Chapter 3

How to code control statements

Applied objectives

1. Code, test, and debug programs that require the skills that you've learned in this chapter. That includes the use of:

   if statements
   while statements
   for statements
   break and continue statements
   pass statements
   assignment expressions

2. Use pseudocode to plan your control structures and programs.

Knowledge objectives (part 1)

1. Distinguish between a Boolean variable and a Boolean expression.
2. Describe the evaluation of a Boolean expression, including order of precedence and the use of parentheses.
3. Describe the sort sequence of string values and the use of the lower() or upper() method of a string for comparing two string values.
4. Describe the flow of control of an if statement that has both elif and else clauses.
5. Distinguish between the flow of control in a while loop and the flow of control in a for loop.
6. Describe the use of break and continue statements.

Knowledge objectives (part 2)

7. Explain how a while statement with an assignment expression can be used to replace a while statement with an infinite loop.
8. Describe the use of pseudocode for planning a program and its control statements.

Relational operators

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

Boolean expressions

```
age == 5                  # variable equal to numeric literal
first_name == "John"      # variable equal to string literal

quantity != 0             # variable not equal to numeric literal

distance > 5.6            # variable greater than numeric literal
fuel_req < fuel_cap       # variable less than variable

distance >= limit         # variable greater than or equal to variable
stock <= reorder_point    # variable less than or equal to variable

rate / 100 >= 0.1        # expression greater than or equal to literal
```

How to assign a Boolean value to a variable

```
active = True             # variable is set to Boolean True value
active = False            # variable is set to Boolean False value
```

Logical operators

Operator	Name
and	AND
or	OR
not	NOT

Order of precedence

1. NOT operator
2. AND operator
3. OR operator

Boolean expressions that use logical operators

```
# The AND operator
age >= 65 and city == "Chicago"

# The OR operator
city == "Greenville" or age >= 65

# The NOT operator
not age >= 65

# Two AND operators
age >= 65 and city == "Greenville" and state == "SC"

# Two OR operators
age >= 65 or age <= 18 or status == "retired"

# AND and OR operators with parens to clarify sequence of operations
(age >= 65 and status == "retired") or age < 18

# AND and OR operators with parens to change sequence of operations
age >= 65 and (status == "retired" or state == "SC")
```

Some string comparisons

Condition	Boolean result
`"apple" < "Apple"`	`False`
`"App" < "Apple"`	`True`
`"1" < "5"`	`True`
`"10" < "5"`	`True`

The sort sequence of digits and letters

- Digits from 0-9
- Uppercase letters from A-Z
- Lowercase letters from a-z

Two string methods

```
lower()
upper()
```

How to call a string method

```
variableName.methodName()
```

How to compare strings with the lower() method

```
string1 = "Mary"
string2 = "mary"
string1 == string2                    # False
string1.lower() == string2.lower()    # True
print(string1)                        # prints 'Mary'
print(string2)                        # prints 'mary'
```

How the lower() method can simplify code

Without the lower() method

```
customer_type == "r" or customer_type == "R"
```

With the lower() method

```
customer_type.lower() == "r"
```

The syntax of the if statement

```
if boolean_expression:
    statements...
[elif boolean_expression:
    statements...]...
[else:
    statements...]
```

Only an if clause

```
if age >= 18:
    print("You may vote.")
```

An if clause and an else clause

```
if age >= 18:
    print("You may vote.")
else:
    print("You are too young to vote.")
```

An if clause, two elif clauses, and an else clause

```
if invoice_total >= 500:
    discount_percent = .2
elif invoice_total >= 250:
    discount_percent = .1
elif invoice_total > 0:
    discount_percent = 0
else:
    print("Invoice total must be greater than zero.")
```

The operation of an if statement

- An *if statement* always contains an *if clause*. In addition, it may contain one or more *elif clauses* and one *else clause*.
- When an if statement is executed, the condition in theif clause is evaluated first. If it is true, the statements in this clause are executed and the if statement ends. Otherwise, the condition in the first elif (else if) clause is evaluated.
- If the condition in the first elif clause is true, the statements in this clause are executed. Otherwise, the condition in the next elif clause is evaluated. This continues until the condition in one of the elif clauses is true or the else clause is reached.
- The statements in the else clause are executed if the conditions in all of the preceding clauses are false.
- Only one block of statements can be run each time an if statement is executed.

An if statement used for grading

```
score = int(input("Enter test score: "))
if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "F"
```

Another way the if statement could be coded

```
score = int(input("Enter test score: "))
if score >= 90 and score <= 100:
    grade = "A"
elif score >= 80 and score < 90:
    grade = "B"
elif score >= 70 and score < 80:
    grade = "C"
elif score >= 60 and score < 70:
    grade = "D"
elif score < 60:
    grade = "F"
```

An if statement that validates the range of a score

```
score = int(input("Enter test score: "))
if score >= 0 and score <= 100:
    total_score += score
else:
    print("Test score must be from 0 - 100.")
```

An if statement that validates the customer type

```
is_valid = True
customer_type = input("Enter customer type (r/w): ")
if customer_type == "r" or customer_type == "w":
    pass            # this statement does nothing
else:
    print("Customer type must be 'r' or 'w'.")
    is_valid = False
```

A table that summarizes the discount rules

Type code	Invoice total	Discount percent
r (for Retail)	< 100	0
	>= 100 and < 250	.1
	>= 250	.2
w (for Wholesale)	< 500	.4
	>= 500	.5

Nested if statements

```
if customer_type.lower() == "r":
    if invoice_total < 100:
        discount_percent = 0
    elif invoice_total >= 100 and invoice_total < 250:
        discount_percent = .1
    elif invoice_total >= 250:
        discount_percent = .2
elif customer_type.lower() == "w":
    if invoice_total < 500:
        discount_percent = .4
    elif invoice_total >= 500:
        discount_percent = .5
else:
    discount_percent = 0
```

An if statement that gets the same results

```
# the discounts for Retail customers
if customer_type.lower() == "r" and invoice_total < 100:
    discount_percent = 0
elif customer_type.lower() == "r" and (
    invoice_total >= 100 and invoice_total < 250):
    discount_percent = .1
elif customer_type.lower() == "r" and invoice_total >= 250:
    discount_percent = .2
# the discounts for Wholesale customers
elif customer_type.lower() == "w" and invoice_total < 500:
    discount_percent = .4
elif customer_type.lower() == "w" and invoice_total >= 500:
    discount_percent = .5
# all other customers
else:
    discount_percent = 0
```

Pseudocode for customer discounts

```
Get customer type
IF type = R
    IF invoice total < 250
        discount = 0
    ELSE IF invoice total >= 250
        discount = 20%
ELSE IF type = W
    discount = 40%
ELSE
    print invalid type message
```

The Python code that's based on the pseudocode

```
customer_type = input("Enter customer type (R or W): ")
if customer_type.lower() == "r":
    if invoice_total < 250:
        discount_percent = 0
    elif invoice_total >= 250:
        discount_percent = .2
elif customer_type.lower() == "w":
    discount_percent = .4
else:
    print("Customer type must be R or W.")
```

Pseudocode for test score entries

Get test score
IF score is from 0 to 100
    add score to total score
    add 1 to the number of scores
ELSE IF score = 999
    print end of program message
ELSE
    print error message

The Python code that's based on the pseudocode

```
total_score = 0
score_counter = 0
score = int(input("Enter test score: "))
if score >= 0 and score <= 100:
    total_score += score
    score_counter += 1
elif score == 999:
    print("Ending program...")
else:
    print("Score must be from 0 through 100. Score discarded.")
```

The user interface with invalid data

```
The Miles Per Gallon program

Enter miles driven:          150
Enter gallons of gas used:   0
Gallons used must be greater than zero. Try again.

Bye!
```

The user interface with valid data

```
The Miles Per Gallon program

Enter miles driven:          150
Enter gallons of gas used:   30
Miles Per Gallon:            5.0

Bye!
```

The code for the Miles Per Gallon program

```
# display a welcome message
print("The Miles Per Gallon program")
print()

# get input from the user
miles_driven = float(input("Enter miles driven:         "))
gallons_used = float(input("Enter gallons of gas used:  "))

if miles_driven <= 0:
    print("Miles driven must be greater than zero. Try again.")
elif gallons_used <= 0:
    print("Gallons used must be greater than zero. Try again.")
else:
    # calculate and display miles per gallon
    mpg = round((miles_driven / gallons_used), 2)
    print("Miles Per Gallon:          ", mpg)

print()
print("Bye!")
```

Another way the if statement could be coded

```
if miles_driven > 0 and gallons_used > 0:
    mpg = round((miles_driven / gallons_used), 2)
    print("Miles Per Gallon:          ", mpg)
else:
   print("Both entries must be greater than zero. Try again.")
```

The user interface for the Invoice program

```
The Invoice program

Enter customer type (r/w):      R
Enter invoice total:            250

Invoice total:              250.0
Discount percent:           0.2
Discount amount:            50.0
New invoice total:          200.0
```

The code for the Invoice program (part 1)

```
#!/usr/bin/env python3

# display a welcome message
print("The Invoice program")
print()

# get user entries
customer_type = input("Enter customer type (r/w):\t")
invoice_total = float(input("Enter invoice total:\t\t"))
print()
```

The code for the Invoice program (part 2)

```
# determine discounts for Retail customers
if customer_type.lower() == "r":
    if invoice_total > 0 and invoice_total < 100:
        discount_percent = 0
    elif invoice_total >= 100 and invoice_total < 250:
        discount_percent = .1
    elif invoice_total >= 250 and invoice_total < 500:
        discount_percent = .2
    elif invoice_total >= 500:
        discount_percent = .25
# determine discounts for Wholesale customers
elif customer_type.lower() == "w":
    if invoice_total > 0 and invoice_total < 500:
        discount_percent = .4
    elif invoice_total >= 500:
        discount_percent = .5
# set discount to zero if neither Retail or Wholesale
else:
    discount_percent = 0
```

The code for the Invoice program (part 3)

```
# calculate discount amount and new invoice total
discount_amount =
    round(invoice_total * discount_percent, 2)
new_invoice_total = invoice_total - discount_amount

# display the results
print(f"Invoice total:\t\t{invoice_total}")
print(f"Discount percent:\t{discount_percent}")
print(f"Discount amount:\t{discount_amount}")
print(f"New invoice total:\t{new_invoice_total}")
print()
print("Bye!")
```

The syntax of the while statement

```
while boolean_expression:
    statements...
```

A while loop that continues as long as the user enters ‘y’ or ‘Y’

```
choice = "y"
while choice.lower() == "y":
    print("Hello!")
    choice = input("Say hello again? (y/n): ")
print("Bye!")  # runs when loop ends
```

The console after the loop runs

```
Hello!
Say hello again? (y/n): y
Hello!
Say hello again? (y/n): n
Bye!
```

A while loop that prints the numbers 0 through 4 to the console

```
counter = 0
while counter < 5:
    print(counter, end=" ")
    counter += 1
print("\nThe loop has ended.")
```

The console after the loop runs

```
0 1 2 3 4
The loop has ended.
```

Code that causes an infinite loop

```
while True:
    # any statements in this loop run forever
    # unless a break statement is executed as shown later
```

How to end an infinite loop

- Press Ctrl+C (Windows) or Command+C (macOS).

The syntax of a for loop with the range() function

```
for int_var in range_function:
    statements...
```

The range() function

```
range(stop)
range(start, stop[, step])
```

Examples of the range() function

```
range(5)                    # 0, 1, 2, 3, 4
range(1, 6)                 # 1, 2, 3, 4, 5
range(2, 10, 2)             # 2, 4, 6, 8
range(5, 0, -1)             # 5, 4, 3, 2, 1
```

A for loop that prints the numbers 0 through 4

```
for i in range(5):
    print(i, end=" ")
print("\nThe loop has ended.")
```

The console after the loop runs

```
0 1 2 3 4
The loop has ended.
```

A for loop that sums the numbers 1 through 4

```
sum_of_numbers = 0
for i in range(1,5):
    sum_of_numbers += i
print(sum_of_numbers)
```

The console after the loop runs

```
10
```

A break statement that exits an infinite while loop

```
print("Enter 'exit' when you're done.\n")
while True:
    data = input("Enter an integer to square: ")
    if data == "exit":
        break
    i = int(data)
    print(i, "squared is", i * i, "\n")
print("Okay, bye!")
```

The console

```
Enter 'exit' when you're done.

Enter an integer to square: 10
10 squared is 100

Enter an integer to square: 23
23 squared is 529

Enter an integer to square: exit
Okay, bye!
```

A continue statement that jumps to the beginning of a while loop

```
more = "y"
while more.lower() == "y":
    miles_driven = float(input("Enter miles driven:\t\t"))
    gallons_used = float(input("Enter gallons of gas used:\t"))

    # validate input
    if miles_driven <= 0 or gallons_used <= 0:
        print("Both entries must be greater than zero. ",
              "Try again.\n")
        continue

    mpg = round(miles_driven / gallons_used, 2)
    print("Miles Per Gallon:", mpg, "\n")

    more = input("Continue? (y/n): ")
    print()

print("Okay, bye!")
```

Loops that calculates the future value of a one-time investment

A for loop

```
investment = 10000
for i in range(20):
    yearly_interest = investment * .05
    investment = investment + yearly_interest
investment = round(investment, 2)
```

A while loop

```
year = 0
investment = 10000
while year < 20:
    yearly_interest = investment * .05
    investment = investment + yearly_interest
    year += 1
investment = round(investment, 2)
```

A for loop that calculates the future value of a monthly investment

```
monthly_investment = 100
monthly_interest_rate = .08 / 12
months = 120
future_value = 0
for month in range(months):
    future_value += monthly_investment
    monthly_interest_amount = future_value *
                              monthly_interest_rate
    future_value += monthly_interest_amount
future_value = round(future_value, 2)
```

Nested loops that get the total of 3 valid test scores

```
total_score = 0
for i in range(3):
    while True:
        score = int(input("Enter test score: "))
        if score >= 0 and score <= 100:
            total_score += score
            break
        else:
            print("Test score must be from 0 - 100.")
print("Total score:", total_score)
```

The console

```
Enter test score: 110
Test score must be from 0 - 100.
Enter test score: -10
Test score must be from 0 - 100.
Enter test score: 100
Enter test score: 90
Enter test score: 0
Total score: 190
```

The operator used with assignment expressions

Operator	Name
`:=`	Walrus

A while statement that uses an infinite loop to process user data

```
print("Enter -1 to quit.")
print("=================")
while True:
    score = input("Enter a score: ") # assign
    if score == "-1":                # check
        break
    print(f"You entered {score}.")
print("Bye!")
```

How to rewrite the code using an assignment expression

```
print("Enter -1 to quit.")
print("==================")
# assign and check
while (score := input("Enter a score: ")) != "-1":
    print(f"You entered {score}.")
print("Bye!")
```

The console for both loops

```
Enter -1 to quit.
==================
Enter a score: 90
You entered 90.
Enter a score: 99
You entered 99.
Enter a score: -1
Bye!
```

Pseudocode for a Test Scores program

```
Display user message
WHILE TRUE
    get score
    IF score is from 0 to 100
            add score to score total
            add 1 to number of scores
    ELSE IF score is 999
            end loop
    ELSE
            print error message
Calculate average score
Display results
```

The user interface for the Test Scores program

```
The Test Scores program

Enter 999 to end input
======================
Enter test score: 85
Enter test score: 95
Enter test score: 155
Test score must be from 0 through 100. Try again.
Enter test score: 75
Enter test score: 999
======================
Total Score: 255
Average Score: 85

Bye!
```

The code for the Test Scores program (part 1)

```
#!/usr/bin/env python3

# display a welcome message
print("The Test Scores program")
print()
print("Enter 999 to end input")
print("======================")

# initialize variables
counter = 0
score_total = 0
test_score = 0
```

The code for the Test Scores program (part 2)

```
while True:
    test_score = int(input("Enter test score: "))
    if test_score >= 0 and test_score <= 100:
        score_total += test_score
        counter += 1
    elif test_score == 999:
        break
    else:
        print("Test score must be from 0 through 100. ",
              "Score discarded. Try again.")

# calculate average score
average_score = round(score_total / counter)

# format and display the result
print("======================")
print(f"Total Score: {score_total}"
      f"\nAverage Score: {average_score}")
print()
print("Bye")
```

Pseudocode for a Future Value program

```
Display user message
WHILE user wants to continue
    get monthly investment, yearly interest rate, and years
    convert yearly interest rate to monthly interest rate
    convert years to months
    set the future value to zero
    FOR each month
        add monthly investment amount to future value
        calculate interest for month
        add interest to future value
    display future value
    ask if user wants to continue
Display end message
```

The user interface for the Future Value Calculator

```
Welcome to the Future Value Calculator

Enter monthly investment:        100
Enter yearly interest rate:      12
Enter number of years:           10
Future value:                    23233.91

Continue (y/n)?:
```

The code for the Future Value Calculator (part 1)

```
#!/usr/bin/env python3

# display a welcome message
print("Welcome to the Future Value Calculator")
print()

choice = "y"
while choice.lower() == "y":

    # get input from the user
    monthly_investment = float(input(
        "Enter monthly investment:\t"))
    yearly_interest_rate = float(input(
        "Enter yearly interest rate:\t"))
    years = int(input(
        "Enter number of years:\t\t"))

    # convert yearly values to monthly values
    monthly_interest_rate = yearly_interest_rate / 12 / 100
    months = years * 12
```

The code for the Future Value Calculator (part 2)

```
    # calculate the future value
        future_value = 0
        for i in range(months):
            future_value += monthly_investment
            monthly_interest_amount = future_value *
                                      monthly_interest_rate
            future_value += monthly_interest_amount

        # display the result
        print(f"Future value:\t\t\t{round(future_value, 2)}")
        print()

        # see if the user wants to continue
        choice = input("Continue (y/n)?: ")
        print()

    print("Bye!")
```