CN101

Lecture 4-5

Decision Structures and Boolean Logic

Topics

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

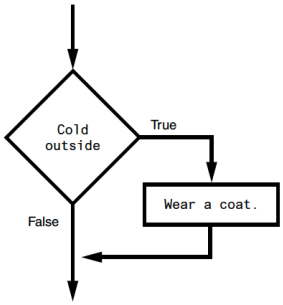
The `if` Statement

- 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.)



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

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.)

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?

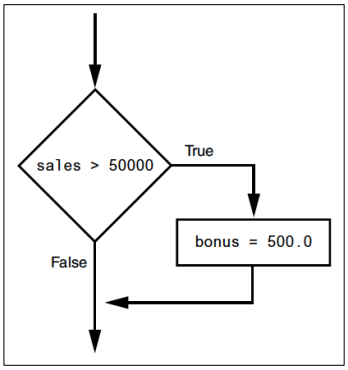
Example

```
1  >>> x = 1 Enter
2  >>> y = 0 Enter
3  >>> y < x Enter
4  True
5  >>> x < y Enter
6  False
7  >>>
```

```
1  >>> x = 1 Enter
2  >>> y = 0 Enter
3  >>> z = 1 Enter
4  >>> x >= y Enter
5  True
6  >>> x >= z Enter
7  True
8  >>> x <= z Enter
9  True
10 >>> x <= y Enter
11 False
12 >>>
```

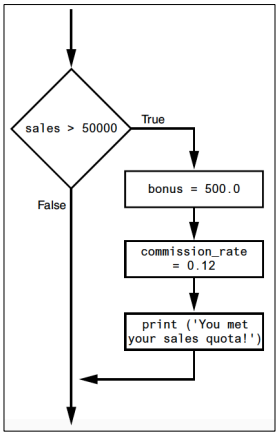

Putting It All Together

```
if sales > 50000:
    bonus = 500.0
```



Putting It All Together

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



Program 3-1 (test_average.py)

```
1  # This program gets three test scores and displays
2  # their average. It congratulates the user if the
3  # average is a high score.
4
5  # The HIGH_SCORE named constant holds the value that is
6  # considered a high score.
7  HIGH_SCORE = 95
8
9  # Get the three test scores.
10 test1 = int(input('Enter the score for test 1: ' ))
11 test2 = int(input('Enter the score for test 2: ' ))
12 test3 = int(input('Enter the score for test 3: ' ))
13
14 # Calculate the average test score.
15 average = (test1 + test2 + test3) / 3
16
17 # Print the average.
18 print('The average score is', average)
19
20 # If the average is a high score,
21 # congratulate the user.
22 if average >= HIGH_SCORE:
23     print('Congratulations!')
24     print('That is a great average!')
```


Program Output (with input shown in bold)

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

Program Output (with input shown in bold)

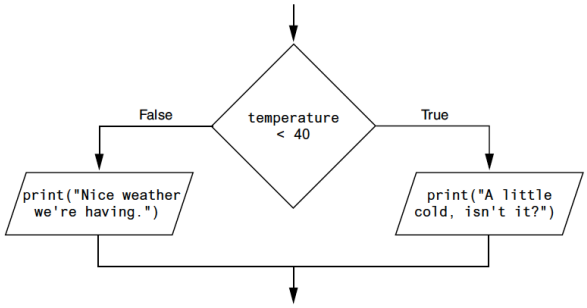
```
Enter the score for test 1: 93 Enter
Enter the score for test 2: 99 Enter
Enter the score for test 3: 96 Enter
The average score is 96.0
Congratulations!
That is a great average!
```


The `if-else` Statement

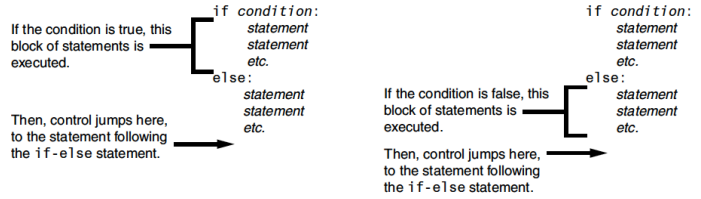
- Dual alternative decision structure: two possible paths of execution
  - One is taken if the condition is true, and the other if the condition is false
  - Syntax:
    ```
    if condition:
        statements
    else:
        other statements
    ```
  - `if` clause and `else` clause must be aligned
  - Statements must be consistently indented

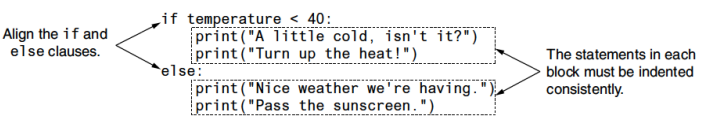
```
if condition:
    statement
    statement
    etc.
else:
    statement
    statement
    etc.
```


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



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





Program 3-2 (auto_repair_payroll.py)

```
1  # Named constants to represent the base hours and
2  # the overtime multiplier.
3  BASE_HOURS = 40      # Base hours per week
4  OT_MULTIPLIER = 1.5  # Overtime multiplier
5
6  # Get the hours worked and the hourly pay rate.
7  hours = float(input('Enter the number of hours worked: '))
8  pay_rate = float(input('Enter the hourly pay rate: '))
9
10 # Calculate and display the gross pay.
11 if hours > BASE_HOURS:
12     # Calculate the gross pay with overtime.
13     # First, get the number of overtime hours worked.
14     overtime_hours = hours - BASE_HOURS
15
16     # Calculate the amount of overtime pay.
17     overtime_pay = overtime_hours * pay_rate * OT_MULTIPLIER
18
19     # Calculate the gross pay.
20     gross_pay = BASE_HOURS * pay_rate + overtime_pay
21 else:
22     # Calculate the gross pay without overtime.
23     gross_pay = hours * pay_rate
24
25 # Display the gross pay.
26 print('The gross pay is $', format(gross_pay, '.2f'), sep='')
```

Program Output (with input shown in bold)

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

Program Output (with input shown in bold)

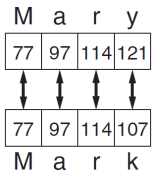
```
Enter the number of hours worked: 50 Enter
Enter the hourly pay rate: 20 Enter
The gross pay is $1,100.00.
```


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.)

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



Program 3-3 (password.py)

```
1  # This program compares two strings.
2  # Get a password from the user.
3  password = input('Enter the password: ')
4
5  # Determine whether the correct password
6  # was entered.
7  if password == 'prospero':
8      print('Password accepted.')
9  else:
10     print('Sorry, that is the wrong password.')
```

Program Output (with input shown in bold)

```
Enter the password: ferdinand Enter
Sorry, that is the wrong password.
```

Program Output (with input shown in bold)

```
Enter the password: prospero Enter
Password accepted.
```


Program 3-4 (sort_names.py)

```
1  # This program compares strings with the < operator.
2  # Get two names from the user.
3  name1 = input('Enter a name (last name first): ')
4  name2 = input('Enter another name (last name first): ')
5
6  # Display the names in alphabetical order.
7  print('Here are the names, listed alphabetically.')
8
9  if name1 < name2:
10     print(name1)
11     print(name2)
12 else:
13     print(name2)
14     print(name1)
```

Program Output (with input shown in bold)

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


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

Program 3-5 (loan_qualifier.py)

```
1  # This program determines whether a bank customer
2  # qualifies for a loan.
3
4  MIN_SALARY = 30000.0 # The minimum annual salary
5  MIN_YEARS = 2        # The minimum years on the job
6
7  # Get the customer's annual salary.
8  salary = float(input('Enter your annual salary: '))
9
10 # Get the number of years on the current job.
11 years_on_job = int(input('Enter the number of' +
12                           'years employed: '))
13
14 # Determine whether the customer qualifies.
15 if salary >= MIN_SALARY:
16       if years_on_job >= MIN_YEARS:
17           print('You qualify for the loan.')
18       else:
19           print('You must have been employed',
20                 'for at least', MIN_YEARS,
21                 'years to qualify.')
22 else:
23       print('You must earn at least $',
24             format(MIN_SALARY, ',.2f'),
25             ' per year to qualify.', sep='')
```


Program Output (with input shown in bold)

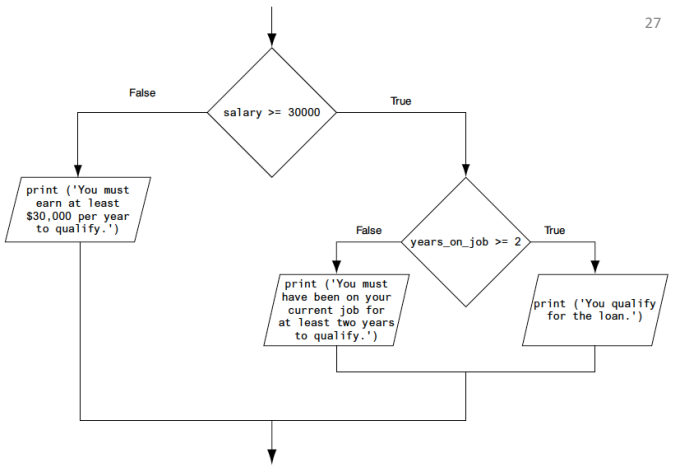
```
Enter your annual salary: 35000 Enter
Enter the number of years employed: 1 Enter
You must have been employed for at least 2 years to qualify.
```

Program Output (with input shown in bold)

```
Enter your annual salary: 25000 Enter
Enter the number of years employed: 5 Enter
You must earn at least $30,000.00 per year to qualify.
```

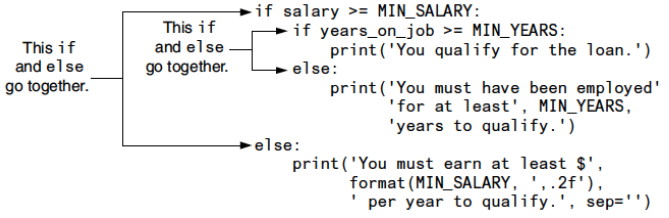
Program Output (with input shown in bold)

```
Enter your annual salary: 35000 Enter
Enter the number of years employed: 5 Enter
You qualify for the loan.
```

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

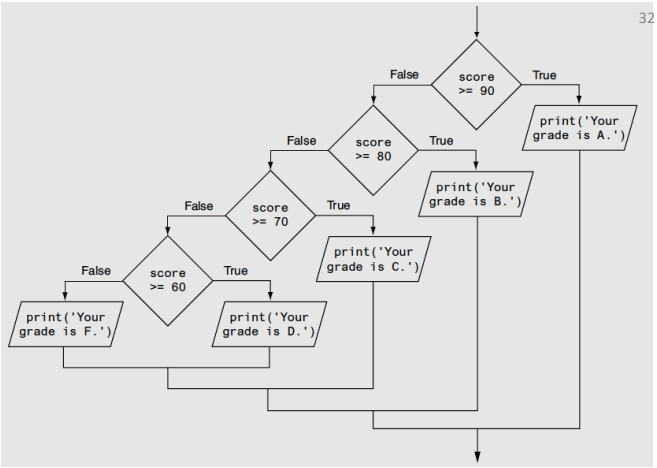
- `if-elif-else` statement: special version of a decision structure
  - Makes logic of nested decision structures simpler to write
    - Can include multiple `elif` statements
  - Syntax:

```
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



Program 3-6 (grader.py)

```
1  # This program gets a numeric test score from the
2  # user and displays the corresponding letter grade.
3
4  # Named constants to represent the grade thresholds
5  A_SCORE = 90
6  B_SCORE = 80
7  C_SCORE = 70
8  D_SCORE = 60
9
10 # Get a test score from the user.
11 score = int(input('Enter your test score: '))
12
13 # Determine the grade.
14 if score >= A_SCORE:
15     print('Your grade is A.')
16 else:
17     if score >= B_SCORE:
18         print('Your grade is B.')
19     else:
20         if score >= C_SCORE:
21             print('Your grade is C.')
22         else:
23             if score >= D_SCORE:
24                 print('Your grade is D.')
25             else:
26                 print('Your grade is F.')
```


Program Output (with input shown in bold)

```
Enter your test score: 78 Enter
Your grade is C.
```

Program Output (with input shown in bold)

```
Enter your test score: 84 Enter
Your grade is B.
```

```
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.')
```


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 or False	False
False or True	True
True or False	True
True or 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
not True	False
not 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`
  - Can also be written as: `10 <= x <= 20`
- To determine whether a numeric value is outside of a specific range of values, use `or`
  - Example: `x < 10 or x > 20`

Program 3-7 (loan_qualifier2.py)

```
1  # This program determines whether a bank customer
2  # qualifies for a loan.
3
4  MIN_SALARY = 30000.0 # The minimum annual salary
5  MIN_YEARS = 2        # The minimum years on the job
6
7  # Get the customer's annual salary.
8  salary = float(input('Enter your annual salary: '))
9
10 # Get the number of years on the current job.
11 years_on_job = int(input('Enter the number of ' +
12                           'years employed: '))
13
14 # Determine whether the customer qualifies.
15 if salary >= MIN_SALARY and years_on_job >= MIN_YEARS:
16     print('You qualify for the loan.')
17 else:
18     print('You do not qualify for this loan.')
```


Program Output (with input shown in bold)

```
Enter your annual salary: 35000 Enter
Enter the number of years employed: 1 Enter
You do not qualify for this loan.
```

Program Output (with input shown in bold)

```
Enter your annual salary: 25000 Enter
Enter the number of years employed: 5 Enter
You do not qualify for this loan.
```

Program Output (with input shown in bold)

```
Enter your annual salary: 35000 Enter
Enter the number of years employed: 5 Enter
You qualify for the loan.
```


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

Boolean Variables (cont'd)

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

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

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


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