Object Oriented Programming II

Dr. Burak Kaleci

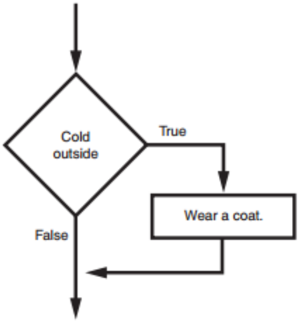
February 7, 2019

Content

Introduction

- A control structure is a logical design that controls the order in which a set of statements execute.
- Generally, programs require control structure: one that can execute a set of statements only under certain circumstances.
- This can be accomplished with a decision structure. (Decision structures are 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.

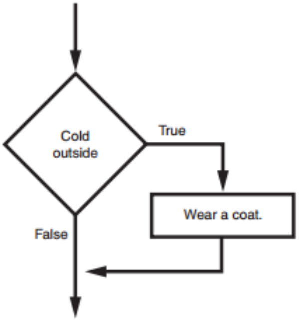
Introduction



- The flowchart shown in Figure shows how the logic of an everyday decision can be diagrammed as a decision structure.
- The diamond symbol represents a true/ false condition.
- In the flowchart, the diamond symbol indicates some condition that must be tested.
- In this case, we are determining whether the condition Cold outside is true or false.
- If this condition is true, the action Wear a coat is performed.
- If the condition is false, the action is skipped.

Introduction

- Programmers call the type of decision structure shown in Figure 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.



Introduction

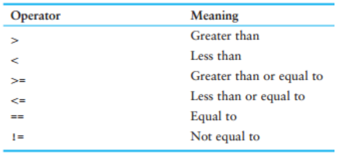
- Here is the general format of the if statement:

```
if condition:
    statement
    statement
    statement
```

- The if statement 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.
- Beginning at the next line is a block of statements.
- **All of the statements in a block must be consistently indented.**
- **This indentation is required because the Python interpreter uses it to tell where the block begins and ends.**

Boolean Expressions and Relational Operators

- The expressions that are tested by the if statement are called Boolean expressions.
- Typically, the 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.
- For example, the greater than operator ($>$) determines whether one value is greater than another.
- The equal to operator ($==$) determines whether two values are equal.
- Table lists the relational operators that are available in Python.



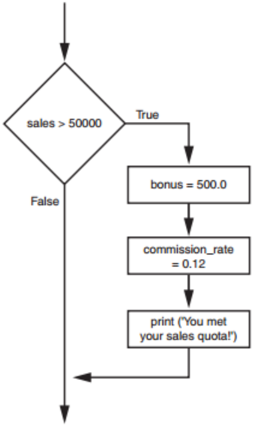
Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

An example

- Let's look at the following example of the if statement:

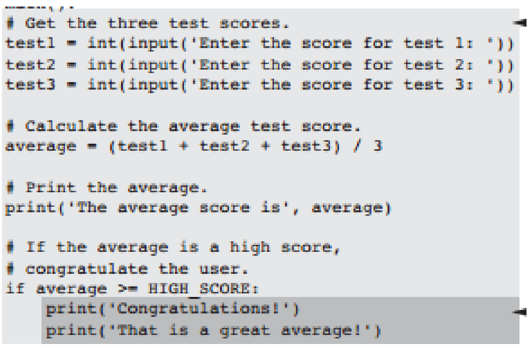
```
if sales > 50000:
    bonus = 500.0
    commission_rate = 0.12
    print('You met your sales quota!')
```

- This statement uses the > operator to determine whether sales is greater than 50,000.



Nested Blocks

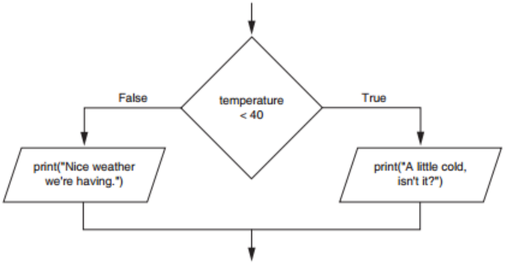
- Python requires you to indent the statements in a block.
- When you have a block nested inside a block, the inner block must be further indented.



```
# 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!')
```

Introduction

- Now we will look at 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.
- Figure shows a flowchart for a dual alternative decision structure.
- The decision structure in the flowchart tests the condition temperature < 40. If this condition is true, the statement print("A little cold, isn't it?")is performed.
- If the condition is false, the statement print("Nice weather we're having.")is performed.


temperature < 40
False
True
print("Nice weather we're having.")
print("A little cold, isn't it?")

Introduction

- In code we write a dual alternative decision structure as an **if-else statement**.
- Here is the general format of the if-else statement:

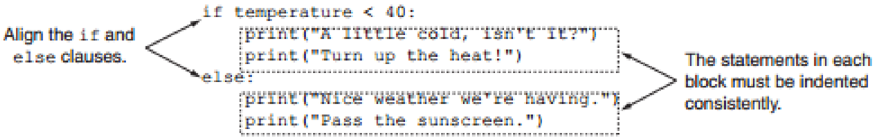
```
if condition:
    statement
    statement
    statement
else:
    statement
    statement
    statement
```

- When this statement executes, the condition is tested. If it is true, the block of indented statements following the if clause is executed, and then control of the program jumps to the statement that follows the if-else statement.
- If the condition is false, the block of indented statements following the else clause is executed, and then control of the program jumps to the statement that follows the if-else statement.

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.**



```
if temperature < 40:
    print("A little cold, isn't it?")
    print("Turn up the heat!")
else:
    print("Nice weather we're having.")
    print("Pass the sunscreen.")
```

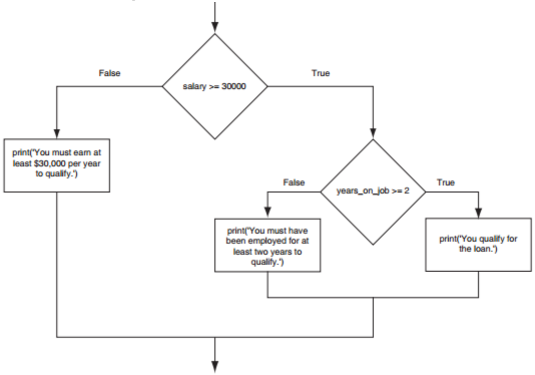
Align the if and else clauses.
The statements in each block must be indented consistently.

Introduction

- You can also nest decision structures inside other decision structures.
- In fact, this is a common requirement in programs that need to test more than one condition.
- For example, consider a program that determines whether a bank customer qualifies for a loan.
- To qualify, two conditions must exist: (1) the customer must earn at least $30,000$ per year, and (2) the customer must have been employed for at least two years.

An Example

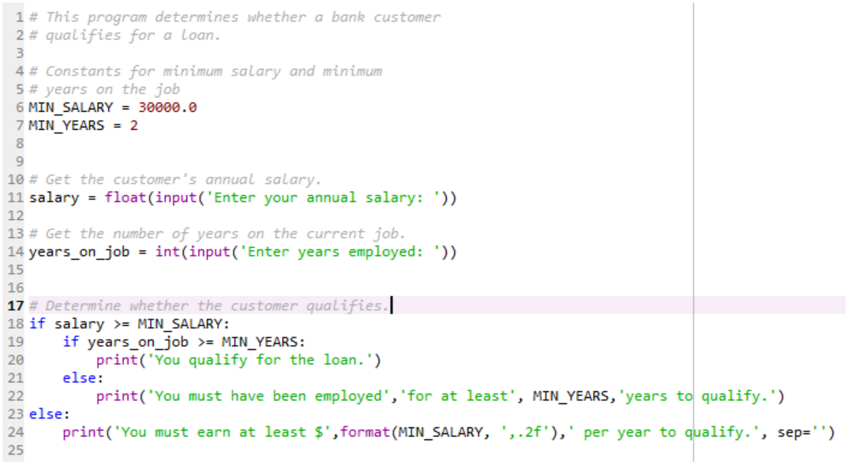
Figure shows a flowchart for an algorithm that could be used in such a program.


False
salary >= 30000
True
print('You must earn at least $30,000 per year to qualify.')
False
years_on_job >= 2
True
print('You must have been employed for at least two years to qualify.')
print('You qualify for the loan.')

An Example

- If we follow the flow of execution, we see that the condition salary $>=$ 30000 is tested.
- If this condition is false, there is no need to perform further tests; we know that the customer does not qualify for the loan.
- If the condition is true, however, we need to test the second condition. This is done with a nested decision structure that tests the condition years_on_job $>=$ 2.
- If this condition is true, then the customer qualifies for the loan. If this condition is false, then the customer does not qualify.

An Example



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

# Constants for minimum salary and minimum
# years on the job
MIN_SALARY = 30000.0
MIN_YEARS = 2


# 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 years employed: '))


# Determine whether the customer qualifies.
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='')

```

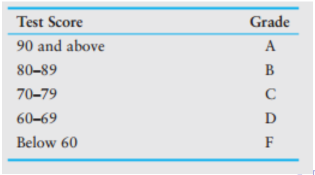
An Example

- Look at the if-else statement that begins in line 18. It tests the condition salary >= MIN_SALARY.
- If this condition is true, the if-else statement that begins in line 19 is executed.
- Otherwise the program jumps to the else clause in line 23 and executes the statement in line 24.
- It's important to use proper indentation in a nested decision structure. Not only is proper indentation required by the Python interpreter, but it also makes it easier for you, the human reader of your code, to see which actions are performed by each part of the structure.
- Follow these rules when writing nested if statements:
  - **Make sure each else clause is aligned with its matching if clause.**
  - **Make sure the statements in each block are consistently indented.**

Testing a Series of Conditions

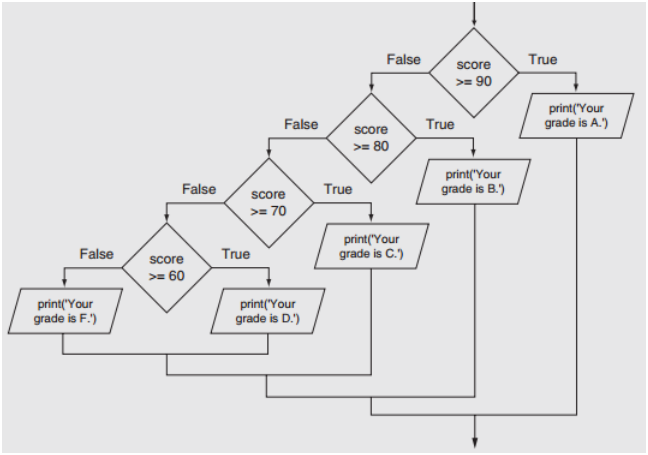
Problem: Consider we use the following grading scale. Write a Python program to achieve that task.

- If the score is greater than or equal to 90, then the grade is A.
- Else, if the score is greater than or equal to 80, then the grade is B.
- Else, if the score is greater than or equal to 70, then the grade is C.
- Else, if the score is greater than or equal to 60, then the grade is D.
- Else, the grade is F.

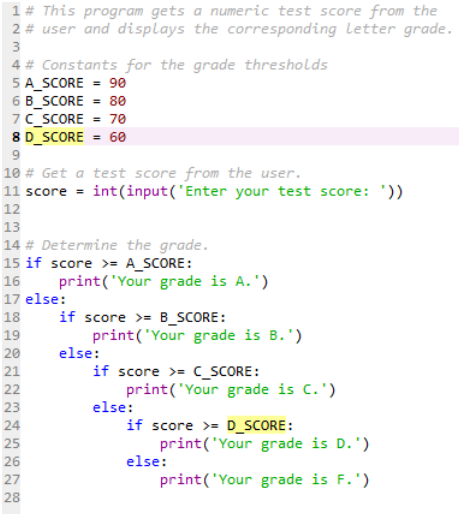


Test Score	Grade
90 and above	A
80–89	B
70–79	C
60–69	D
Below 60	F

Testing a Series of Conditions


score >= 90
False
True
print('Your grade is A.')
score >= 80
False
True
print('Your grade is B.')
score >= 70
False
True
print('Your grade is C.')
score >= 60
False
True
print('Your grade is D.')
print('Your grade is F.')

Testing a Series of Conditions



```
# This program gets a numeric test score from the
# user and displays the corresponding letter grade.

# Constants for the grade thresholds
A_SCORE = 90
B_SCORE = 80
C_SCORE = 70
D_SCORE = 60

# Get a test score from the user.
score = int(input('Enter your test score: '))


# Determine the grade.
if score >= A_SCORE:
    print('Your grade is A.')
else:
    if score >= B_SCORE:
        print('Your grade is B.')
    else:
        if score >= C_SCORE:
            print('Your grade is C.')
        else:
            if score >= D_SCORE:
                print('Your grade is D.')
            else:
                print('Your grade is F.')

```

The if-elif-else Statement

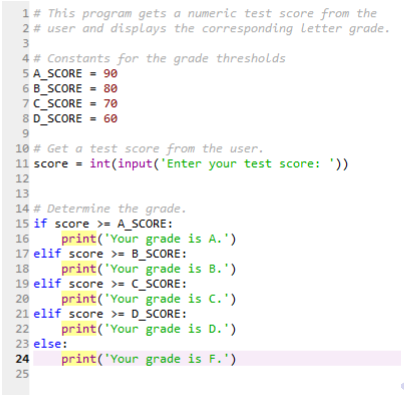
- Here is the general format of the if-elif-else statement:

```
if condition_1:
    statement
    statement
    etc.
elif condition_2:
    statement
    statement
    etc.
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.
- If condition_1 is false, however, the program jumps to the very next elif clause and tests condition_2. If it 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.

The if-elif-else Statement

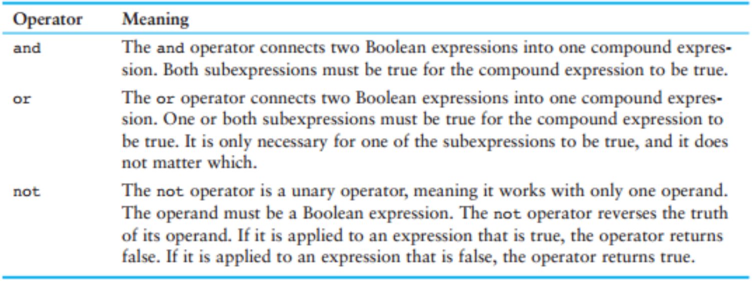
Notice the alignment and indentation that is used with the if-elif-else statement: The if, elif, and else clauses are all aligned, and the conditionally executed blocks are indented.



```python
# This program gets a numeric test score from the
# user and displays the corresponding letter grade.

# Constants for the grade thresholds
A_SCORE = 90
B_SCORE = 80
C_SCORE = 70
D_SCORE = 60

# Get a test score from the user.
score = int(input('Enter your test score: '))


# Determine the grade.
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.')

```

Introduction

- Python provides a set of operators known as logical operators, which you can use to create complex Boolean expressions.
- Table describes these operators.

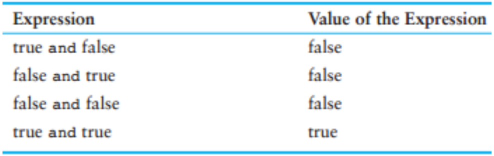


Operator	Meaning
and	The and operator connects two Boolean expressions into one compound expression. Both subexpressions must be true for the compound expression to be true.
or	The or operator connects two Boolean expressions into one compound expression. One or both subexpressions must be true for the 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.

The and Operator

- The and operator takes two Boolean expressions as operands and creates a compound Boolean expression that is true only when both subexpressions are true.
- The following is an example of an if statement that uses the and operator:

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

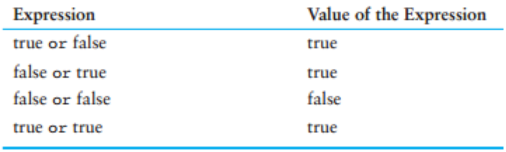


Expression	Value of the Expression
true and false	false
false and true	false
false and false	false
true and true	true

The or Operator

- 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.
- The following is an example of an if statement that uses the or operator:

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



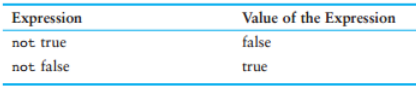
Expression	Value of the Expression
true or false	true
false or true	true
false or false	false
true or true	true

The not Operator

- 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.
- The following is an example of an if statement that uses the not operator:

```
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.



Expression	Value of the Expression
not true	false
not false	true

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.
- When determining whether a number is inside a range, it is best to use the and operator.
- 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.')
```

- When determining whether a number is outside a range, it is best to use the or operator.
- The following statement determines whether x is outside the range of 20 through 40:

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

Introduction

- So far in this book we have worked with int, float, and str(string) variables.
- In addition to these 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.
- Boolean variables are most commonly used as flags.
- A flag is a variable that signals when some condition exists in the program.
- When the flag variable is set to False, it indicates the condition does not exist.
- When the flag variable is set to True, it means the condition does exist.

Introduction

- 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:
    sales_quota_met = True
else:
    sales_quota_met = False
```

- As a result of this code, the sales_quota_met variable can be used as a flag to indicate whether the sales quota has been met.

Introduction

- Later in the program we might test the flag in the following way:

```
if sales_quota_met:
    print('You have met your sales quota!')
```

- This code displays 'You have met your sales quota!' if the bool variable sales_quota_met is True.
- Notice that we did not have to use the == operator to explicitly compare the sales_quota_met variable with the value True.
- This code is equivalent to the following:

```
if sales_quota_met == True:
    print('You have met your sales quota!')
```