
```

Predict Four Results

- Write the four outputs in order.
- Explain how each index and name are connected.
- Run the code only after your pair agrees.

```
students = ["Mina", "Chen", "Sara", "David"]  
print(students[0])  
print(students[1])  
print(students[3])  
print(students[-1])
```

Check the Indexes

- `students[0]` displays Mina.
- `students[1]` displays Chen.
- `students[3]` displays David.
- `students[-1]` also displays David.
- Write the rule for the index of the first item in your own words.

Changing One Item

- Use an index to select the item.
- Assign a new value to that position.
- Only the selected item changes.
- The other items remain in the same order.

```
courses = ["Python", "Mathematics", "History"]  
courses[2] = "English"  
print(courses)
```

Problem: Correct One Course

- The third course should be English.
- Print the list before the change.
- Print the required list after the change.
- Which index must receive a new value?

```
courses = ["Python", "Mathematics", "History"]
```

Solution: Replace One Item

- Predict the complete list, then run the code.
- Explain why only one item changes.

```
courses = ["Python", "Mathematics", "History"]  
courses[2] = "English"  
print(courses)
```

Adding an Item with append()

- `append()` adds one item to the end of a list.
- The existing items keep their order.
- The list becomes one item longer.

```
tasks = ["read notes", "write code"]  
tasks.append("test program")  
print(tasks)
```

Problem: Add a New Task

- The list contains read notes and write code.
- Add test program as the last task.
- Print the complete list before coding.
- Which method adds an item at the end?

```
tasks = ["read notes", "write code"]
```

Solution: Add with append()

- Run the code and compare.
- Explain where append() places the new item.

```
tasks = ["read notes", "write code"]  
tasks.append("test program")  
print(tasks)
```

Removing the Last Item with pop()

- pop() removes the last item by default.
- The removed value can be stored in a variable.
- The remaining list becomes one item shorter.

```
tasks = ["read notes", "write code", "test program"]  
completed_task = tasks.pop()  
print(completed_task)  
print(tasks)
```

Problem: Complete the Last Task

- Remove and remember the last task.
- Predict the removed value.
- Predict the remaining list.
- Which operation should the program use?

```
tasks = ["read notes", "write code", "test program"]
```

Solution: Remove with pop()

- Trace the two results separately.
- Explain why `completed_task` and `tasks` now hold different information.

```
tasks = ["read notes", "write code", "test program"]  
completed_task = tasks.pop()  
print("Completed:", completed_task)  
print("Remaining:", tasks)
```

Counting Items with len()

- len() returns the number of items in a list.
- The count changes after append() or pop().
- Predict each displayed number before running the code.

```
tasks = ["read notes", "write code"]  
print(len(tasks))  
tasks.append("test program")  
print(len(tasks))  
tasks.pop()  
print(len(tasks))
```

Practice

Final Problem: Build a Study Plan

- Create a list with three study activities.
- Display the complete list.
- Display the first activity.
- Replace the second activity.
- Add one new activity.
- Complete and remove the last activity.
- Display the remaining list and its length.

Plan the State Changes

- Before coding, draw four versions of the list.
- Starting list
- After replacing one item
- After adding one item
- After removing the last item
- Write the expected length beside every version.

Start with the Given Code

- Type and run the starting program.
- Check that the starting list matches your plan.

```
study_plan = ["read notes", "watch a video", "write code"]  
print("Complete plan:", study_plan)  
print("First activity:", study_plan[0])
```

Replace and Add

- Add these operations to the program.
- Predict the complete list before running.

```
study_plan[1] = "review examples"  
study_plan.append("test program")  
print("Updated plan:", study_plan)
```

Complete and Count

- Finish the program.
- Predict completed_activity, remaining, and the length.

```
completed_activity = study_plan.pop()
remaining = len(study_plan)
print("Completed:", completed_activity)
print("Remaining:", study_plan)
print("Number remaining:", remaining)
```

Check the Final State

- Compare the output with your state-change plan.
- If they differ, trace one operation at a time.
- Explain each index and list method to a partner.
- Then change the activities and test a second version.

Recap

What You Learned Today

- A list stores related values in one variable.
- Indexes identify individual items and begin at 0.
- Assignment can replace one item.
- `append()` adds an item at the end.
- `pop()` removes and returns the last item.
- `len()` counts the current items.
- Tracing the list after every change makes the program easier to understand.

Homework

Session 3 Homework

- Create a list containing five useful places near campus.
- Display the complete list and the first and last items.
- Replace one place, append one new place, and pop the last place.
- Display the removed place, the final list, and its length.
- Submit the .py file and your written predictions. Be ready to explain every line.