

Chapter 4: Loops (4.1 – 4.3)

PART1: THE WHILE LOOP, PROBLEM SOLVING:
HAND TRACING, PROCESSING SENTINEL VALUES

Chapter 3-Part1: The While loop

- **Goals**
 - To implement while loop
 - To hand-trace the execution of a program
- **Contents:**
 - The **while** loop
 - Problem Solving: Hand-Tracing
 - Application: Processing Sentinels



The while Loop

- Examples of loop applications
 - Calculating compound interest
 - Simulations, event driven programs
 - Drawing tiles...
- Compound interest algorithm (Chapter 1)

Start with a year value of 0, a column for the interest, and a balance of \$10,000.

year	interest	balance
0		\$10,000

Repeat the following steps while the balance is less than \$20,000.

Steps

Add 1 to the year value.

Compute the interest as $\text{balance} \times 0.05$ (i.e, 5 percent interest).

Add the interest to the balance.

Report the final year value as the answer.



Code Academy

Planning the while Loop

RATE=5

balance = 10000.0

TARGET= 20000.0

year = 0

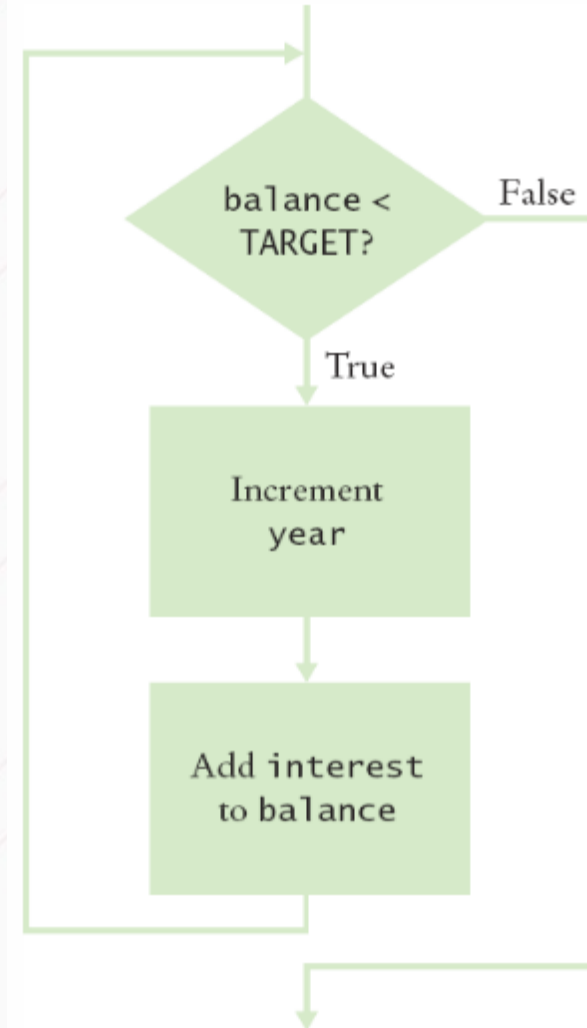
while balance < TARGET :

 year = year + 1

 interest = balance * RATE/100

 balance = balance + interest

A loop executes instructions repeatedly while a condition is True.





Code Academy

Syntax: `while` Statement

This variable is initialized outside the loop and updated in the loop.

If the condition never becomes false, an infinite loop occurs.



See page 161.

```
balance = 10000.0
```

```
.  
. .  
. .
```

```
while balance < TARGET :
```

```
    interest = balance * RATE / 100
```

```
    balance = balance + interest
```

Beware of "off-by-one" errors in the loop condition.



See page 161.

Put a colon here!
See page 95.

Statements in the body of a compound statement must be indented to the same column position.
See page 95.

These statements are executed while the condition is true.



Execution of the Loop

1 Check the loop condition

balance = 10000.0

year = 0

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100  
    balance = balance + interest
```

The condition is true

2 Execute the statements in the loop

balance = 10500.0

year = 1

interest = 500.0

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100  
    balance = balance + interest
```

3 Check the loop condition again

balance = 10500.0

year = 1

interest = 500.0

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100  
    balance = balance + interest
```

The condition is still true



Execution of the Loop (2)

4 After 15 iterations

balance = 20789.28

year = 15

interest = 989.97

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100  
    balance = balance + interest
```

The condition is no longer true

5 Execute the statement following the loop

balance = 20789.28

year = 15

interest = 989.97

```
while balance < TARGET :  
    year = year + 1  
    interest = balance * RATE / 100  
    balance = balance + interest
```

```
print(year)
```



Code Academy

Investment Example: Doubleinv.py

```
1  ##
2  # This program computes the time required to double an investment.
3  #
4
5  # Create constant variables.
6  RATE = 5.0
7  INITIAL_BALANCE = 10000.0
8  TARGET = 2 * INITIAL_BALANCE
9
10 # Initialize variables used with the loop.
11 balance = INITIAL_BALANCE
12 year = 0
13
14 # Count the years required for the investment to double.
15 while balance < TARGET :
16     year = year + 1
17     interest = balance * RATE / 100
18     balance = balance + interest
19
20 # Print the results.
21 print("The investment doubled after", year, "years.")
```

Declare and initialize a variable outside of the loop to count `year`

Increment the `year` variable each time through

Count-Controlled Loops

- A **while** loop that is controlled by a counter

```
counter = 1                                # Initialize the counter
while counter <= 10 :                       # Check the counter
    print(counter)                          # Update the loop variable
    counter = counter + 1
```

- **Count-Controlled Loop Example:** Sum the numbers from 1 to 10

```
counter = 1
sum = 0
while counter <= 10 :
    sum = sum + counter
    counter = counter + 1
print(sum)
```



Code Academy

Event-Controlled Loops

- A **while** loop that is controlled by an event variable.

```
balance = INITIAL_BALANCE
```

```
# Initialize the loop variable
```

```
year = 0
```

```
while balance <= TARGET:
```

```
# Check the loop variable
```

```
    year = year + 1
```

```
    balance = balance * 2
```

```
# Update the loop variable
```



Code Academy

while Loop Examples

Loop	Output	Explanation
<pre>i = 0 total = 0 while total < 10 : i = i + 1 total = total + i print(i, total)</pre>	<pre>1 1 2 3 3 6 4 10</pre>	When total is 10, the loop condition is false, and the loop ends.
<pre>i = 0 total = 0 while total < 10 : i = i + 1 total = total - 1 print(i, total)</pre>	<pre>1 -1 2 -3 3 -6 4 -10 . . .</pre>	Because total never reaches 10, this is an “infinite loop” (see Common Error 4.2 on page 161).
<pre>i = 0 total = 0 while total < 0 : i = i + 1 total = total - i print(i, total)</pre>	(No output)	The statement total < 0 is false when the condition is first checked, and the loop is never executed.



while Loop Examples (2)

Loop	Output	Explanation
<pre>i = 0 total = 0 while total >= 10 : i = i + 1 total = total + i print(i, total)</pre>	(No output)	The programmer probably thought, "Stop when the sum is at least 10." However, the loop condition controls when the loop is executed, not when it ends (see Common Error 4.2 on page 161).
<pre>i = 0 total = 0 while total >= 0 : i = i + 1 total = total + i print(i, total)</pre>	(No output, program does not terminate)	Because total will always be greater than or equal to 0, the loop runs forever. It produces no output because the print function is outside the body of the loop, as indicated by the indentation.

Common Error: Incorrect Test Condition

- The loop body will only execute if the test condition is **True**.
- If bal is initialized as less than the TARGET and should grow until it reaches TARGET
 - Which version will execute the loop body?

```
while bal >= TARGET :  
    year = year + 1  
    interest = bal * RATE  
    bal = bal + interest
```

```
while bal < TARGET :  
    year = year + 1  
    interest = bal * RATE  
    bal = bal + interest
```



Code Academy

Common Error: Infinite Loops

- The loop body will execute until the test condition becomes False.
- What if you forget to update the test variable?
 - bal is the test variable (TARGET doesn't change)
 - You will loop forever! (or until you stop the program)

```
while bal < TARGET :  
    year = year + 1  
    interest = bal * RATE  
    bal = bal + interest
```


Common Error: Off-by-One Errors

- A 'counter' variable is often used in the test condition
- Your counter can start at 0 or 1, but programmers often start a counter at 0
- If I want to paint all 5 fingers on one hand, when I am done?
 - If you start at 0, use "<"
0, 1, 2, 3, 4
 - If you start at 1, use "<="
1, 2, 3, 4, 5

```
finger = 0
FINGERS = 5
while finger < FINGERS :
    # paint finger
    finger = finger + 1
```

```
finger = 1
FINGERS = 5
while finger <= FINGERS :
    # paint finger
    finger = finger + 1
```



Code Academy

Hand-Tracing Loops

- Example: Calculate the sum of digits (1+7+2+9)
 - Make columns for key variables (n, total, digit)
 - Examine the code and number the steps
 - Set variables to state before loop begins

n	total	digit
1729	0	



Code Academy

Tracing Sum of Digits

n	total	digit
1729	0	



```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

- Start executing loop body statements changing variable values on a new line
 - Cross out values in previous line



Code Academy

Tracing Sum of Digits


n	total	digit
1729	0	
	9	9

```
n = 1729
```

```
total = 0
```

```
while n > 0 :
```

```
    digit = n % 10
```

```
     total = total + digit
```

```
    n = n // 10
```

```
print(total)
```

- Continue executing loop statements changing variables
 - 1729 / 10 leaves 172 (no remainder)



Code Academy

Tracing Sum of Digits

- Test condition. If True, execute loop again
 - Variable n is 172, Is $172 > 0$?, True!
- Make a new line for the second time through and update variables

n	total	digit
1729	0	
172	9	9
17	11	2



```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```



Code Academy

Tracing Sum of Digits

- Third time through
 - Variable n is 17 which is still greater than 0
- Execute loop statements and update variables

n	total	digit
1729	0	
172	9	9
17	11	2
1	18	7

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```




Tracing Sum of Digits

- Fourth loop iteration:
 - Variable n is 1 at start of loop. $1 > 0$? True
 - Executes loop and changes variable n to 0 ($1/10 = 0$)

n	total	digit
1729	0	
172	9	9
17	11	2
1	18	7
0	19	1



```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```



Code Academy

Tracing Sum of Digits

- Because n is 0, the expression(n > 0) is False
- Loop body is not executed
 - Jumps to next statement after the loop body
- Finally prints the sum!

n	total	digit	output
1729	0		
172	9	9	
17	11	2	
7	18	7	
0	19	1	19



```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```



Sentinel-controlled While loop

- Sentinel values are often used:
 - When you don't know how many items are in a list, use a 'special' character or value to signal the "last" item
 - For numeric input of positive numbers, it is common to use the value -1

A sentinel value denotes the end of a data set, but it is not part of the data.

```
salary = 0.0
while salary >= 0 :
    salary = float(input("Enter a salary or -1 to finish: "))
    if salary >= 0.0 :
        total = total + salary
        count = count + 1
```


Averaging a Set of Values

- Declare and initialize a 'total' variable to 0
- Declare and initialize a 'count' variable to 0
- Declare and initialize a 'salary' variable to 0
- Prompt user with instructions
- Loop until sentinel value is entered
 - Save entered value to input variable ('salary')
 - If salary is not -1 or less (sentinel value)
 - Add salary variable to total variable
 - Add 1 to count variable
- Make sure you have at least one entry before you divide!
 - Divide total by count and output.
 - Done!



Sentinel.py (1)

```
5 # Initialize variables to maintain the running total and count.
```

```
6 total = 0.0
```

```
7 count = 0
```

Outside the while loop: declare and initialize variables to use

```
8
```

```
9 # Initialize salary to any non-sentinel value.
```

```
10 salary = 0.0
```

```
13 while salary >= 0.0 :
```

Since salary is initialized to 0, the while loop statements will execute at least once

```
14     salary = float(input("Enter a salary or -1 to finish: "))
```

```
15     if salary >= 0.0 :
```

Input new salary and compare to sentinel

```
16         total = total + salary
```

```
17         count = count + 1
```

Update running total and count (to calculate the average later)



Sentinel.py (2)

```
19 # Compute and print the average salary.
20 if count > 0 :
21     average = total / count
22     print("Average salary is", average)
```

Prevent divide by 0

Calculate and output the average salary using the total and count variables

```
23 else :
24     print("No data was entered.")
```

Program run:

```
Enter a salary or -1 to finish: 100
Enter a salary or -1 to finish: 200
Enter a salary or -1 to finish: 300
Enter a salary or -1 to finish: -1
Average salary is 200.0
```


Priming Read and Modification Read

- Some programmers don't like the "trick" of initializing the input variable with a value other than a sentinel.

```
# Set salary to a value to ensure that the loop
# executes at least once.
salary = 0.0
while salary >= 0 :
```

- An alternative is to change the variable with a priming read before the loop. A modification read at the bottom of the loop is used to obtain the next input. ([see sentinel2.py for full code](#))

```
# Priming read
salary = float(input("Enter a salary or -1 to finish: "))
while salary >= 0.0 :
    total = total + salary
    count = count + 1
    # Modification read
    salary = float(input("Enter a salary or -1 to finish:"))
```



Sentinel2.py:

```
# Prints the average of salary of values terminated with a sentinel.

# Initialize variables to maintain the running total and count.
total = 0.0
count = 0
# Priming read
salary = float(input("Enter a salary or -1 to finish: "))
while salary >= 0.0 :
    total = total + salary
    count = count + 1
    # Modification read
    salary = float(input("Enter a salary or -1 to finish:"))

# Compute and print the average salary.
if count > 0 :
    average = total / count
    print("Average salary is", average)
else :
    print("No data was entered.")
```



Code Academy

Boolean Variables and Sentinels

- A boolean variable can be used to control a loop
 - Sometimes called a 'flag' variable

```
done = False
while not done :
    value = float(input("Enter a salary or -1 to finish: "))
    if value < 0.0:
        done = True
    else :
        # Process value
```

Initialize done so that the loop will execute

Set done 'flag' to True if sentinel value is found

Summary

- **while** loops are very common
- Initialize variables before you test
 - The condition is tested BEFORE the loop body
 - This is called pre-test
 - The condition often uses a counter variable
 - Something inside the loop should change one of the variables used in the test
- Watch out for infinite loops!
- A **sentinel** value denotes the end of a data set, but it is not part of the data.
- You can use a boolean variable to control a loop.
 - Set the variable to **True** before entering the loop
 - Set it to **False** to leave the loop.