Python语言程序设计

Python Programming

Session 06



Tom Lotz (tom.lotz@outlook.com)

Content

	Review
01	Logical operators
02	Understanding while loops
03	Accumulators and repeated input
04	break and interactive programming

Review

True and False



- A comparison produces a Boolean value: True or False.

Python code

```
print(10 > 5)
print(10 < 5)
print(10 == 5)
```

Expected output

```
True
False
False
```

Comparison Operators



Operator	Meaning
==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

= assigns a value. == compares two values.

Exercise 1: Predict Each Result



- Write down all five results before running the code.

Python code

```
x = 8
y = 12

print(x > y)
print(x < y)
print(x == 8)
print(y != 12)
print(x >= 8)
```

The if Statement



- Python checks the condition. True runs the indented block.
- False skips the block. Code outside it continues normally.

Python code

```
x = 10

if x > 5:
    print("Large")

print("Finished")
```

Two Possible Outcomes



- True runs the if block. False runs the else block.
- Exactly one of these branches runs.
- else has a colon, but no condition.

Python code

```
x = 10

if x > 5:
    print("Large")
else:
    print("Small")
```

More Than Two Outcomes



- Python checks conditions from top to bottom.
- elif tests another condition if the earlier ones were False.
- After a match, Python skips the remaining branches.

Python code

```
x = 0

if x > 0:
    print("Positive")
elif x < 0:
    print("Negative")
else:
    print("Zero")
```

Logical Operators

and: Both Conditions

Both conditions must be True. Here, 15 is greater than 10 and less than 20.

Python code

```
x = 15

print(x > 10 and x < 20)
```

Expected output

```
True
```

or: At Least One Condition

At least one condition must be True. Both may be True. Here, neither condition is True.

Python code

```
x = 15

print(x < 10 or x > 20)
```

Expected output

```
False
```

not: Reverse the Result

not changes True to False and False to True.

Python code

```
x = 15

print(not x == 15)
print(not True)
print(not False)
```

Expected output

```
False
False
True
```

Logical Operators: Truth Table

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

not True is False. not False is True.
Use parentheses when combining several operators.

Practice: Boolean Results

CORE EXERCISE

Predict all four results before running.

Python code

```
a = 8
b = 15

print(a > 5 and b > 10)
print(a > 10 or b > 10)
print(not a == 8)
print(a < 5 or b < 10)
```

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Change a and b so all four expressions become False.

Is it possible? Explain why.

Boolean Results: Solution

Expected output

```
True
True
False
False
```

1. Both comparisons are True.
2. b > 10 is True.
3. a == 8 is True, so not gives False.
4. Neither comparison is True.

Optional challenge: a = 8 and b = 10 make all four expressions False. The boundary value 10 matters.

Practice: Conditions for Numbers

CORE EXERCISE

Start with x = 15.

1. Print "Inside" for 10 to 20, including both endpoints.

2. Print "Outside" below 10 or above 20.

3. Use not to print "Not equal to 15" when needed.

Test: 5, 10, 15, 20, 25.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Ask for two numbers.

Check whether both are positive.

Check whether at least one is negative.

Display a result for each check.

Conditions for Numbers: Solution

Python code

```
x = 15

if x >= 10 and x <= 20:
    print("Inside")
if x < 10 or x > 20:
    print("Outside")
if not x == 15:
    print("Not equal to 15")
```

10, 15, and 20 print Inside. 5 and 25 print Outside.
Every test value except 15 also prints Not equal to 15.

Understanding While Loops

The while Loop

while repeats an indented block as long as its condition is True.

Python code

```
x = 1

while x <= 5:
    print(x)
    x = x + 1

print("Finished")
```

Condition: x <= 5

Iteration: one repetition

Counter: x

Python checks the condition before every repetition.

Tracing a while Loop

Iteration	x before	x <= 5	x after
1	1	True	2
2	2	True	3
3	3	True	4
4	4	True	5
5	5	True	6
End	6	False	—

The loop prints 1, 2, 3, 4, and 5 on separate lines.
When x becomes 6, the loop ends. Then Finished prints once.

Zero Repetitions or One Repetition

Python code

```
x = 5
while x <= 5:
    print(x)
    x = x + 1
print("Finished")
```

Start at 5:
5
Finished

Start at 6:
Finished

The first check can already be False. In that case, the loop body never runs.

Practice: Predict a while Loop

CORE EXERCISE

Predict the output. Then change:

- Start x at 10.
- Start x at 2.
- Start at 0 and add 4 each time.

Python code

```
x = 0
while x < 10:
    print(x)
    x = x + 3
```

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Choose a starting value and a positive increment that make the loop run exactly twice. Explain your choice.

while Prediction: Solution

Version	Output, one value per line
x = 0, add 3	0, 3, 6, 9
x = 10, add 3	No output
x = 2, add 3	2, 5, 8
x = 0, add 4	0, 4, 8

Optional example: start at 4 and add 3.
The loop prints 4 and 7, then stops at x = 10.

Practice: Counters

CORE EXERCISE

Use while for each version:

1. Print 1 up to 10.

2. Print 10 down to 1.

3. Print even numbers from 2 to 20.

Print "Finished" once after each loop.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Print the first 10 multiples of 7.

B. Ask for a nonnegative starting number. Count down to zero.

C. Calculate 5 factorial:
5 × 4 × 3 × 2 × 1.

Counter Solutions: Up and Down

Python code

```
x = 1
while x <= 10:
    print(x)
    x = x + 1
print("Finished")
```

Python code

```
x = 10
while x >= 1:
    print(x)
    x = x - 1
print("Finished")
```

The update moves x toward a value that makes the condition False.

Counter Solution: Even Numbers

Python code

```
x = 2
while x <= 20:
    print(x)
    x = x + 2
print("Finished")
```

Output: 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, then Finished.
Each number prints on its own line.

Debugging: A Loop That Never Stops

CORE EXERCISE

What happens? Why does it never stop?

Repair the program.

Code to repair

```
x = 1
while x <= 5:
    print(x)
```

In IDLE: Ctrl+C interrupts execution.
Use it if you run an infinite loop.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Make a loop that prints 1 to 5 without changing the initial value x = 1. Use a different stopping condition.

Debugging: The Wrong Direction

CORE EXERCISE

Why does this loop never stop?
Correct the update.

Code to repair

```
x = 1
while x <= 5:
    print(x)
    x = x - 1
```

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

Keep x = x - 1. Change the start and condition so the loop prints 5 down to 1 and stops.

Infinite Loops: Corrected Solution

Python code

```
x = 1
while x <= 5:
    print(x)
    x = x + 1
```

Both broken programs can use this correction.

The update must eventually make the condition False.
Check the start, update, and stopping condition together.

Accumulators and Repeated Input

A Running Total

A counter controls repetition. An accumulator stores a running total.

Python code

```
total = 0
x = 1

while x <= 5:
    total = total + x
    x = x + 1

print(total)
```

x is the counter.

total is the accumulator.

Each update adds x to the previous total.

Expected output: 15

Tracing Two Variables

Iteration	x added	total before	total after
1	1	0	1
2	2	1	3
3	3	3	6
4	4	6	10
5	5	10	15

Initialize total once before the loop.
Resetting total inside the loop would lose the earlier sum.

Practice: Sum of Numbers

CORE EXERCISE

Use a while loop to add the numbers from 1 to 10.

Display the final sum.

Check: core 55
A: 110 B: 385

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Add the even numbers from 2 to 20.

B. Add the squares from 1 to 10.

Sum of Numbers: Solution

Python code

```
total = 0
x = 1

while x <= 10:
    total = total + x
    x = x + 1

print(total)
```

Expected output: 55.

Optional A adds 2 at each step. Optional B adds x * x to total.

Repeated Input: Zero to Stop

Python code

```
total = 0
number = int(input("Number (0 to stop): "))

while number != 0:
    total = total + number
    number = int(input("Number (0 to stop): "))

print("Total:", total)
```

The user controls the number of repetitions.
0 is the stopping value, also called a sentinel.
Read a new value inside the loop. The stopping value is excluded.

Testing Repeated Input

Input sequence	Expected final output
5, 8, -3, 0	Total: 10
0	Total: 0
7, 0	Total: 7
-4, -2, 0	Total: -6

A first input of 0 skips the loop body.
Negative integers are valid values for this program.

Practice — Input Until –1

CORE EXERCISE

1. Repeatedly ask for an integer.
2. Stop when the user enters -1.
3. Add all numbers before -1.
4. Display the total.
5. Zero is valid and must not stop the program.

Tests (inputs → output)
5, 8, 0, 3, -1 → Total: 16
-1 → Total: 0
-4, 6, -1 → Total: 2

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Count every input except -1.

B. Calculate their average.

C. Display the largest value.

D. Handle -1 as the first input without an incorrect average or largest value.

Repeated Input: Core Solution

Python code

```
total = 0
number = int(input("Number (-1 to stop): "))

while number != -1:
    total = total + number
    number = int(input("Number (-1 to stop): "))

print("Total:", total)
```

The new input replaces number before the next condition check.

Optional Solution: Tracking More Values

Part 1 of 2. Keep both parts in the same program.

Python code

```
total = 0
count = 0
largest = 0
number = int(input("Number (-1 to stop): "))
while number != -1:
    total = total + number
    if count == 0 or number > largest:
        largest = number
    count = count + 1
    number = int(input("Number (-1 to stop): "))
```

Optional Solution: Reporting the Results

Part 2 of 2. Add this after the loop, with no indentation.

Python code

```
print("Total:", total)
print("Count:", count)
if count > 0:
    print("Average:", total / count)
    print("Largest:", largest)
else:
    print("No numbers entered")
```

count == 0 accepts the first value as largest, even if zero or negative. An immediate -1 gives no average or largest value and avoids division by zero.

Simple Validation: An Accepted Range

Ask again until the user enters an integer from 1 to 5.

Python code

```
choice = int(input("Choose 1 to 5: "))

while choice < 1 or choice > 5:
    print("Please enter 1 to 5")
    choice = int(input("Choose 1 to 5: "))

print("Accepted:", choice)
```

Test: 0, 6, then 3. Also test 1 and 5. Enter integers for now.
This checks the numerical range, not text such as "hello".

Try It Yourself: Validation

Modify the previous program to accept integers from -3 to 3, including both endpoints. Keep asking until the input is valid.

TEST YOUR PROGRAM

-4: Rejected
-3: Accepted
 0: Accepted
 3: Accepted
 4: Rejected

break and Interactive Programming

Stopping with break

Python code

```
while True:
    command = input("Text (q to quit): ")
    if command == "q":
        break
    print(command)

print("Finished")
```

while True keeps repeating unless something stops it.
break exits the nearest enclosing loop immediately.
The code after the loop still runs.

Two Ways to Stop a Loop

A condition becomes False

The loop checks before each repetition.

When the user enters 0, number != 0 becomes False.

Python skips the loop body.

break exits immediately

The while condition can stay True.

An if statement checks for q.

When break runs, Python leaves the loop at that point.

Repeating Calculator: Expected Behavior

Extend the Session 05 calculator so the user can calculate again.

Example interaction

```
First number (q to quit): 12
Second number: 5
Operation (+, -, *): *
Result: 60

First number (q to quit): q
Goodbye
```

Independent Practice: Repeating Calculator

CORE EXERCISE

1. Ask for the first number or q.
2. If it is q, stop with break.
3. Otherwise, convert it to int.
4. Ask for a second integer.
5. Ask for +, -, or *.
6. Use if/elif/else to calculate.
7. Show the result, or "Unknown operation".
8. Repeat using while True.
9. After quitting, print "Goodbye".

Example: 12, 5, * gives 60.
Then q ends with Goodbye.

OPTIONAL CHALLENGES

Finish the core first.
Choose any challenge.

A. Count successful calculations. Show the count when quitting.

B. Add successful results to a running total. Show it on exit.

C. Store successful results in a list. Display the history on exit.

D. Accept decimals with float().

E. Add division and prevent division by zero. Optionally add **.

Optional Hint: Repeating Calculator

- Put the repeated steps inside a while True loop.
- Read the first input as text.
- Check for q before converting it to an integer.
- Place the existing calculator after the quitting check.
- Put Goodbye after the loop.

Repeating Calculator: Solution, Part 1

One program across two slides. Continue the loop on the next slide.

Python code

```
while True:
    first = input("First number (q to quit): ")
    if first == "q":
        break
    a = int(first)
    b = int(input("Second number: "))
    operation = input("Operation (+, -, *): ")
```

Repeating Calculator: Solution, Part 2

Continue inside the same loop. Goodbye has no indentation.

Python code

```
    if operation == "+":
        print("Result:", a + b)
    elif operation == "-":
        print("Result:", a - b)
    elif operation == "*":
        print("Result:", a * b)
    else:
        print("Unknown operation")
print("Goodbye")
```

Understanding the Repeating Calculator

- The Session 05 calculator performed one calculation.
- The loop repeats the input and calculation steps.
- first stores text, so it can hold a number or "q".
- int(first) runs only after checking for "q".
- An unknown operation shows a message, then repeats.
- break stops the loop. Goodbye prints once.

Testing the Calculator

Test	Expected behavior
q immediately	Goodbye, with no conversion error
12, 5, * then q	Result: 60, then Goodbye
-4, 2, + then q	Result: -2, then Goodbye
0, 7, * then q	Result: 0, then Goodbye
12, 5, ? then q	Unknown operation, then Goodbye

Also perform two calculations before quitting.
Use integer input in the core version. Arbitrary text handling comes later.

The Final Project

- Submit a documented Python program.
- Choose a small program you can explain and demonstrate.
- Start thinking about a possible topic now.
- We will add functions, dictionaries, objects, and files.
- Weeks 15–16 provide time to develop and finish your project.

Independent Check: Questions 1 and 2

1. Predict the complete output.

Python code

```
x = 1
while x < 5:
    print(x)
    x = x + 2
```

2. Why does this loop never end?

Python code

```
x = 0
while x < 10:
    print(x)
```

Independent Check: Questions 3 and 4

3. What is the difference between break and a while condition
   becoming False?

4. What does an accumulator do?

Recap

- and, or, not combine or reverse conditions.
- while repeats while its condition is True.
- A counter tracks repetitions or progress.
- An accumulator stores a running result.
- Repeated input updates a value in the loop.
- A stopping value can end repetition.
- break leaves a loop immediately.
- Check updates to avoid infinite loops.

Next session: functions

Homework: Number Summary

CORE TASK

1. Ask for an integer, or q to quit.
2. Repeat until q is entered.
3. Add the numbers to a total.
4. Count the positive, negative, and zero values separately.
5. Display the total and all three counts.

Use while, if/elif/else, and break.

Submit your .py file.
Be ready to explain your code.

TEST YOUR PROGRAM

5, -2, 0, 7, q
Total: 10
Positive: 2, Negative: 1, Zero: 1

q immediately: total and counts = 0.

OPTIONAL CHALLENGE

Display the average and largest value.
Handle q before any number.