Chapter 3

Decision Structures and
Boolean Logic

The `if` Statement

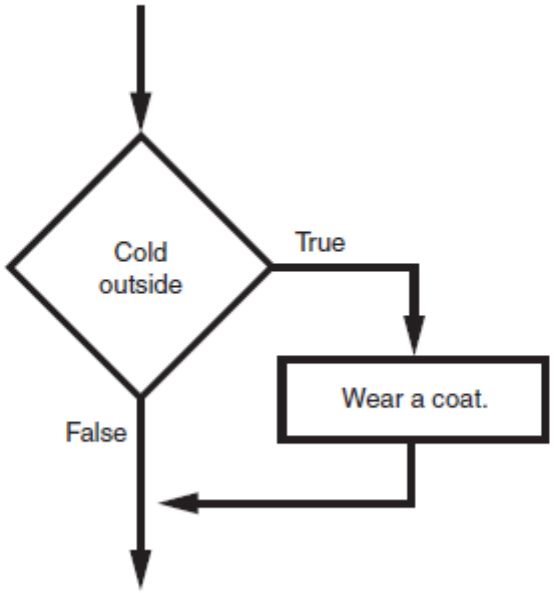
- Control structure: logical design that controls the order in which set of statements execute
- Sequence structure: set of statements that execute in the order they appear
- Decision structure: specific action(s) performed only if a condition exists
  - Also known as selection structure

The `if` Statement (cont'd.)

- In a flowchart, a diamond represents true/false condition that must be tested
- Actions can be *conditionally executed*
  - Performed only when a condition is true
- Single alternative decision structure: provides only one alternative path of execution
  - If condition is not true, exit the structure

The `if` Statement (cont'd.)

Figure 3-1 A simple decision structure



The `if` Statement (cont'd.)

- Python syntax:

```
if condition:
    Statement
    Statement
```

- First line known as the `if` clause
  - Includes the keyword `if` followed by condition
    - The condition can be true or false
    - When the `if` statement executes, the condition is tested, and if it is true the block statements are executed. otherwise, block statements are skipped

```
# This program gets three test scores and displays
# their average.  It congratulates the user if the
# average is a high score.

# The high score variable holds the value that is
# considered a high score.
high_score = 95

# Get the three test scores.
test1 = int(input('Enter the score for test 1: '))
test2 = int(input('Enter the score for test 2: '))
test3 = int(input('Enter the score for test 3: '))

# Calculate the average test score.
average = (test1 + test2 + test3) / 3

# Print the average.
print('The average score is', average)

# If the average is a high score,
# congratulate the user.
if average >= high_score:
    print('Congratulations!')
    print('That is a great average!')
```

Boolean Expressions and Relational Operators

- Boolean expression: expression tested by if statement to determine if it is true or false
  - Example: a > b
    - `true` if a is greater than b; `false` otherwise
- Relational operator: determines whether a specific relationship exists between two values
  - Example: greater than (>)

Boolean Expressions and Relational Operators (cont'd.)

- >= and <= operators test more than one relationship
  - It is enough for one of the relationships to exist for the expression to be true
- == operator determines whether the two operands are equal to one another
  - Do not confuse with assignment operator (=)
- != operator determines whether the two operands are not equal

Boolean Expressions and Relational Operators (cont'd.)

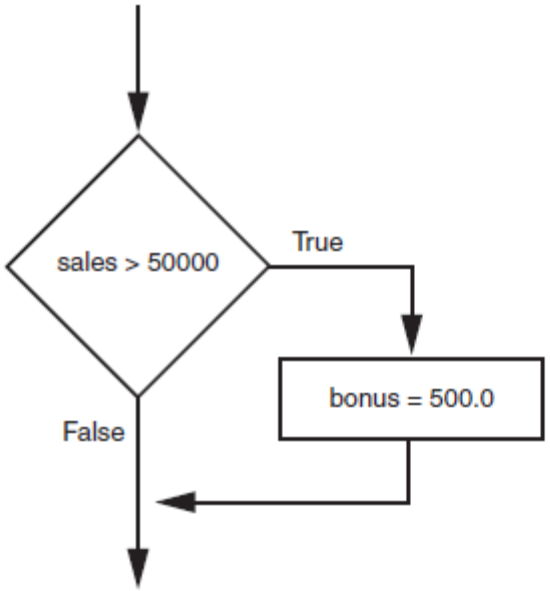
Table 3-2 Boolean expressions using relational operators

Expression	Meaning
`x > y`	Is x greater than y?
`x < y`	Is x less than y?
`x >= y`	Is x greater than or equal to y?
`x <= y`	Is x less than or equal to y?
`x == y`	Is x equal to y?
`x != y`	Is x not equal to y?

Boolean Expressions and Relational Operators (cont'd.)

- Using a Boolean expression with the > relational operator

Figure 3-3 Example decision structure



Boolean Expressions and Relational Operators (cont'd.)

- Any relational operator can be used in a decision block
  - Example: `if balance == 0`
  - Example: `if payment != balance`
- It is possible to have a block inside another block
  - Example: `if` statement inside a function
  - Statements in inner block must be indented with respect to the outer block

The `if-else` Statement

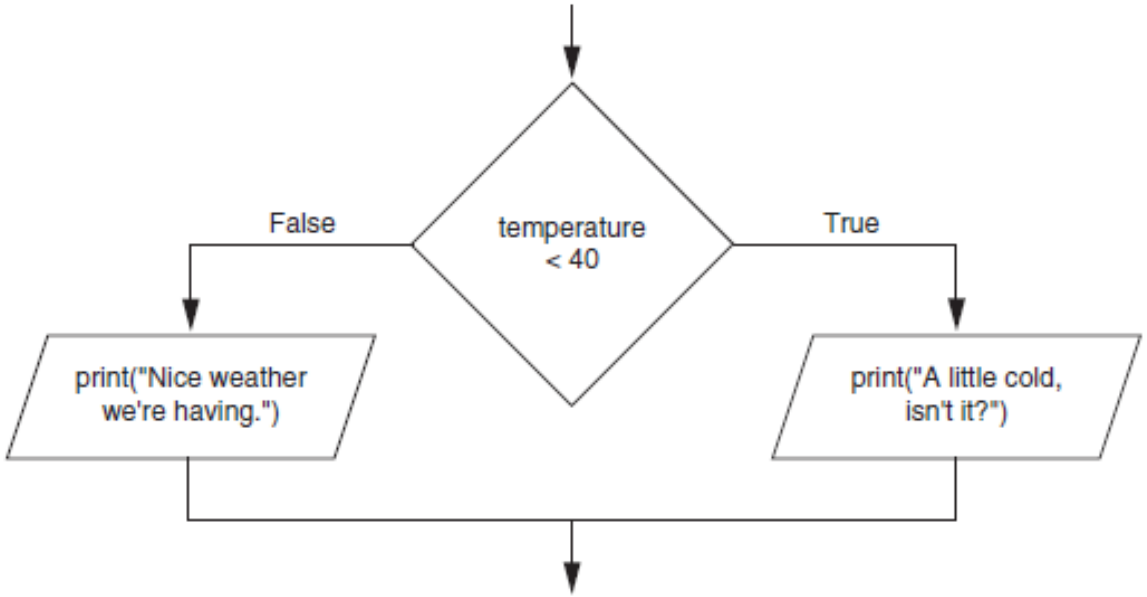
- Dual alternative decision structure: two possible paths of execution
  - One is taken if the condition is true, and the other if the condition is false
  - Syntax:

```
if condition:
        statements
else:
        other statements
```

  - `if` clause and `else` clause must be aligned
  - Statements must be consistently indented

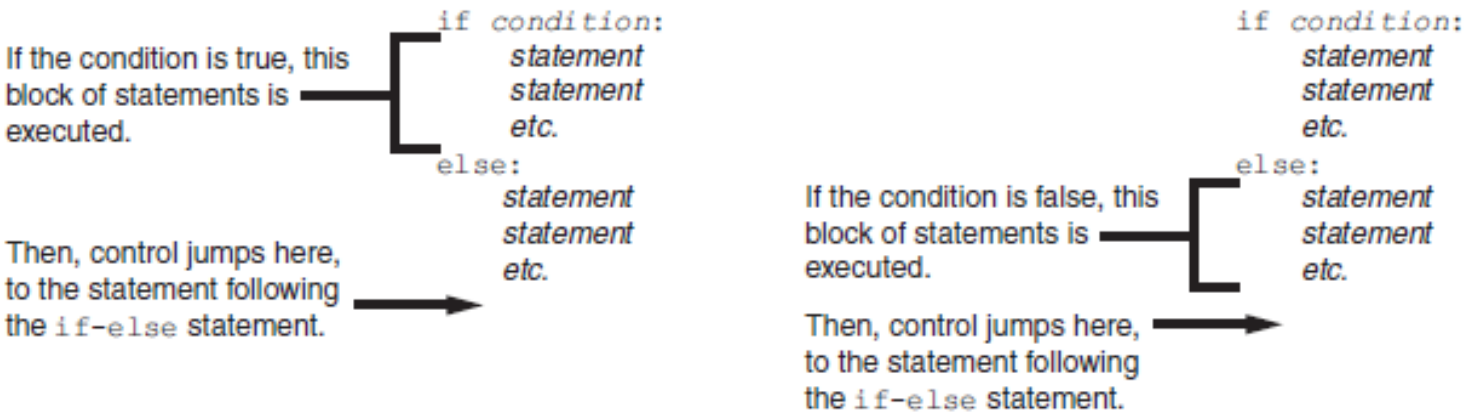
The `if-else` Statement (cont'd.)

Figure 3-5 A dual alternative decision structure



The `if-else` Statement (cont'd.)

Figure 3-6 Conditional execution in an `if-else` statement



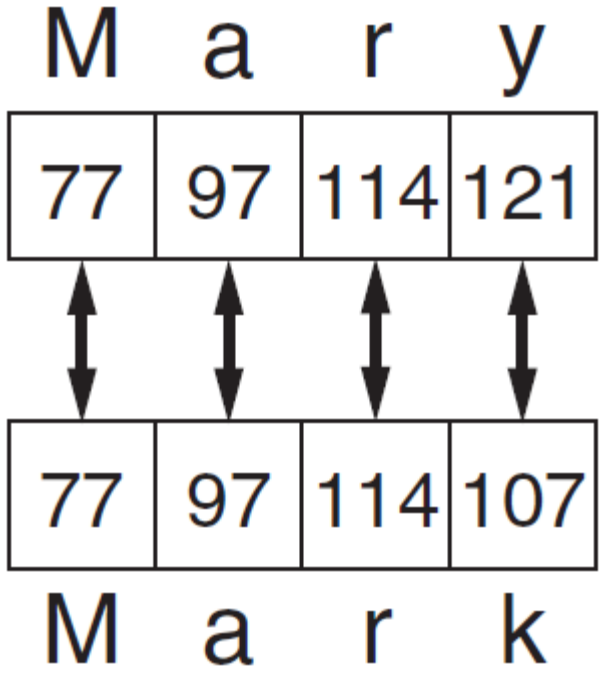
```
# This program determines whether a bank customer
# qualifies for a loan.

min_salary = 30000.0  # The minimum annual salary
min_years = 2         # The minimum years on the job

# Get the customer's annual salary.
salary = float(input('Enter your annual salary: '))

# Get the number of years on the current job.
years_on_job = int(input('Enter the number of ' +
                         'years employed: '))

# Determine whether the customer qualifies.
if salary >= min_salary and years_on_job >= min_years:
    print('You qualify for the loan.')
else:
    print('You do not qualify for this loan.')
```

Comparing Strings

- Strings can be compared using the == and != operators
- String comparisons are case sensitive
- Strings can be compared using >, <, >=, and <=
  - Compared character by character based on the ASCII values for each character
  - If shorter word is substring of longer word, longer word is greater than shorter word

Comparing Strings (cont'd.)

Figure 3-9 Comparing each character in a string

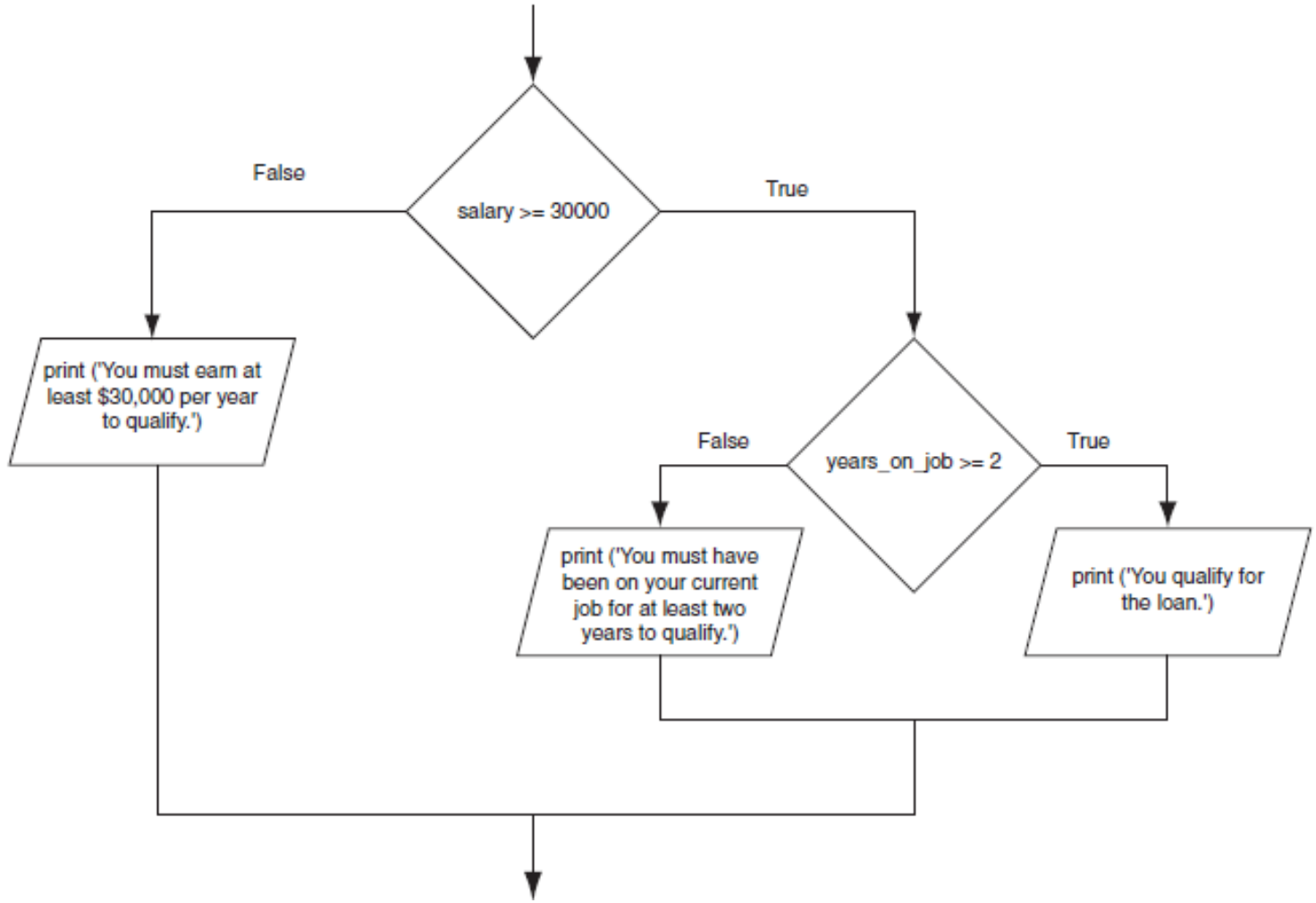


```
# This program compares two strings.
# Get a password from the user.
password = input('Enter the password: ')

# Determine whether the correct password
# was entered.
if password == 'prospero':
    print('Password accepted.')
else:
    print('Sorry, that is the wrong password.')
```

Nested Decision Structures and the `if-elif-else` Statement

- A decision structure can be nested inside another decision structure
  - Commonly needed in programs
  - Example:
    - Determine if someone qualifies for a loan, they must meet two conditions:
      - Must earn at least $30,000/year
      - Must have been employed for at least two years
    - Check first condition, and if it is true, check second condition

Figure 3-12 A nested decision structure



Nested Decision Structures and the `if-elif-else` Statement (cont'd.)

- Important to use proper indentation in a nested decision structure
  - Important for Python interpreter
  - Makes code more readable for programmer
  - Rules for writing nested if statements:
    - `else` clause should align with matching `if` clause
    - Statements in each block must be consistently indented

The `if-elif-else` Statement

- `if-elif-else` statement: special version of a decision structure
  - Makes logic of nested decision structures simpler to write
    - Can include multiple `elif` statements
  - Syntax:

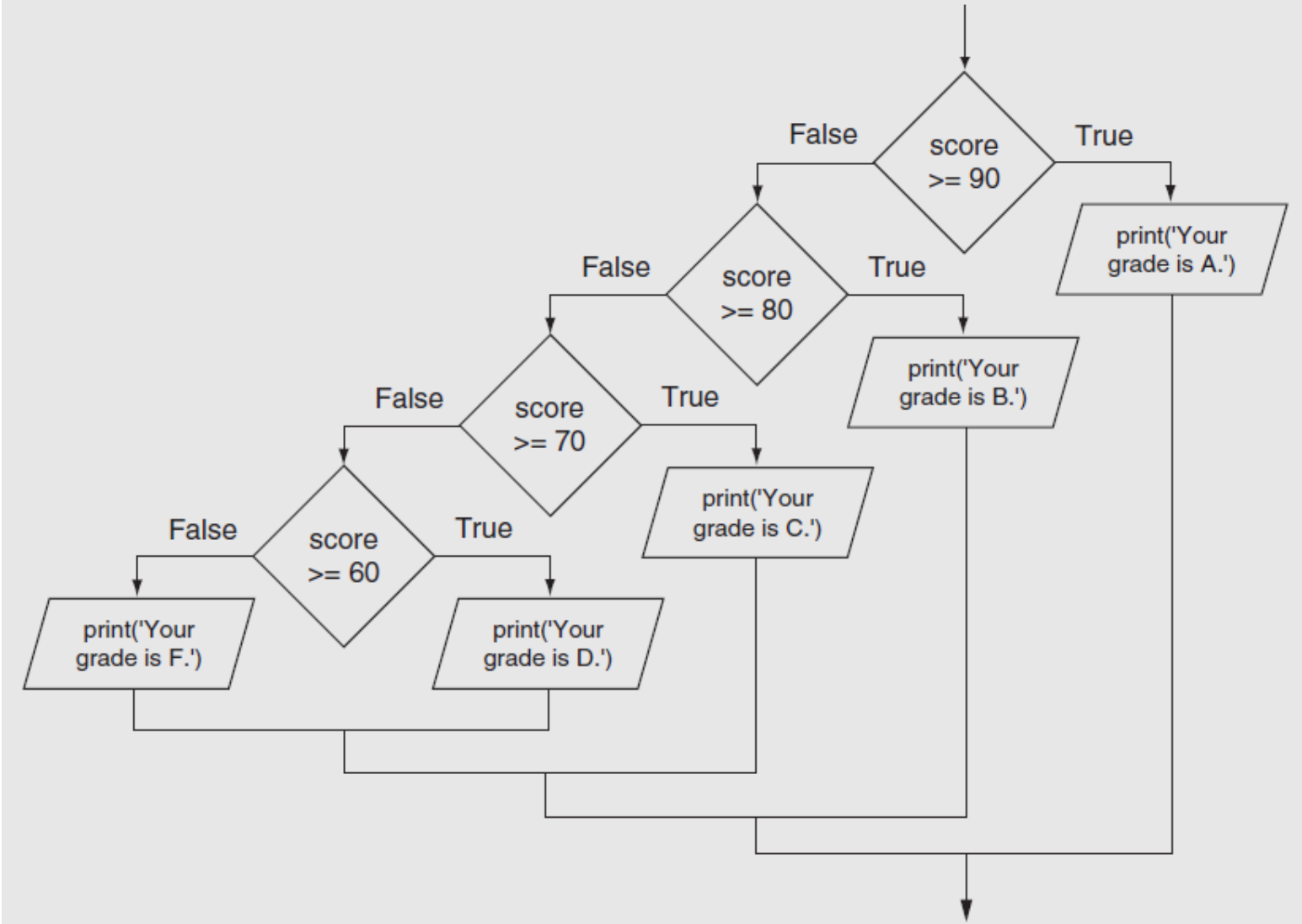
```
if condition_1:
    statement(s)
elif condition_2:
    statement(s)
elif condition_3:
    statement(s)
else:
    statement(s)
```

Insert as many `elif` clauses as necessary.

The `if-elif-else` Statement (cont'd.)

- Alignment used with `if-elif-else` statement:
  - `if`, `elif`, and `else` clauses are all aligned
  - Conditionally executed blocks are consistently indented
- `if-elif-else` statement is never required, but logic easier to follow
  - Can be accomplished by nested `if-else`
    - Code can become complex, and indentation can cause problematic long lines

Figure 3-15 Nested decision structure to determine a grade



Logical Operators

- Logical operators: operators that can be used to create complex Boolean expressions
  - `and` operator and `or` operator: binary operators, connect two Boolean expressions into a compound Boolean expression
  - `not` operator: unary operator, reverses the truth of its Boolean operand

The `and` Operator

- Takes two Boolean expressions as operands
  - Creates compound Boolean expression that is true only when both sub expressions are true
  - Can be used to simplify nested decision structures
- Truth table for the `and` operator

Expression	Value of the Expression
false and false	false
false and true	false
true and false	false
true and true	true

The `or` Operator

- Takes two Boolean expressions as operands
  - Creates compound Boolean expression that is true when either of the sub expressions is true
  - Can be used to simplify nested decision structures
- Truth table for the `or` operator

Expression	Value of the Expression
false and false	false
false and true	true
true and false	true
true and true	true

Short-Circuit Evaluation

- Short circuit evaluation: deciding the value of a compound Boolean expression after evaluating only one sub expression
  - Performed by the `or` and `and` operators
    - For `or` operator: If left operand is true, compound expression is true. Otherwise, evaluate right operand
    - For `and` operator: If left operand is false, compound expression is false. Otherwise, evaluate right operand

The `not` Operator

- Takes one Boolean expressions as operand and reverses its logical value
  - Sometimes it may be necessary to place parentheses around an expression to clarify to what you are applying the not operator
- Truth table for the `not` operator

Expression	Value of the Expression
true	false
false	true

Checking Numeric Ranges with Logical Operators

- To determine whether a numeric value is within a specific range of values, use `and`

  Example: `x >= 10 and x <= 20`

- To determine whether a numeric value is outside of a specific range of values, use `or`

  Example: `x < 10 or x > 20`

- Testing whether a numeric value is within a certain range can also be accomplished without the use of logical operators

  Example: `10 <= x <= 20`

Boolean Variables

- Boolean variable: references one of two values, `True` or `False`
  - Represented by `bool` data type
- Commonly used as flags
  - Flag: variable that signals when some condition exists in a program
    - Flag set to `False` → condition does not exist
    - Flag set to `True` → condition exists