PROGRAMMING IN PYTHON

UNIT-I

CH-3: Decision Structures & Boolean Logic

Ms. SALMA BEGUM

Assistant Professor, Dept. Of Computer Science

RBVRR Women's College, Narayanaguda, Hyderabad.

CONTENT

Introductions

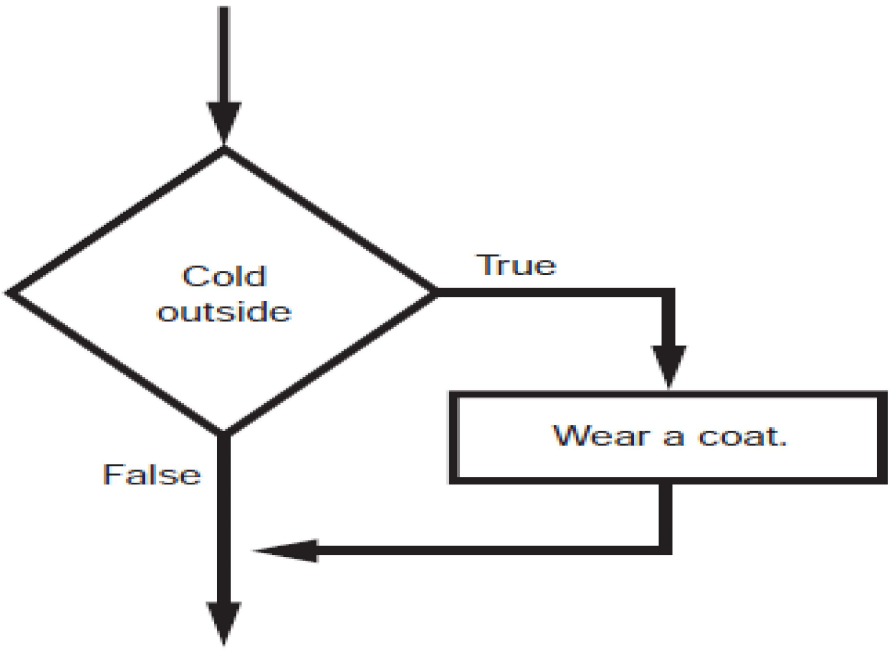
- A *control structure* is a logical design that controls the order in which a set of statements execute. The simplest type of control structure is the ***sequence structure***.
- A ***sequence structure*** is a set of statements that execute in the order that they appear.

Eg:

```
1. name = input('What is your name? ')
2. print('Here is the data you entered:')
3. print('Age:', age)
```

- It cannot handle every type of problems which cannot be solved by performing a set of ordered steps, one after the other.
- For example, consider a pay calculating program that determines whether an employee has worked overtime. If the employee has worked more than 40 hours, he or she gets paid extra for all the hours over 40. Otherwise, the overtime calculation should be skipped. This type of program require a different type of control structure: one that can execute a set of statements only under certain circumstances. This can be accomplished with a ***decision structure***. (Also known as ***selection structures***.)
- In a ***decision structure's*** simplest form, a specific action is performed only if a certain condition exists. If the condition does not exist, the action is not performed.

Cont.....



- The type of decision structure shown in the above figure is a ***single alternative decision structure***. This is because it provides only one alternative path of execution. If the condition in the diamond symbol is true, we take the alternative path. Otherwise, we exit the structure.
- In Python we use the if statement to write a single alternative decision structure.

3.1 The if Statement

The if statement is used to create a decision structure, which allows a program to have more than one path of execution. The if statement causes one or more statements to execute only when a Boolean expression is true.

General format of the if statement:

if ***condition:***

statement

statement

etc.

Cont.....

- *The if clause begins with* the word if, followed by a *condition,* which is an expression that will be evaluated as either true or false.
- A colon appears after the *condition.* The next line is a *block of statements*. A block is set of statements that belong together as a group.
- This indentation is required because the Python interpreter uses it to tell where the block begins and ends.
- When the if statement executes, the *condition is tested.* If the condition is
  - *true,* the statements that appear in the block following the if clause are executed.
  - false, the statements in the block are skipped.

Boolean Expressions and Relational Operators

- The expressions that are tested by the if statement are called *Boolean expressions*, named in honor of the English mathematician George Boole.
- In the 1800s Boole invented a system of mathematics in which the abstract concepts of true and false can be used in computations.
- Boolean expression that is tested by an if statement is formed with a relational operator. A ***relational operator*** determines whether a specific relationship exists between two values.

Cont.....

- For example, the greater than operator (>) determines whether one value is greater than another. The equal to operator (==) determines whether two values are equal or not.

Table 3-1 Relational operators

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

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?

Cont.....

- **The >= and <= Operators,** test for more than one relationship.
  - >= operator: determines whether the operand on its left is greater than *or* equal to the operand on its right.
  - <= operator: determines whether the operand on its left is less than *or* equal to the operand on its right.
- **The = = Operator** determines whether the operand on its left is equal to the operand on its right. If the values referenced by both operands are the same, the expression is true.
- **The != operator** is the not-equal-to operator. It determines whether the operand on its left is not equal to the operand on its right, which is the opposite of the == operator.

Cont.....

Kathryn teaches a science class and her students are required to take three tests. She wants to write a program that her students can use to calculate their average test score. She also wants the program to congratulate the student enthusiastically if the average is greater than 95.

Algorithm in pseudocode:

1. *Get the first test score*
2. *Get the second test score*
3. *Get the third test score*
4. *Calculate the average*
5. *Display the average*
6. *If the average is greater than 95:*
7. *Congratulate the user*

Output with Input

Enter the score for test 1: **82**
Enter the score for test 2: **76**
Enter the score for test 3: **91**
The average score is 83.0

Output with Input

Enter the score for test 1: **93**
Enter the score for test 2: **99**
Enter the score for test 3: **96**
The average score is 96.0
Congratulations!
That is a great average!

PROGRAM:

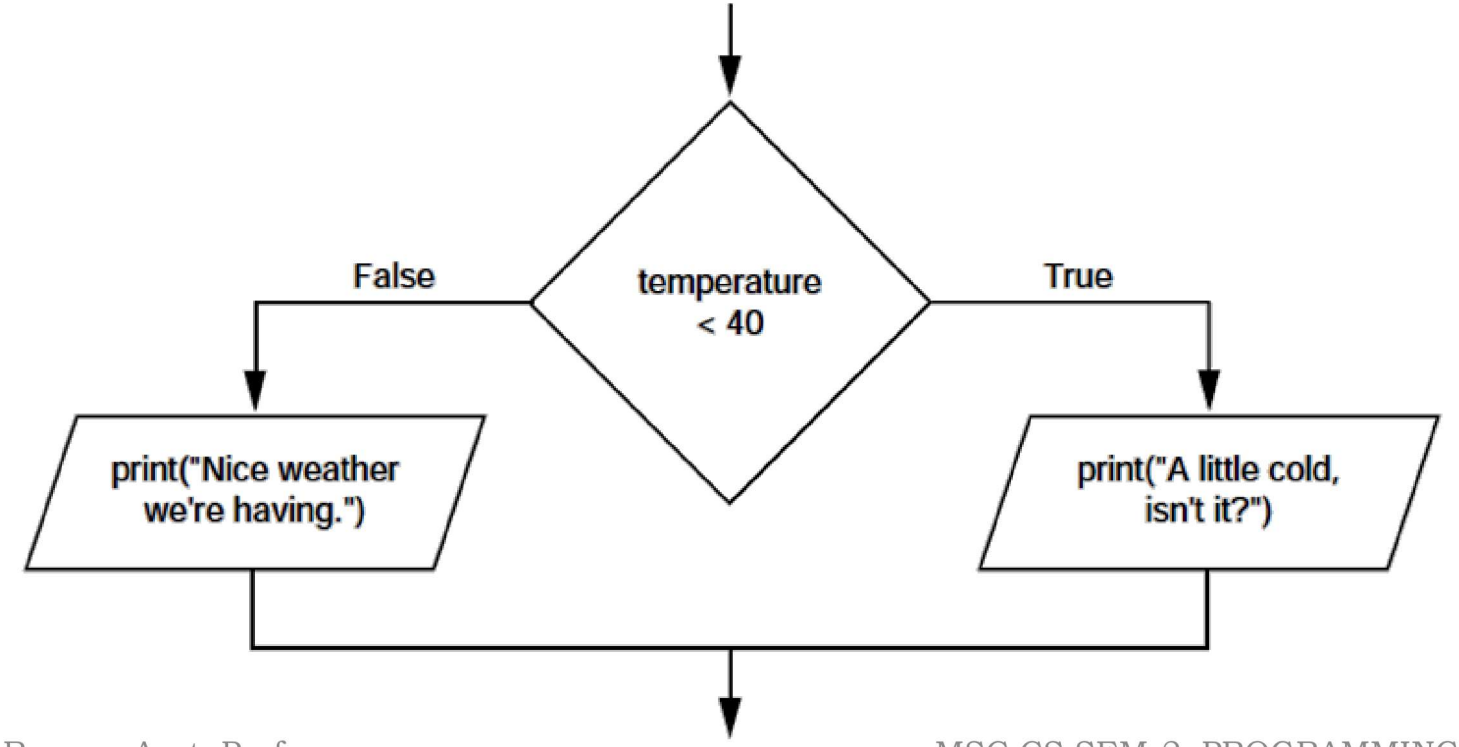
```
high_score = 95
test1 = int(input('Enter the score for test 1:' )) # Get the test1 score.
test2 = int(input('Enter the score for test 2:' )) # Get the test2 score.
test3 = int(input('Enter the score for test 3:' )) # Get the testscore.
average = (test1 + test2 + test3) / 3   # Calculate the average test score.
print('The average score is', average) # Print the average.
# If the average is a high score, congratulate the user.
if average >= high_score:
    print('Congratulations!')
    print('That is a great average!')
```

TEST

- Write an if statement that assigns 0 to x if y is equal to 20.
- Write an if statement that assigns 0.2 to commissionRate if sales is greater than or equal to 10000.

3.2 The if-else Statement

- An if-else statement will execute one block of statements if its condition is true, or another block if its condition is false.
- The ***dual alternative decision structure,*** which has two possible paths of execution—one path is taken if a condition is true, and the other path is taken if the condition is false.
- Below figure shows a flowchart for a dual alternative decision structure.



Cont.....

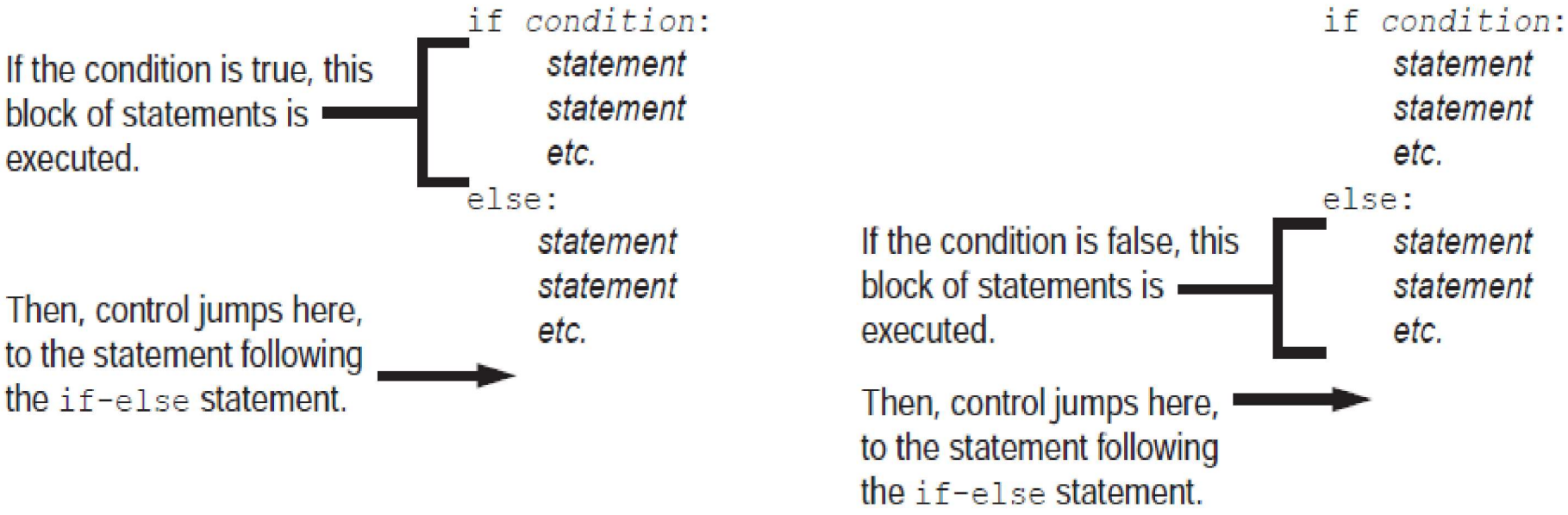
General format of the if-else statement:

if *condition:*
    statement
    statement
    etc.
else:
    statement
    statement
    etc.

Example:

```
if temperature < 40:
    print("A little cold, isn't it?")
else:
    print("Nice weather we're having.")
```

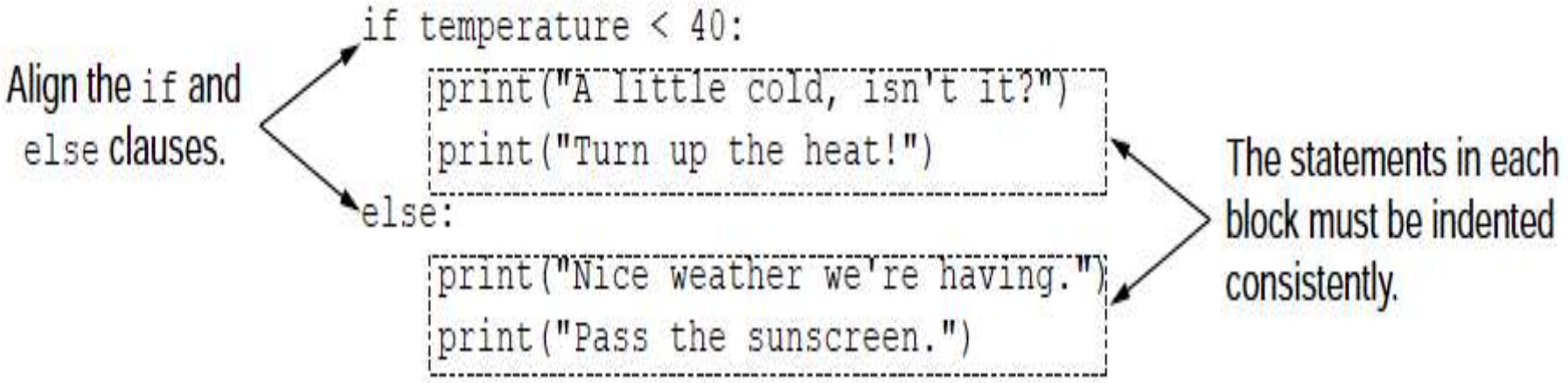
The below figure shows conditional execution in an if-else Statement



Cont.....

Indentation in the if-else Statement

- When you write an if-else statement, follow these guidelines for indentation:
  - Make sure the if clause and the else clause are aligned.
  - The if clause and the else clause are each followed by a block of statements. Make sure the statements in the blocks are consistently indented.



Cont.....

Chris owns an auto repair business and has several employees. If any employee works over .40 hours in a week, he pays them 1.5 times their regular hourly pay rate for all hours over 40. He has asked you to design a simple payroll program that calculates an employee's gross pay, including any overtime wages.

Algorithm:

Get the number of hours worked.
Get the hourly pay rate.
If the employee worked more than 40 hours:
Calculate and display the gross pay with overtime.
Else:
Calculate and display the gross pay as usual.

Output with input

Enter the number of hours worked: **40**
Enter the hourly pay rate: **20**
The gross pay is $800.00.

Program

```
# Variables to represent the base hours and the overtime multiplier.
base_hours = 40 # Base hours per week
ot_multiplier = 1.5 # Overtime multiplier
hours = float(input('Enter the number of hours worked:')) # Get the hours worked.
pay_rate = float(input('Enter the hourly pay rate:')) # Get the hourly pay rate.
# Calculate and display the gross pay.
if hours > base_hours:
    overtime_hours = hours − base_hours  # First, get the number of overtime hours worked.
    overtime_pay = overtime_hours * pay_rate * ot_multiplier  # Calculate the amount of overtime pay.
    gross_pay = base_hours * pay_rate + overtime_pay  # Calculate the gross pay.
else:
     gross_pay = hours * pay_rate # Calculate the gross pay without overtime.
     print('The gross pay is $', format(gross_pay, ',.2f'), sep='') # Display the gross pay.
```

3.3 Comparing Strings

- Python allows you to compare strings. This allows you to create decision structures that test the value of a string.
- We can compare strings. For example:

```
name1 = 'Mary'
name2 = 'Mark'
if name1 = = name2:
    print('The names are the same.')
else:
    print('The names are NOT the same.')
```

- The == operator compares name1 and name2 to determine whether they are equal.
- The != operator is use to determine whether the value referenced by month is not equal to 'October'.

```
if month != 'October':
    print('This is the wrong time for October fest!')
```

- String comparisons are case sensitive. For example, the strings 'saturday' and 'Saturday‘ are not equal because the "s" is lowercase in the first string, but uppercase in the second string.

Cont.....

In addition to determining whether strings are equal or not equal, you can also determine whether one string is greater than or less than another string. This is a useful capability because programmers commonly need to design programs that sort strings in some order.

ASCII codes:

- The uppercase characters A through Z are represented by the numbers 65 through 90.
- The lowercase characters a through z are represented by the numbers 97 through 122.
- When the digits 0 through 9 are stored in memory as characters, they are represented by the numbers 48 through 57.
  (For example, the string 'abc123' would be stored in memory as the codes 97, 98, 99, 49, 50, and 51.)
- A blank space is represented by the number 32.

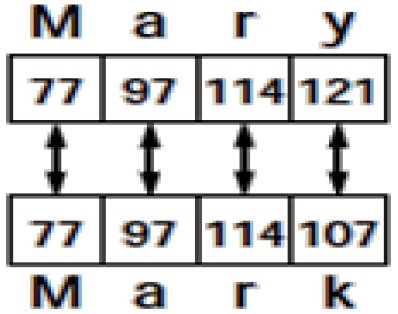
When a program compares characters, it actually compares the codes for the characters. For example

```
if 'a' < 'b':
    print('The letter a is less than the letter b.')
```

The code determines whether the ASCII code for the character 'a' is less than the ASCII code for the character 'b'. The expression 'a' < 'b' is true because the code for 'a' is less than the code for 'b'.

Eg: Program uses the strings 'Mary' and 'Mark' as follows:

```
name1 = 'Mary'
name2 = 'Mark'
if name1 > name2:
    print('Mary is greater than Mark')
else:
    print('Mary is not greater than Mark')
```



The > operator compares each character in the strings 'Mary' and 'Mark', beginning with the first, or leftmost, characters.

Cont.....

If one of the strings in a comparison is shorter than the other, only the corresponding characters will be compared. If the corresponding characters are identical, then the shorter string is considered less than the longer string.

For example, suppose the strings 'High‘ and 'Hi' were being compared. The string 'Hi' would be considered less than 'High‘ because it is shorter.

Program :

```
# This program compares strings with the < operator.
# Get two names from the user.|
name1 = input('Enter a name (last name first):')
name2 = input('Enter another name (last name first):')
print('Here are the names, listed alphabetically.')
# Display the names in alphabetical order.
if name1 < name2:
    print(name1)
    print(name2)
else:
    print(name2)
    print(name1)
```

Output with input :

Enter a name (last name first):**Jones, Richard**
Enter another name (last name first) **Costa, Joan**
Here are the names, listed alphabetically:
Costa, Joan
Jones, Richard

TEST

- **What would the following code display**?

```
if 'z' < 'a':
print('z is less than a.')
else:
print('z is not less than a.')
```

- **What would the following code display?**

```
s1 = 'New York'
s2 = 'Boston'
if s1 > s2:
    print(s2)
    print(s1)
else:
    print(s1)
    print(s2)
```

3.4 Nested Decision Structures

Combining sequence structures with a decision structure

To test more than one condition, a decision structure can be nested inside another decision structure.

Programs are designed as combinations of different control structures. For example, figure shows a flowchart that combines a decision structure with two sequence structures.

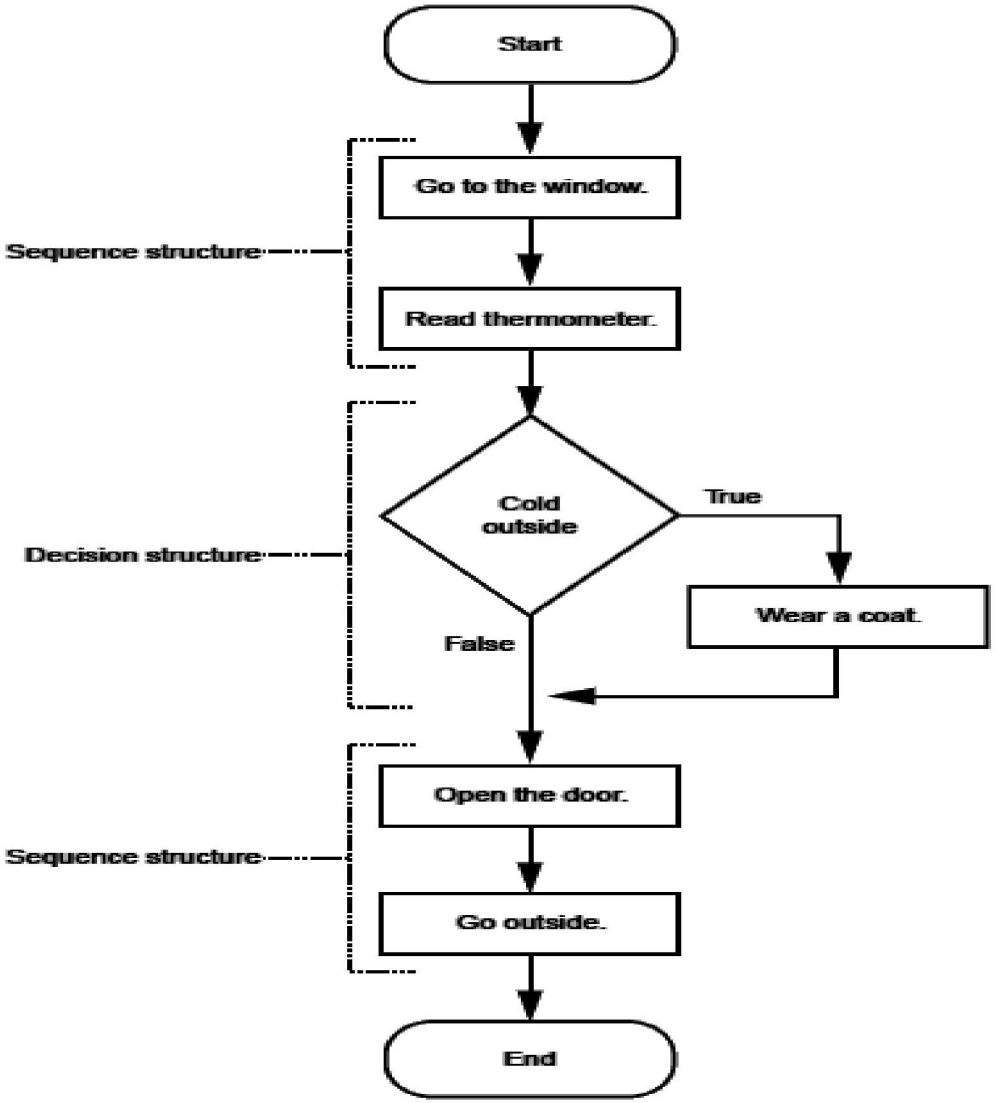


Figure A nested decision structure

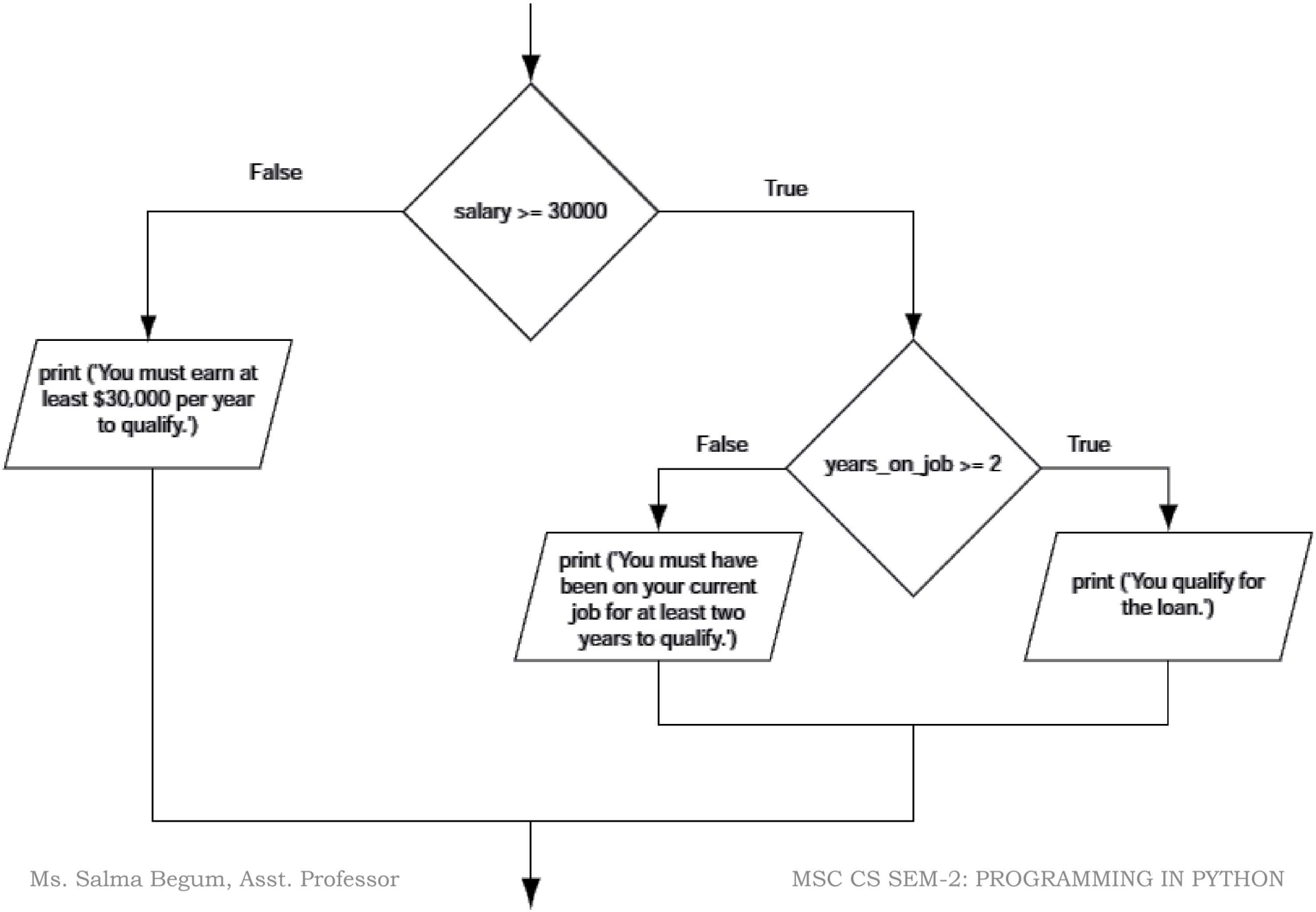
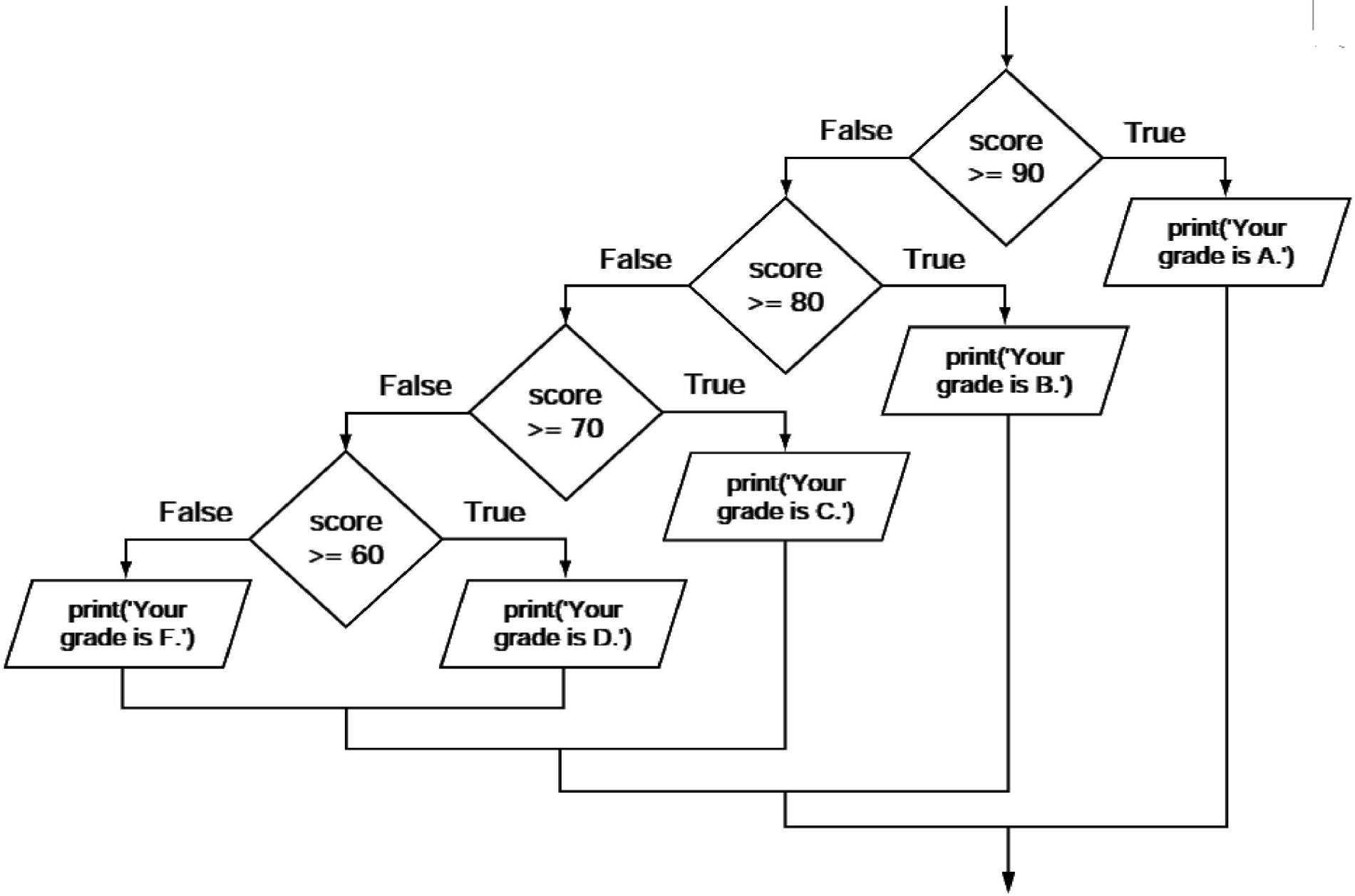


Figure Nested decision structure to determine a grade



3.4 The if-elif-else Statement

Python provides a special version of the decision structure known as the ***if-elif-else*** statement.

General format of the if-elif-else statement:

if *condition_1:*
    statement
    statement
    etc.
elif *condition_2:*
    statement
    statement
    etc.
Insert as many elif clauses as necessary . . .
else:
    statement
    statement
    etc.

- ✓ When the statement executes, *condition_1* is tested. If *condition_1* is
- true, the block of statements that immediately follow is executed, up to the elif clause.The rest of the structure is ignored.
- *false,* the program jumps to the very next elif clause and tests *condition_2.*

If condition_2 is

- *true,* the block of statements that immediately follow is executed, up to the next elif clause. The rest of the structure is then ignored.
- ✓ This process continues until a condition is found to be true, or no more elif clauses are left.
- ✓ If no condition is true, the block of statements following the else
- ✓ clause is executed.

Cont.....

Example for demonstrating if-elif-else statement.

```
if score >= A_score:
    print('Your grade is A.')
elif score >= B_score:
    print('Your grade is B.')
elif score >= C_score:
    print('Your grade is C.')
elif score >= D_score:
    print('Your grade is D.')
else:
    print('Your grade is F.')
```

Nested if-else statements has two particular disadvantages during debugging code:

- The code can grow complex and become difficult to understand.
- Because of the required indentation, a long series of nested if-else statements can become too long to be displayed on the computer screen without horizontal scrolling. Also, long statements tend to "wrap around" when printed on paper, making the code even more difficult to read.

The logic of an if-elif-else statement is easier to follow than a long series of nested if-else statements, because all of the clauses are aligned in an if-elif-else statement, the lengths of the lines in the statement tend to be shorter.

3.5 Logical Operators

- The ***logical and*** operator and the logical or operator allow you to connect multiple Boolean expressions to create a compound expression.
- The ***logical not*** operator reverses the truth of a Boolean expression.
- Python provides a set of operators known as ***logical operators***, *which you can use to create* complex Boolean expressions.

Logical operators

Operator	Meaning
`and`	The `and` operator connects two Boolean expressions into one compound expression. Both subexpressions must be true for the compound expression to be true.
`or`	The `or` operator connects two Boolean expressions into one compound expression. One or both subexpressions must be true for the compound expression to be true. It is only necessary for one of the subexpressions to be true, and it does not matter which.
`not`	The `not` operator is a unary operator, meaning it works with only one operand. The operand must be a Boolean expression. The `not` operator reverses the truth of its operand. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

Cont....

- **The *and* Operator** takes two Boolean expressions as operands and creates a compound Boolean expression that is true only when both subexpressions are true.

  Eg:

  ```
  if temperature < 20 and minutes > 12:
      print('The temperature is in the danger zone.')
  ```

- **The *or* operator** takes two Boolean expressions as operands and creates a compound Boolean expression that is true when either of the subexpressions is true.

  Eg:

  ```
  if temperature < 20 or temperature > 100:
      print('The temperature is too extreme')
  ```

Cont....

Short-Circuit Evaluation

Both the ***and*** & ***or*** operators perform *short-circuit evaluation.*

- With the ***and*** operator: If the expression on the left side of the and operator is false, the expression on the right side will not be checked. Because the compound expression will be false if only one of the subexpressions is false, it would waste CPU time to check the remaining expression. So, when the and operator finds that the expression on its left is false, it shortcircuits and does not evaluate the expression on its right.
- With the ***or*** operator: If the expression on the left side of the or operator is true, the expression on the right side will not be checked. Because it is only necessary for one of the expressions to be true, it would waste CPU time to check the remaining expression.

Cont....

The not Operator is a unary operator that takes a Boolean expression as its operand and reverses its logical value. In other words, if the expression is true, the not operator returns false, and if the expression is false, the not operator returns true.

Eg:

```
if not(temperature > 100):
    print('This is below the maximum temperature.')
```

First, the expression (temperature > 100) is tested and a value of either true or false is the result. Then the not operator is applied to that value.

- If the expression (temperature > 100) is
  - true, the not operator returns false.
  - false, the not operator returns true.

Note: In this example, we have put parentheses around the expression temperature > 100. This is to make it clear that we are applying the not operator to the value of the expression temperature > 100, not just to the temperature variable.

Cont....

In some situations the and operator can be used to simplify nested decision structures.

For example, the following program uses nested if-else statements:

```
if salary >= min_salary:
if years_on_job >= min_years:
print('You qualify for the loan.')
else:
print('You must have been employed', \
'for at least', min_years, \
'years to qualify.')
else:
print('You must earn at least $', \
format(min_salary, ',.2f'), \
' per year to qualify.', sep='')
```

The purpose of this decision structure is to determine that a person's salary is at least $30,000 and that he or she has been at their current job for at least two years.

Program(Using logic Operator 'and')

```
# 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.')
```

Output with input

Enter your annual salary: **35000**

Enter the number of years employed: **1**

You do not qualify for this loan.

Cont....

Checking Numeric Ranges with Logical Operators

- Sometimes you will need to design an algorithm that determines whether a numeric value
- is within a specific range of values or outside a specific range of values.
- The ***and*** operator is used to determine whether a number is inside a range or not.
- For example, the following if statement checks the value in x to determine whether it is in the range of 20 through 40:

```
if x >= 20 and x <= 40:
    print('The value is in the acceptable range.')
```

TEST

- Assume the variables a = 2, b = 4, and c = 6. Circle the T or F for each of the following conditions to indicate whether its value is true or false.

  1. a = = 4 or b > 2
  2. 6 <= c and a > 3
  3. 1 != b and c != 3
  4. a >= −1 or a <= b
  5. not (a > 2)

- Write an if statement that displays the message “The number is valid” if the value referenced by speed is within the range 0 through 200.

- Write an if statement that displays the message “The number is not valid” if the value referenced by speed is outside the range 0 through 200.

3.6 Boolean Variables

A Boolean variable can reference one of two values: True or False.

They are commonly used as flags, which indicate whether specific conditions exist.

- A *flag* is a variable that signals when some condition exists in the program. A flag variable set to:
  - False, indicates the condition does not exist.
  - True, indicates the condition exist.
- In addition to int, float, and str (string) data types, Python also provides a bool data type. The bool data type allows you to create variables that may reference one of two possible values: True or False.

Eg:

```
hungry = True
sleepy = False
```

- For example, suppose a salesperson has a quota of $50,000. Assuming sales references the amount that the salesperson has sold, the following code determines whether the quota has been met:

```
if sales >= 50000.0:
    sales_quota_met = True
else:
    sales_quota_met = False
```

Here, the *sales_quota_met* variable is used as a flag to indicate whether the sales quota has been met.

Cont....

- For example, suppose a salesperson has a quota of $50,000. Assuming sales references the amount that the salesperson has sold, the following code determines whether the quota has been met:

```
if sales >= 50000.0:
    sales_quota_met = True
else:
    sales_quota_met = False
```

Here, the *sales_quota_met* variable is used as a flag to indicate whether the sales quota has been met.

TEST

Multiple Choice

1. A logical design that controls the order in which a set of statements execute is called a _____.

a. control structure b. sequence structure c. logical structure d. relational structure

2. The _____ statement causes one or more statements to execute only when a Boolean expression is _____.

a. else, false b. if, true c. if, false d. else, true

3. A(n) _____ expression has a value of either true or false.
   a. binary
   b. decision
   c. unconditional
   d. Boolean

4. When a program compares characters, it actually compares the _____ codes
   a. ASCII
   b. binary
   c. letter
   d. numeric

5. A(n) _____ structure tests a condition and then takes one path if the condition is true, or another path if the condition is false.
   a. `if` statement
   b. single alternative decision
   c. dual alternative decision
   d. sequence

6. If the expression on the left side of the `and` operator is false, the expression on the right side will not be checked. This is called as _____.
   a. relational operator
   b. logical operator
   c. long-circuit evaluation
   d. short-circuit evaluation

7. You use a(n) _____ statement to write a dual alternative decision structure.
   a. `test-jump`
   b. `if`
   c. `if-else`
   d. `if-call`

8. `and`, `or`, and `not` are _____ operators.
   a. relational
   b. logical
   c. conditional
   d. ternary

9. A compound Boolean expression created with the _____ operator is true only if both of its subexpressions are true.
   a. `and`
   b. `or`
   c. `not`
   d. `both`

10. A compound Boolean expression created with the _____ operator is true if either of its subexpressions is true.
    a. `and`
    b. `or`
    c. `not`
    d. `either`

11. Boolean variables are commonly used as _____ and if it is set to _____, it indicates the condition does not exist.
    a. signals, `False`
    b. signals, `True`
    c. flags, `False`
    d. flags, `True`
12. A _____ is a Boolean variable that signals when some condition exists in the program.
    a. flag
    b. signal
    c. sentinel
    d. siren

True or False

1. It is best to use the `or` operator when determining whether a number is inside a range.
2. A program can be made of only one type of control structure. You cannot combine structures.
3. A single alternative decision structure tests a condition and then takes one path if the condition is true, or another path if the condition is false.
4. A compound Boolean expression created with the `or` operator is true only when one subexpression is true.
5. A compound Boolean expression created with the `and` operator is true only when both subexpressions are true.