Topics

- The `if` Statement
- The `if-else` Statement
- Comparing Strings
- Nested Decision Structures and the `if-elif-else` Statement
- Logical Operators
- Boolean Variables

The `if` Statement

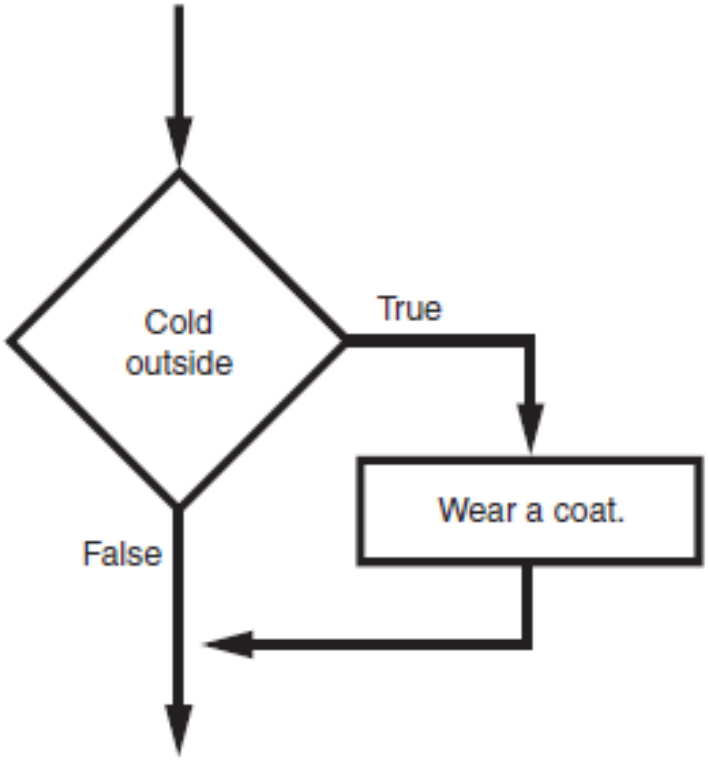
- **Control structure: logical design that controls order in which set of statements execute**
- **Sequence structure: set of statements that execute in the order they appear**
- **Decision structure: specific action(s) performed only if a condition exists**
  - Also known as selection structure

The `if` Statement (cont'd.)

- **In flowchart, 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 4-1 A simple decision structure



The `if` Statement (cont'd.)

- **Python syntax:**

```
if condition:
    Statement
    Statement
```

- **First line know 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

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.)

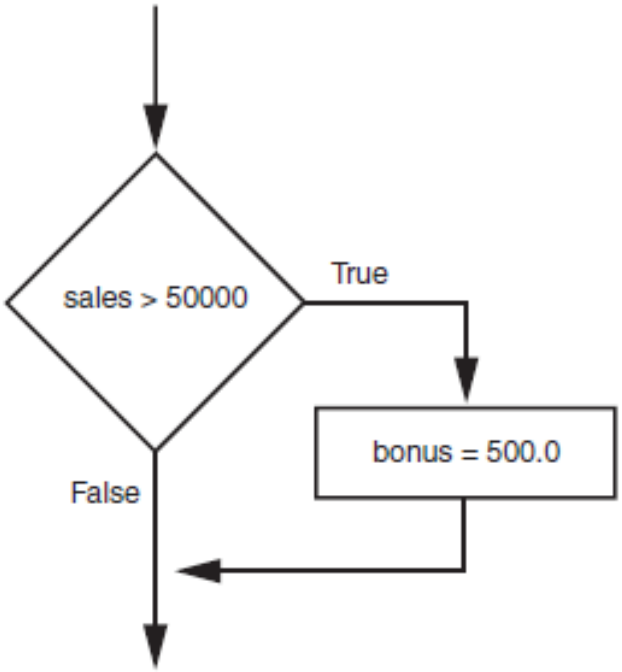
Table 4-2 Boolean expressions using relational operators

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 4-3 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

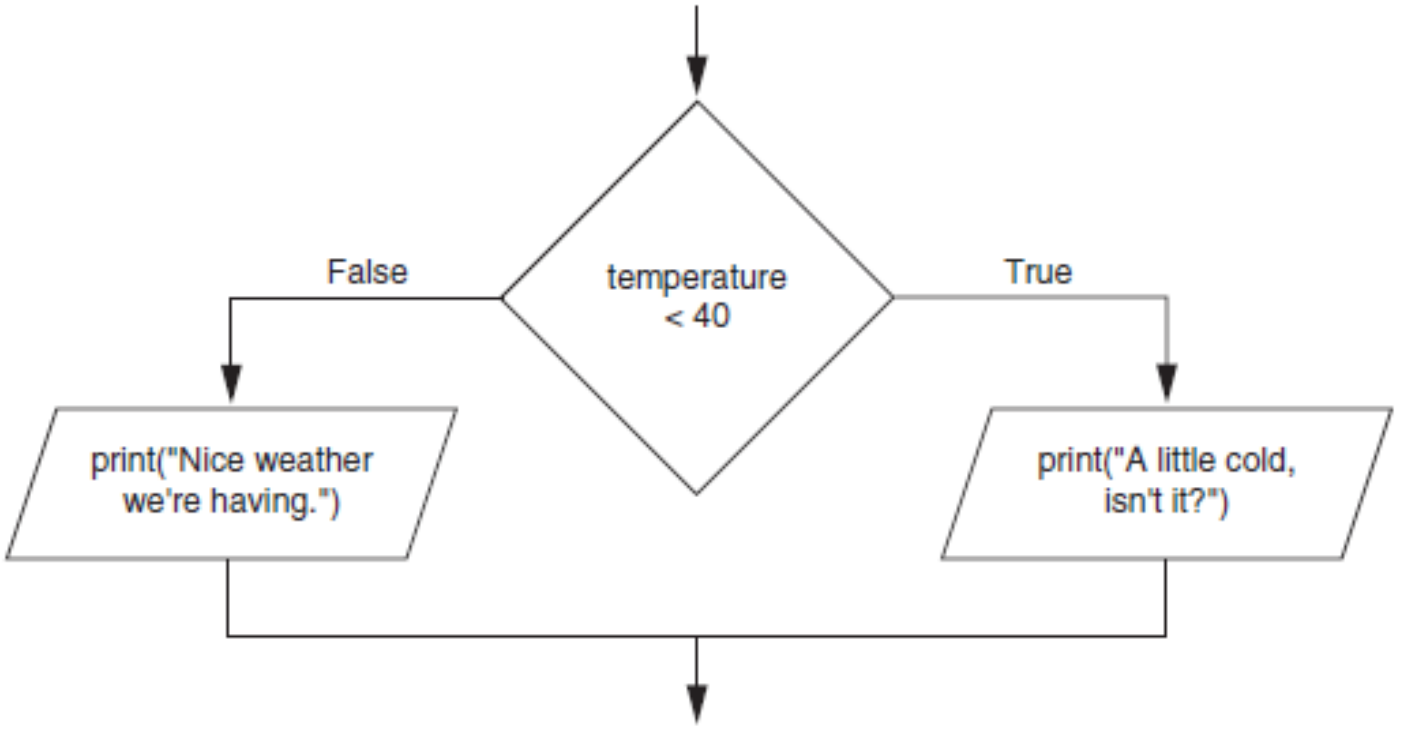
- **<u>Dual alternative decision structure</u>: two possible paths of execution**
  - One is taken if the condition is true, and the other if the condition is false
  - Syntax:

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

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

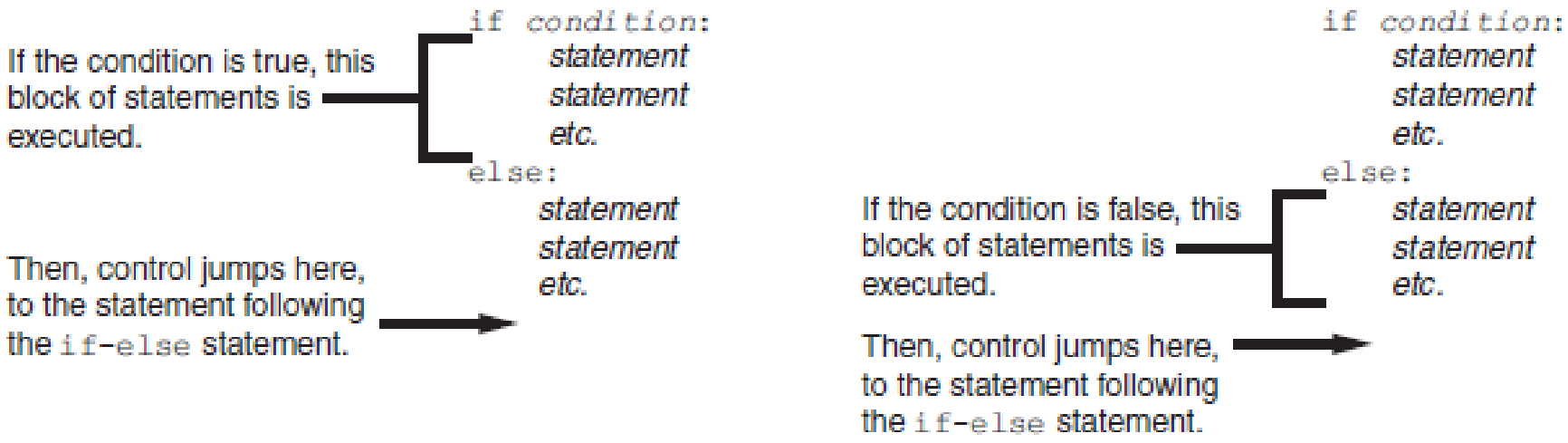
The `if-else` Statement (cont'd.)

Figure 4-6 A dual alternative decision structure



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

Figure 4-7 Conditional execution in an `if-else` statement

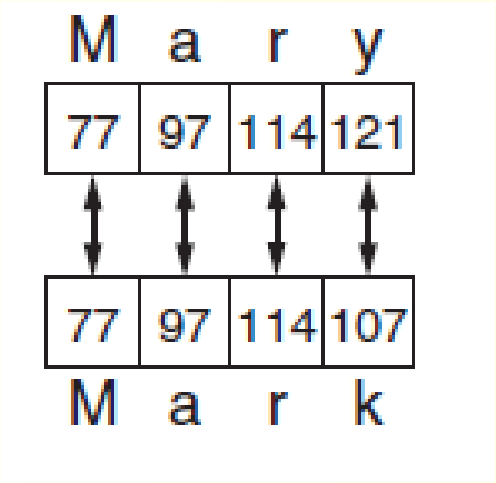


Comparing Strings

- **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.)

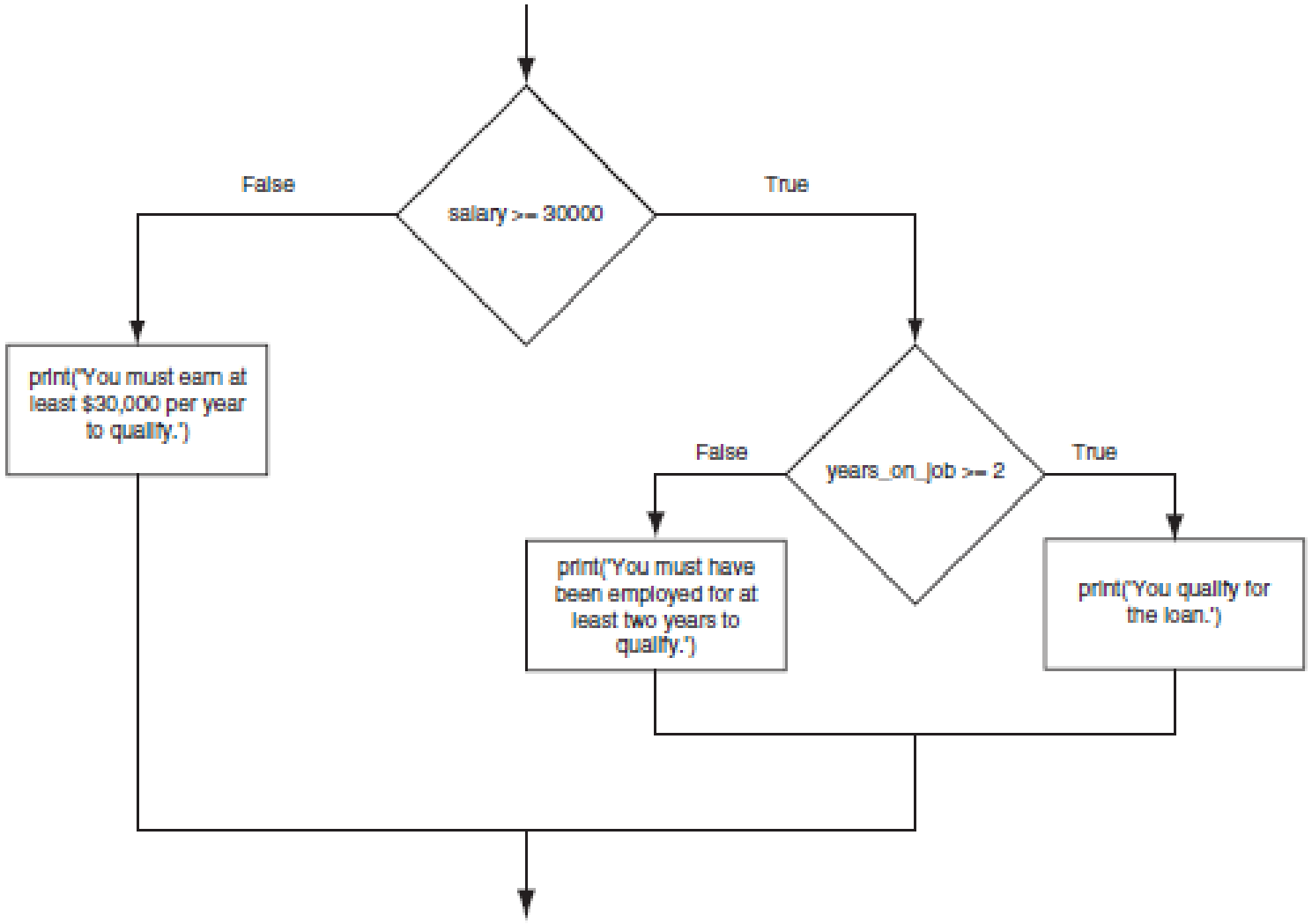
Figure 4-11 Comparing each character in a string



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 4-14 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

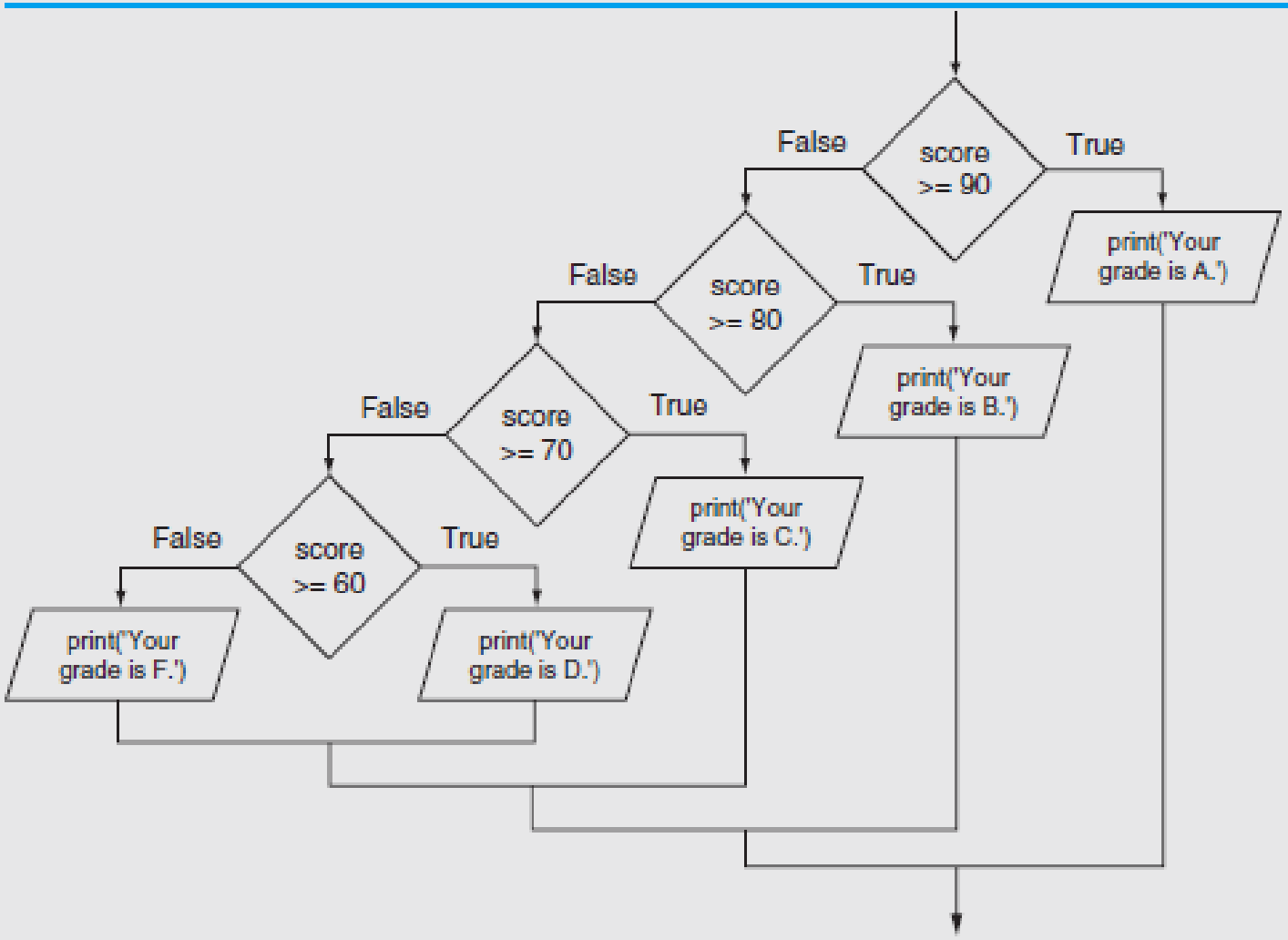
- **`if-elif-else` statement: special version of a decision structure**
  - Makes logic of nested decision structures simpler to write
    - Can include multiple `elif` statements
  - Syntax:

```
if condition1
      statements
elif condition2
      statements
else
      statements
```

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 4-17 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`

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

Summary

- **This chapter covered:**
  - Decision structures, including:
    - Single alternative decision structures
    - Dual alternative decision structures
    - Nested decision structures
  - Relational operators and logical operators as used in creating Boolean expressions
  - String comparison as used in creating Boolean expressions
  - Boolean variables

Summary

- **This chapter covered:**
  - Decision structures, including:
    - Single alternative decision structures
    - Dual alternative decision structures
    - Nested decision structures
  - Relational operators and logical operators as used in creating Boolean expressions
  - String comparison as used in creating Boolean expressions
  - Boolean variables

```
#pg.121
sales = int(input("Enter sales: "))
bonus = 0
if sales >= 5000:
    bonus = 500.0
print("Your bonus is: ", bonus)
```

```
#This program compares 2 strings
#Get password from the user
pswd = raw_input("Enter password ")
#Determine if the password is correct
if pswd == 'prosperous':
    print('Password accepted')
else:
    print('sorry, that"s is the wrong password')
```

```
#This program comapres strings with the < operator
#Get 2 names from the user
name1 = raw_input("Enter a name: ")
name2 = raw_input("Enter another name: ")

#display the names in alphabetical order
print ("Here are the names, listed alphabetically")

if name1 < name2:
    print(name1)
    print(name2)
else:
    print(name2)
    print(name1)
```

```
#This program ask for a temperature
#and displays messages accordingly

temp = float(input("Enter temperature "))
if temp < 40:
    print ("it is cold ")
    print("turn up the heat!")
else:
    print("nice weather...")
    print("Pass the sunscreen.")
```

```
#This program finds out a letter grade based on a score

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

#This program calculates regular wages or overtime wages

```
#Global constants
BASE_HOURS = 40
OT_MULTIPLIER = 1.5

def calc_pay_with_OT(hours, rate):
    #calculate the number of overtime hours worked
    overtime_hours = hours - BASE_HOURS

    #Calculate the amont of overtime pay
    overtime_pay = overtime_hours * rate * OT_MULTIPLIER

    #Calculate the gross pay
    gross_pay = BASE_HOURS * rate + overtime_pay

    #Display the gross pay
    print("The gross pay is $",format(gross_pay, ',.2f'))
```

```
def calc_regular_pay(hours,rate):
    #Calculate the gross pay
    gross_pay = hours * rate

    #display the gross pay
    print('The gross pay is $', format(gross_pay, ',.2f'))

def main():
    #get hours worked and hourly pay rate
    hours_worked = float(input("Enter hours worked "))
    pay_rate = float(input("Enter hourly rate "))

    #calculate and display gross pay
    if hours_worked > BASE_HOURS:
        calc_pay_with_OT(hours_worked, pay_rate)
    else:
        calc_regular_pay(hours_worked, pay_rate)


#call main function
main()
```