Python语言程序设计

Python Programming

Session 08



Tom Lotz (tom.lotz@outlook.com)

Content

	Review
01	Dictionaries: Keys and values
02	Adding and updating entries
03	Looping over dictionaries
04	Dictionaries and lists

Review

A Function: A Named Block of Code



A function groups instructions under a name. Call the function whenever you want those instructions to run.

Python code

```
def show_message():
    print("Ready")
    print("Let us begin")

show_message()
```

Expected output

```
Ready
Let us begin
```

A Parameter Receives a Value



number is a parameter: a name inside the function.
4 and 9 are arguments: the values supplied by the calls.

Python code

```
def show_double(number):
    print(number * 2)

show_double(4)
show_double(9)
```

Expected output

```
8
18
```

Two Parameters and Their Order



Arguments match parameters from left to right.
Reversing the arguments can change the result.

Python code

```
def show_difference(a, b):
    print(a - b)

show_difference(9, 4)
show_difference(4, 9)
```

Expected output

```
5
-5
```

return Sends a Value Back



return gives a result back to the caller.
The caller can store that result in a variable.

Python code

```
def double(number):
    return number * 2

result = double(7)
print(result)
```

Expected output

```
14
```

return Ends the Function Call



Once a return runs, that call ends.
A function can return numbers, strings, or Boolean values.

Python code

```
def magnitude(number):
    if number < 0:
        return -number
    return number

print(magnitude(-6))
print(magnitude(4))
```

Expected output

```
6
4
```

Local Variables



Each call has its own parameters and local variables.
The x outside the function remains unchanged.

Python code

```
x = 100
def double(x):
    result = x * 2
    return result

print(double(4))
print(x)
```

Expected output

```
8
100
```

Default Arguments



A default value makes an argument optional.
Parameters with defaults come after required parameters.

Python code

```
def total_price(price, quantity=1):
    return price * quantity

print(total_price(6))
print(total_price(6, 4))
```

Expected output

```
6
24
```

A Simple Module: Another Python File



- A module is a Python file that another program can import.
- Put reusable calculations in calculations.py.
- Keep main.py and calculations.py in the same folder.
- In IDLE, save both files and run main.py with F5.
- Keep keyboard input and the program loop in main.py.

Two Files Working Together



calculations.py

```
def add(a, b):
    return a + b
```

main.py

```
import calculations

print(calculations.add(9,4))
```

import makes the module available. calculations.add() calls its function.
Expected output: 13. The module itself does not ask for input.

Dictionaries: Keys and values

The Café Menu: Two Lists

The item name and its price share the same index.
At index 1, the item is fruit cup and the price is 8 yuan.

Python code

```
items = ["tea", "fruit cup", "sandwich"]
prices = [6, 8, 15]

print(items[1], prices[1])
```

Expected output

```
fruit cup 8
```

The Café Menu: One Dictionary

A dictionary stores key–value pairs.
Here the key is an item name and the value is its price.

Python code

```
prices = {"tea": 6,
          "fruit cup": 8,
          "sandwich": 15}

print(prices["fruit cup"])
```

Expected output

```
8
```

Dictionary Syntax

Part	Example	What it does
Braces	{ ... }	Hold the dictionary entries
Colon	"tea": 6	Separates a key from its value
Comma	"tea": 6, "cake": 9	Separates two entries
Square brackets	prices["tea"]	Look up a value by key

Lookup Using a Variable

Python uses the value stored in item as the key.
The expression prices[item] looks up the key "tea".

Python code

```
prices = {"tea": 6, "cake": 9}
item = "tea"
price = prices[item]
print(price)
```

Expected output

```
6
```

Practice: Predict the Lookups

CORE EXERCISE

Predict both lines of output.
Then check in IDLE.

Python code

```
rooms = {"art": 12,
         "music": 18}
subject = "music"
print(rooms[subject])
print(rooms["art"])
```

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Change only subject so the two lines print the same number.

Explain why rooms["subject"] would fail.

Room Lookups: Solution

subject holds "music", so the first lookup finds 18.
The second lookup uses "art" directly and finds 12.

Python code

```
rooms = {"art": 12, "music": 18}
subject = "music"
print(rooms[subject])
print(rooms["art"])
```

Expected output

```
18
12
```

Practice: A Points Dictionary

CORE EXERCISE

Store these team scores:
red: 4, blue: 7, green: 2.

Set team to "blue". Use team to look up and print its score.

Print the red score using the key directly.

Expected output: 7, then 4.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Print the combined score of red and green. Get both values from the dictionary.

Change team to "green" and explain what changes.

Team Points: Solution

A key can come from a variable or appear directly in the lookup.

Python code

```
scores = {"red": 4, "blue": 7,
          "green": 2}
team = "blue"
print(scores[team])
print(scores["red"])
```

Expected output

```
7
4
```

Adding and updating entries

Adding an Entry

Assign a value to a new key to add an entry.

Python code

```
prices = {"tea": 6, "cake": 9}
prices["juice"] = 10
print(prices["juice"])
```

Expected output

```
10
```

Updating an Entry

Assigning to an existing key replaces its old value.
Each key has one value at a time.

Python code

```
prices = {"tea": 6, "cake": 9}
prices["tea"] = 7
print(prices["tea"])
print(prices["cake"])
```

Expected output

```
7
9
```

Missing Keys

"juice" is not a key, so Python raises a KeyError.
The program stops unless the exception is handled.
Next: check whether the key exists before looking it up.

Python code

```
prices = {"tea": 6, "cake": 9}
print(prices["juice"])
```

Result: KeyError exception

```
KeyError: 'juice'
```

Checking a Key with in

Use in to check the keys. Put the lookup inside the if branch. Try tea, then try juice.

Python code

```
prices = {"tea": 6, "cake": 9}
item = input("Item: ")
if item in prices:
    print(prices[item])
else:
    print("Item not found")
```

Debugging: The Wrong Key

CORE EXERCISE

The check succeeds, but the lookup fails. Repair the lookup.

Python code

```
stock = {"pens": 8}
item = "pens"
if item in stock:
    print(stock["item"])
```

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Ask the user to enter an item. Print a clear message when the item is missing.

Missing Key: Debugging Solution

Use item without quotes. Its value is "pens".
With quotes, Python looks for a key literally named "item".

Python code

```
stock = {"pens": 8}
item = "pens"
if item in stock:
    print(stock[item])
```

Expected output

```
8
```

Practice: Update a Stock List

CORE EXERCISE

Start with 8 pens and 3 notebooks. Store the counts in a dictionary.

Add 5 rulers. Change pens to 6.

Ask for an item. Show its count or print "Item not found".

Try pens, rulers, and erasers.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Start with an empty dictionary. Add the three entries one at a time.

B. Allow another lookup until the user enters q.

Stock Lookup: Solution

Update the dictionary before asking which item to look up.

Python code

```
stock = {"pens": 8, "notebooks": 3}
stock["rulers"] = 5
stock["pens"] = 6

item = input("Item: ")
if item in stock:
    print(stock[item])
else:
    print("Item not found")
```

Looping over dictionaries

A for Loop Visits the Keys

Each time around the loop, item holds one key.
Use that key to look up the corresponding value.

Python code

```
prices = {"tea": 6,
          "fruit cup": 8,
          "sandwich": 15}

for item in prices:
    print(item)
```

Displaying Keys and Values

Use the loop variable twice: once as the item name and once as the key for the price lookup.

Python code

```
prices = {"tea": 6,
          "fruit cup": 8,
          "sandwich": 15}

for item in prices:
    print(item, prices[item], "yuan")
```

Menu Output

Expected output

```
tea 6 yuan
fruit cup 8 yuan
sandwich 15 yuan
```

Each line shows one item and its price.
Add another entry and the same loop will display it.

Practice: A Score Report

CORE EXERCISE

Store these scores in a dictionary: Mina: 8, Kai: 12, Noor: 5.

Use one loop to print each name and score.

Add Leo with 9 points. Run the same loop again.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Print only players with at least 9 points.

B. Use a running total to add all the scores.

C. Report the highest score and all names tied for it.

Score Report: Solution

The same loop prints all entries, including Leo.

Python code

```
scores = {"Mina": 8, "Kai": 12, "Noor": 5}
scores["Leo"] = 9

for name in scores:
    print(name, scores[name])
```

Independent Practice: Temperature Lookup

CORE EXERCISE

Make a dictionary: day names as keys, integers as values.
Monday: 21, Tuesday: 24, Wednesday: 19.

Use a for loop to display each pair.
Ask the user for a day name.
Print its temperature if it exists.
Otherwise print "Day not found".

Use the capitalization shown above.
Try Tuesday and Friday.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Add Thursday with a temperature of 22. Check that the original loop displays it without changes.

B. Count how many days have temperatures below 22.

Temperature Lookup: Solution

"Monday" is a literal key. choice is a variable holding the key.

Python code

```
temperatures = {
    "Monday": 21,
    "Tuesday": 24,
    "Wednesday": 19
}
for day in temperatures:
    print(day, temperatures[day])
choice = input("Day: ")
if choice in temperatures:
    print(temperatures[choice])
else:
    print("Day not found")
```

Dictionaries and lists

One Dictionary for One Product

A product can have several fields. Here, name is a string, and price and stock are integers.

Python code

```
product = {"name": "tea",
           "price": 6, "stock": 12}
print(product["name"])
print(product["stock"])
```

Expected output

```
tea
12
```

A List of Product Dictionaries

Use an index to choose the product.
Then use a key to choose a field from that product.

Python code

```
products = [
    {"name": "tea", "price": 6},
    {"name": "cake", "price": 9}
]

print(products[0]["name"])
print(products[1]["price"])
```

Expected output

```
tea
9
```

Looping Over Product Records

This time the loop visits a list.
Each product in the list is a dictionary.

Python code

```
products = [
    {"name": "tea", "price": 6, "stock": 12},
    {"name": "cake", "price": 9, "stock": 4}
]

for product in products:
    print(product["name"], product["stock"])
```

Updating a Dictionary Inside a List

products[0] selects the first dictionary. ["stock"] selects its stock. The assignment changes 12 to 3. Other values stay the same. Expected output: 3.

Python code

```
products = [
    {"name": "tea", "price": 6, "stock": 12},
    {"name": "cake", "price": 9, "stock": 4}
]

products[0]["stock"] = 3
print(products[0]["stock"])
```

Practice: Low Stock

CORE EXERCISE

Make a list of two product records:
tea: price 6, stock 12
cake: price 9, stock 4

Loop over the list. Print the name and stock for any product with fewer than 5 left.

Change tea stock to 3 and retest.

First test: cake 4.
After the change: tea 3 and cake 4.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Test what happens when stock is exactly 5.

B. Write a function that receives the list and returns how many products have low stock.

Low Stock: Solution (1 of 2)

Run the original inventory first. It prints cake 4.

Python code

```
products = [
    {"name": "tea", "price": 6, "stock": 12},
    {"name": "cake", "price": 9, "stock": 4}
]

for product in products:
    if product["stock"] < 5:
        print(product["name"], product["stock"])
```

Low Stock: Solution (2 of 2)

Add this code below Part 1.
The second check prints tea 3 and cake 4.

Python code

```
products[0]["stock"] = 3

for product in products:
    if product["stock"] < 5:
        print(product["name"], product["stock"])
```

A Dictionary Can Hold Lists

Here the arrangement is reversed: a dictionary holds lists. Choose a key first, then use an index in that list.

Python code

```
groups = {"red": ["Mina", "Kai"],
          "blue": ["Noor", "Leo"]}

print(groups["red"][0])
```

Expected output

```
Mina
```

Independent Check

1. How does a dictionary lookup differ from a list lookup?

2. Write Python code. Start with numbers = {"a": 5, "b": 10}. Change "a" to 8, then print the values of "a" and "b". Expected values: 8 and 10.

3. How do you avoid a missing-key error when using input()?

4. In products[0]["price"], what do 0 and "price" select?

Recap

A dictionary stores key–value pairs.

Use a key to look up or change a value. Check user input with in.

A for loop over a dictionary visits its keys.

A list can hold dictionaries. A dictionary can hold lists.

Homework: A Travel Time Lookup

CORE EXERCISE

Make a dictionary with travel times in minutes: bus 25, walk 45, bike 18.

Print all options with a loop.
Ask for a travel option.
Print its time or "Option not found".

Add train (12) and run it again.
Submit your .py file.

Test bike, train, and taxi.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Let the user look up another option until they enter q.

B. Ask for a maximum travel time.
Show all options within that time.