**Chapter 3 – Decision Structures and Boolean Logic**

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.
   1. Introduction
      1. Control structure
         1. Logical design that controls the order statements execute
      2. Only used sequence structure so far
         1. A set of statements that execute in the order they appear
         2. Used heavily in programming but cannot do everything
            1. I.e. calculating overtime pay

Uses *decision structure*

* + 1. Decision Structure
       1. Executes statements only under certain circumstances
       2. Also known as selection structures
       3. A specific action is performed only if a certain condition exists
          1. If the condition doesn’t exist, the action isn’t performed
          2. Figure 3-1, pg. 110
       4. Figure 3-1 is a single alternative decision structure
          1. Only one alternative execution path
       5. Format for coding found on pg. 110
          1. Starts with *if* clause followed by the condition
          2. Next line starts with a *block* of statements

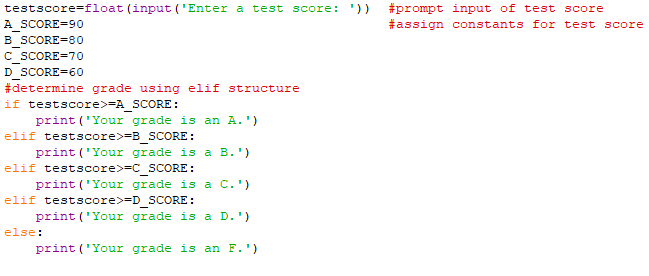
A block is a set of statements that belong together as a group

Must be indented for Python interpreter to understand where block starts and ends

* 1. Boolean Expressions and Relational Operators
     1. Expressions tested by the *if* statement are Boolean expressions

1. 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.
   1. Introduction
      1. Also called dual alternative decision structure
      2. Figure 3-5, pg. 118 shows the general format
      3. When statement executes, the condition is tested
         1. If condition is true, *if* block is executed
         2. If condition is false, *else* block is executed
         3. Figure 3-6, pg. 119
   2. Indentation in the *if-else* Statement
      1. Guidelines:
         1. *If* and *else* clausemust be aligned
         2. Block of statements following clause must be indented consistently
         3. Figure 3-7, pg. 119
      2. Program 3-2, pg. 120 shows example
2. Comparing Strings – Python allows you to compare strings. This allows you to create decision structures that test the value of strings.
   1. Introduction
      1. Strings can also be compared like numbers
      2. Program 3-3 shows how Boolean expressions can be used for passwords, pg. 121
      3. String comparisons are case sensitive
   2. Other String Comparisons
      1. Besides equal or not equal, strings can also be compared as greater than or less than
      2. Python takes the ASCII number used to store each character in order to sort them
      3. Program 3-4, pg. 124 shows this program
3. Nested Decision Structures and the *if-elif-else* Statement – To test more than one condition, a decision structure can be nested inside another decision structure.
   1. Introduction
      1. Figure 3-10, pg. 125 is a flowchart that combines a decision structure with two sequence structures
      2. Often, structures are nested inside other structures
      3. Decision structures can also be nested inside other decision structures
         1. Common requirement in programs that test more than one condition
      4. Important to use proper indentation
         1. Required by interpreter
         2. Also makes it easier to read and see which actions are performed by which structure
         3. Rules for writing nested if functions:
            1. Make sure each *else* clause is aligned with the *if* clause
            2. Make sure blocks are properly indented

Figure 3-14, pg. 129 highlights proper indenting

* 1. Testing a Series of Conditions
     1. Introduction
        1. Not uncommon for a program to have a series of conditions to test
           1. Then perform an action based on which condition is true
           2. Accomplished by having a decision structure with many decision structures nested inside
  2. The *if-elif-else* Statement
     1. Introduction
        1. Helps condense down nested decision structure
        2. *If, elif,* and *else* clauses are lined up
           1. Blocks stay indented
        3. If-elif-else statements aren’t required because it can be coded as if-else statements
           1. Two disadvantages to using nested if-else statements:

Code can grow complex and hard to read

Constant indents can cause the code to become too long horizontally

Makes it hard to read without scrolling

Can get messed up when printed

* + - 1. IEE statements tend to be easier to follow along because it’s all aligned

1. 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 reveres the truth Boolean expression.
   1. Introduction
      1. *Logical operators*
         1. Set of operators used to create complex Boolean expressions
         2. Table 3-3, pg. 133 describe operators
         3. Table 3-4 shows how Boolean expressions can be compounded
      2. The *and* Operator
         1. Takes two Boolean expressions as operands and creates a compound expression
         2. Only true when both operands are true
      3. The *or* Operator
         1. Tales two Boolean expressions as operands and creates a compound expression
         2. True if one of the operands are true
         3. Only one side of expression needs to be true
      4. Short-Circuit Evaluation
         1. Both *and* and *or* perform short-circuit evaluation
         2. If the left side of the *and* operator is false, the expression on the right will not be checked
            1. Short-circuits and does not evaluate the expression on the right
         3. If the right side of the *or* operator is true, the operator will short-circuit and not check the left
      5. The *not* Operator
         1. A unary operator that takes a Boolean expression as its operand and reverses its logical value
         2. If expression is true, operator will report false
         3. If expression is false, operator will report true
      6. Checking Numeric Ranges with Logical Operators
         1. Best to use *and* operator to check whether a number is within a specific range
         2. Use *or* to test if a number is outside a range
2. Boolean Variables – A Boolean variable can reference one of two values: True or False. Boolean variables are commonly used as flags, which indicate whether specific conditions exist.