Chapter 4: Loops

Chapter Goals

- To implement while and for loops
- To hand-trace the execution of a program
- To become familiar with common loop algorithms
- To understand nested loops
- To implement programs that read and process data sets
- To use a computer for simulations

In this chapter, you will learn about loop statements in Python, as well as techniques for writing programs that simulate activities in the real world.

Contents

The **while** Loop

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

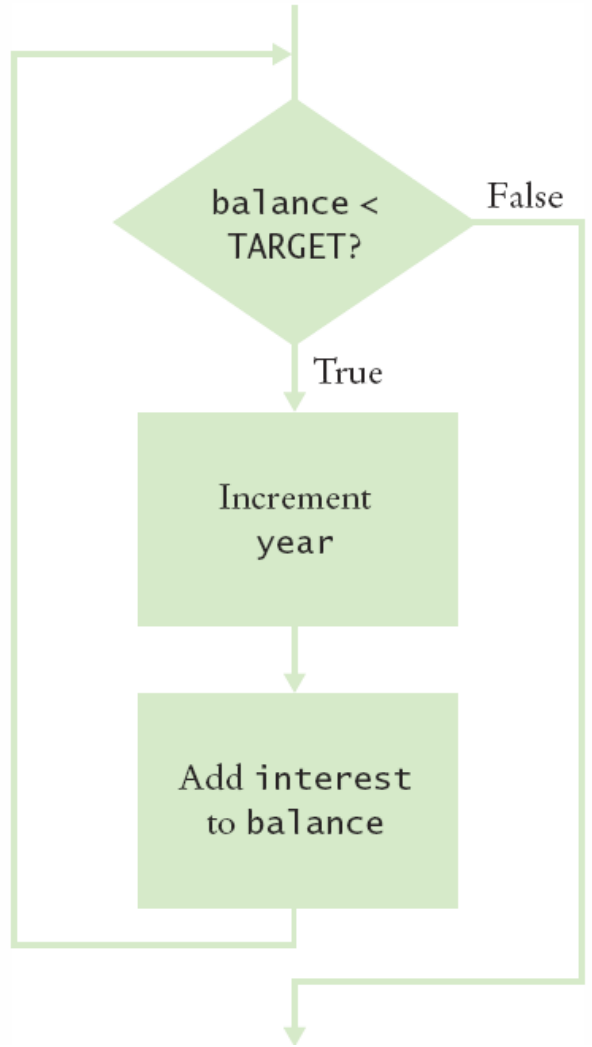
- Add 1 to the year value.
- Compute the interest as balance x 0.05 (i.e, 5 percent interest).
- Add the interest to the balance.

Report the final year value as the answer.

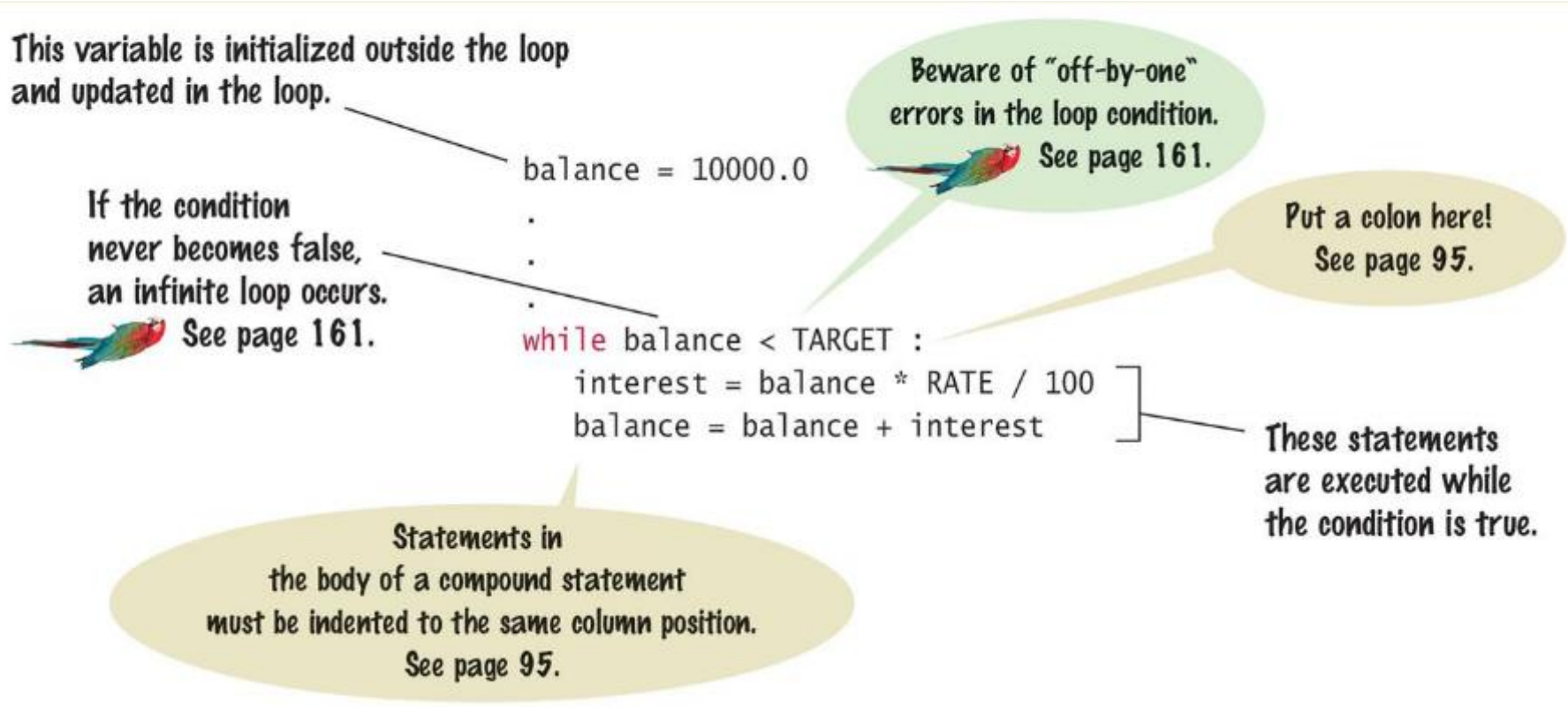
Planning the **while** Loop

```
balance = 10.0
target = 100.0
year = 0
rate = 0.025
while balance < TARGET :
   year = year + 1
   interest = balance * RATE/100
   balance = balance + interest
```

A loop executes instructions repeatedly while a condition is True.



Syntax: `while` Statement



Count-Controlled Loops

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

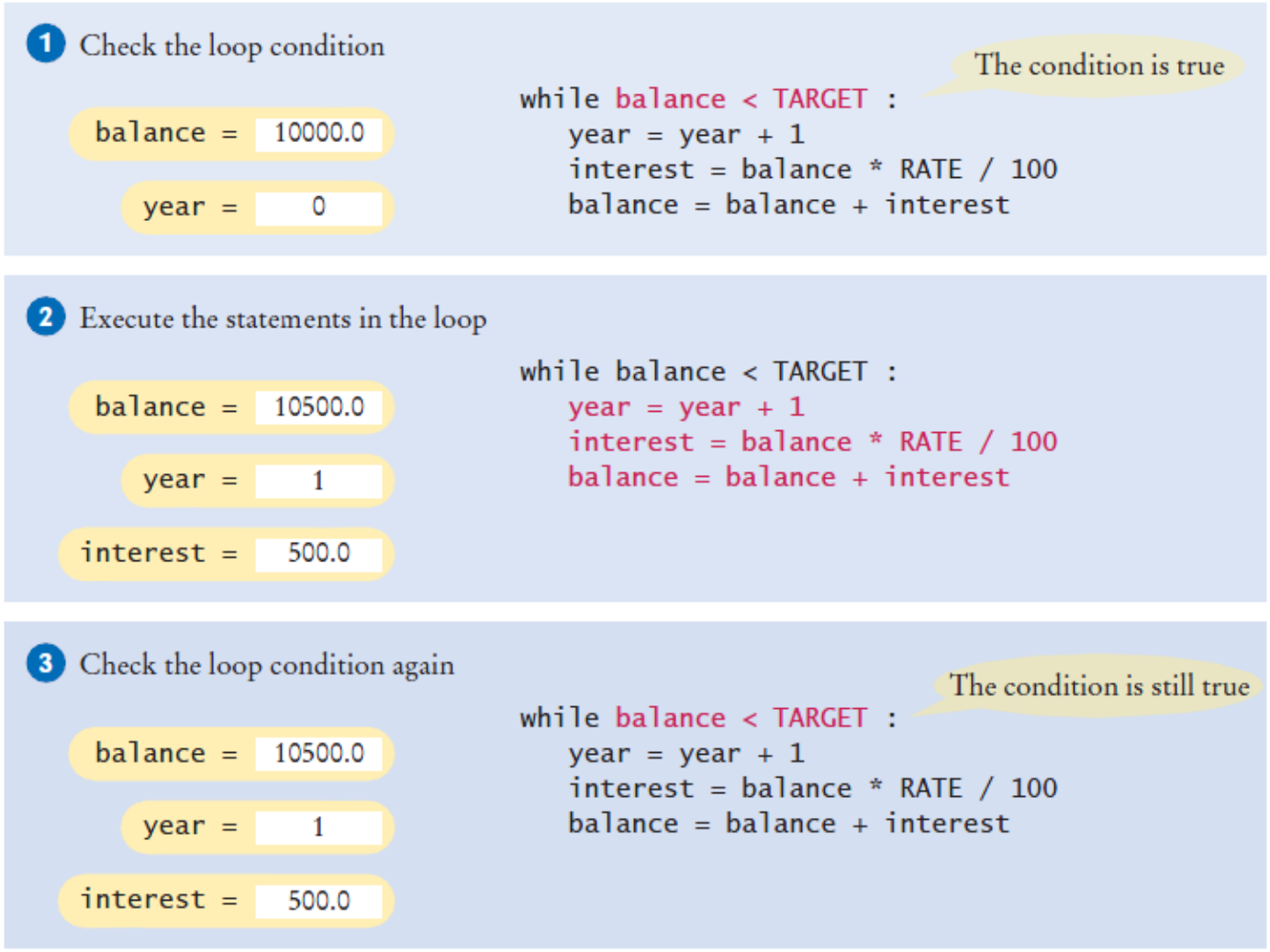
```
counter = 1                                    # Initialize
the counter

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

Event-Controlled Loops

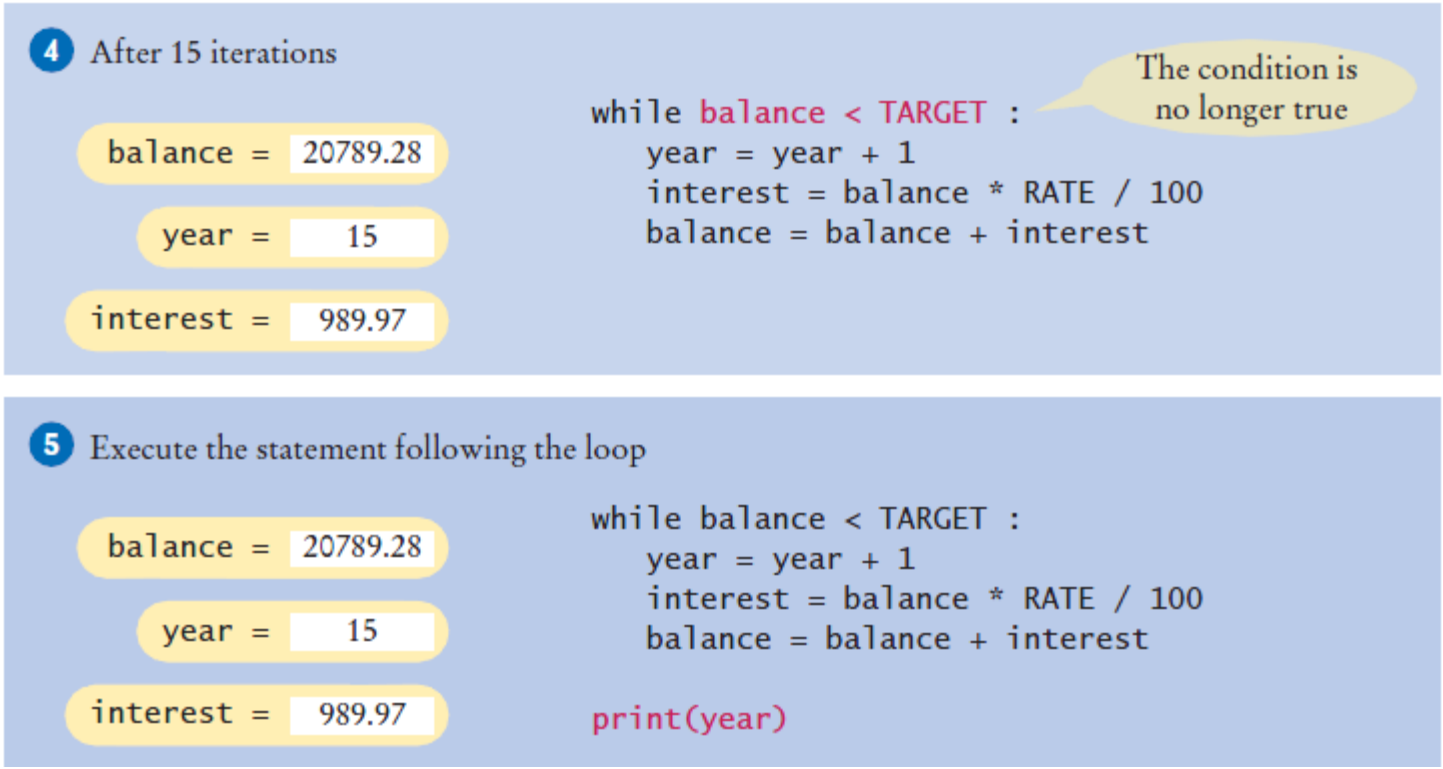
- A **while** loop that is controlled by an event

```
balance = INITIAL_BALANCE  # Initialize the loop variable
while balance <= TARGET:   # Check the loop variable
  year = year + 1
  balance = balance * 2    # Update the loop variable
```

Execution of the Loop



Execution of the Loop (2)



Doubleinv.py

ch04/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

`while` Loop Examples

Loop	Output	Explanation

`i = 0`
`total = 0`
`while total < 10 :`
`  i = i + 1`
`  total = total + i`
`  print(i, total)`
	`1 1`
	`2 3`
	`3 6`
	`4 10`
		When `total` is 10, the loop condition is false, and the loop ends.

`i = 0`
`total = 0`
`while total < 10 :`
`  i = i + 1`
`  total = total - 1`
`  print(i, total)`
	`1 -1`
	`2 -3`
	`3 -6`
	`4 -10`
	`. . .`
		Because `total` never reaches 10, this is an "infinite loop" (see Common Error 4.2 on page 161).

`i = 0`
`total = 0`
`while total < 0 :`
`  i = i + 1`
`  total = total - i`
`  print(i, total)`
	(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

`i = 0`
`total = 0`
`while total >= 10 :`
`  i = i + 1`
`  total = total + i`
`  print(i, total)`
	(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).

`i = 0`
`total = 0`
`while total >= 0 :`
`  i = i + 1`
`  total = total + i`
`print(i, total)`
	(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
```

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
```

Hand Tracing Loops

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	

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

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)

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)
```

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)

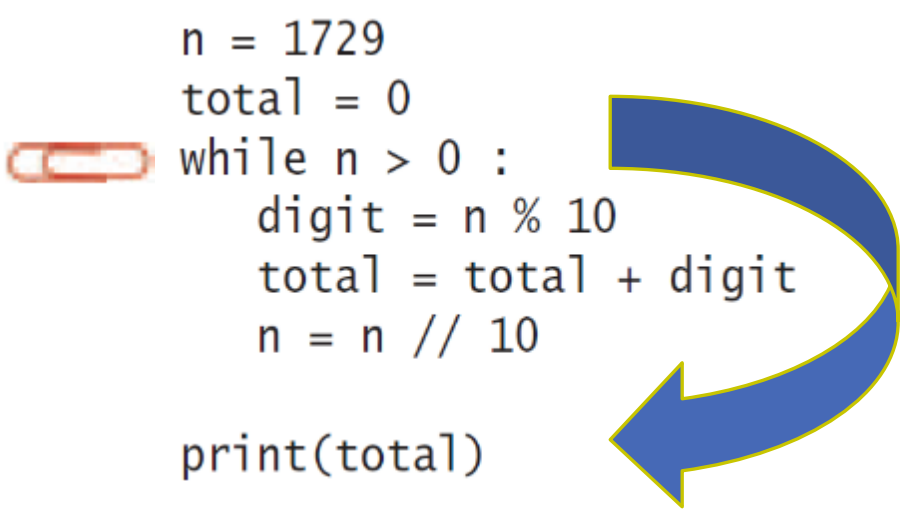
n	total	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
~~1~~	~~18~~	~~7~~
0	19	1

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

print(total)
```

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~~	
~~1~~	~~18~~	~~7~~	
0	19	1	19



Summary of the **while** Loop

- `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!

Investment Example

- Open the file:
  - Doubleinv.py
- Run the program with several test cases
  - The program will prompt you for a rate
  - Enter a mix of valid and invalid rates

Sentinel Values

Processing Sentinel Values

- 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())
      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
 8
 9  # Initialize salary to any non-sentinel value.
10  salary = 0.0

13  while salary >= 0.0 :

14      salary = float(input("Enter a salary or -1 to finish: "))
15      if salary >= 0.0 :

16          total = total + salary
17          count = count + 1
```

Outside the while loop: declare and initialize variables to use

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

Input new `salary` and compare to sentinel

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)

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

Prevent divide by 0

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

Program Run

```
Enter salaries, -1 to finish: 10 10 40 -1
Average salary: 20
```

Sentinel Example

- Open the file:
  - Sentinal.py
- Notice the use of the **IF()** test inside the **while** loop
  - The IF() checks to make sure we are not processing the sentinel value

Priming 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 read before the loop.

```
salary = float(input("Enter a salary or -1 to finish: "))
while salary >= 0 :
```

Modification Read

- The input operation at the bottom of the loop is used to obtain the next input.

```
# 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:
    "))
```

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

Storyboards

Storyboards

- One useful problem solving technique is the use of storyboards to model user interaction. It can help answer:
  - What information does the user provide, and in which order?
  - What information will your program display, and in which format?
  - What should happen when there is an error?
  - When does the program quit?

- ***A storyboard consists of annotated sketches for each step in an action sequence.***

Storyboard Example

- Goal: Converting a sequence of values
  - Will require a loop and some variables
  - Handle one conversion each time through the loop

Converting a Sequence of Values

What unit do you want to convert from? cm
What unit do you want to convert to? in
Enter values, terminated by zero
30
30 cm = 11.81 in
100
100 cm = 39.37 in
0
What unit do you want to convert from?

Allows conversion of multiple values

Format makes clear what got converted

What Can Go Wrong?

Unknown unit types

- What is the user misspells centimeters and inches?
- What other conversions are available?
- Solution:
  - Show a list of the acceptable unit types

From unit (in, ft, mi, mm, cm, m, km, oz, lb, g, kg, tsp, tbsp, pint, gal): cm
To unit: in

No need to list the units again

What Else Can Go Wrong?

- How does the user quit the program?

Exiting the Program

```
From unit (in, ft, mi, mm, cm, m, km, oz, lb, g, kg, tsp, tbsp, pint, gal): cm
To unit: in
Enter values, terminated by zero
30
30 cm = 11.81 in
0
More conversions (y, n)? n
(Program exits)
```

Sentinel triggers the prompt to exit