The Practice of Computing Using

PYTHON

William Punch  Richard Enbody

Chapter 2

Control
Control: A Quick Overview
Selection
Selection

• Selection is how programs make choices, and it is the process of making choices that provides a lot of the power of computing.
**FIGURE 2.1** Sequential program flow.
Python if Statement

• if boolean expression :
  • suite

• evaluate the boolean (True or False)
• if True, execute all statements in the suite
FIGURE 2.2 Decision making flow of control.
Relational Operators

- Less than: <
- Greater than: >
- Equal to: ==  (Not the same as =)
- Not equal to: !=
- Less than or equal to: <=
- Greater than or equal to: >=
Warning About Indentation

• Elements of the “suite” must all be indented the same number of spaces/tabs
• Python only recognizes suites when they are indented the same “distance”
• You must be careful to get the indentation right to get suites right.
Python Selection, Round 2

if boolean expression:
    suite1
else:
    suite2

The process is:
• evaluate the boolean
• if True, run suite1
• if False, run suite2
Repetition: A Quick Overview
Repeating Statements

• Besides selecting which statements to execute, a fundamental need in a program is repetition
  – repeat a set of statements under some conditions

• With both selection and repetition, we have the two most necessary programming statements
While and For Statements

• The while statement is the more general repetition construct. It repeats a set of statements while some condition is True.

• The for statement is useful for iteration, moving through all the elements of data structure, one at a time.
while Loop

• Top-tested loop (pretest)
  – test the boolean before running
  – test the boolean before each iteration of the loop

while boolean expression:
  statementSuite
FIGURE 2.3 *while* loop.
Repeat While the Boolean is True

• while loop will repeat the statements in the suite while the boolean is True (or its Python equivalent)
• If the boolean expression never changes during the course of the loop, the loop will continue forever.
For and Iteration

• One of Python’s strengths is its rich set of built-in data structures

• The for statement is a common statement for manipulation of a data structure
  – for each element in the data structure
    • perform some operation on that element
**FIGURE 2.4** Operation of a *for* loop.
Perfect Number Example
A Perfect Number

• numbers and their factors were mysterious to the Greeks and early mathematicians
• They were curious about the properties of numbers as they held some significance
• A perfect number is a number whose sum of factors (excluding the number) equals the number
• First perfect number is: 6 (1+2+3)
Abundant, Deficient

• abundant numbers summed to more than the number.
  – 12: 1+2+3+4+6 =16

• deficient numbers summed to less than the number.
  – 13: 1
Design

• prompt for a number
• for the number, collect all the factors
• once collected, sum up the factors
• compare the sum and the number and respond accordingly
Code Listing 2.14
Finding Factors
divisor = 1
sumOfDivisors = 0
while divisor < theNum:
    if theNum % divisor==0:  # divisor evenly
divi
des
th
e
um
# divides theNum
des
e
um
sumOfDivisors = sumOfDivisors + divisor
divisor = divisor + 1
Improving the Perfect Number Program

Work with a range of numbers
For each number in the range of numbers:
  • collect all the factors
  • once collected, sum up the factors
  • compare the sum and the number and respond accordingly

Print a summary
Code Listing 2.16
Examine a Range of Numbers
topNumStr = raw_input("What is the upper number for the range:")
topNum = int(topNumStr)
theNum=2
while theNum <= topNum:
    # sum up the divisors, see Code Listing 2.14
    if theNum == sumOfDivisors:
        print theNum,"is perfect"
    theNum += 1
Code Listing 2.18
(Focus on Number Classification)
topNum = raw_input("Upper range number:")

topNum = int(topNum)

theNum=2

while theNum <= topNum:
    # sum up the divisors, see Code Listing 2.14
    # classify the number based on its divisor sum
    if theNum == sumOfDivisors:
        print theNum,"is perfect"
    if theNum < sumOfDivisors:
        print theNum,"is abundant"
    if theNum > sumOfDivisors:
        print theNum,"is deficient"

    theNum += 1
Control In Depth
Booleans
Boolean Expressions

• George Boole’s (mid-1800’s) mathematics of logical expressions
• Boolean expressions (conditions) have a value of True or False
• Conditions are the basis of choices in a computer, and, hence, are the basis of the appearance of intelligence in them.
Boolean Expressions

- Every boolean expression has the form:
  - expression booleanOperator expression
- The result of evaluating something like the above is also just True or False.
Relational Operators

• In Python 2.x, you can compare different types and get an answer
  – just don’t do it! Weird answers (fixed in 3.x)
• Relational Operators have low preference
  • $5 + 3 < 3 - 2$
  • $8 < 1$
  • False
Examples

• If the value of integer myInt is 5, then the value of expression myInt < 7 is
  – True
• If the value of char myChar is 'A', then the value of expression myChar == 'Q' is
  – False
Chained Comparisons

• In python, chained comparisons work just like you would expect in a mathematical expression:
• Given myInt has the value 5
  – 0 <= myInt <= 5
  – True
  – 0 < myInt <= 3
  – False
Pitfall

• Be careful of floating point equality comparisons, especially with zero, e.g. `myFloat==0`.
• Result = `2.0/2.0000000000000001`
• Result == 1.0
  – True
Compound Expressions

• Logically $0 < X < 3$ is actually $(0 < X) \text{ and } (X < 3)$

• Logical Operators (lower case)
  – and
  – or
  – not
Truth Tables

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>not p</th>
<th>p and q</th>
<th>p or q</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Truth Tables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>not p</th>
<th>p and q</th>
<th>p or q</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td></td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td></td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td></td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>
# Truth Tables

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>not p</th>
<th>p and q</th>
<th>p or q</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Compound Evaluation

- Logically $0 < X < 3$ is actually $(0 < X)$ and $(X < 3)$
- Evaluate using $X$ with a value of 5: $(0 < X)$ and $(X < 3)$
- Parenthesis first: (True) and (False)
- Final value: False
- Note: parentheses are not necessary in this case.
Advanced Search on Google

- Google search uses Booleans
- by default, all terms are and’ed together
- you can specify or (using OR)
- you can specify not (using -)
- Example is:
  - punch bill OR william -gates
More on Assignments
Remember Assignments?

• Format: lhs = rhs
• Behavior:
  – expression in the rhs is evaluated producing a value
  – the value produced is placed in the location indicated on the lhs
Can do Multiple Assignments

- \( x, y = 2, 3 \) # assigns \( x=2 \) and \( y=3 \)
- print \( x, y \) # prints 2 3
Swap

• Initial values: X is 2, Y is 3
• Behavior: swap values of X and Y
  – Note: X=Y Y=X doesn’t work
  – introduce extra variable “temp”
    • temp = X  // save X’s value in temp
    • X=Y      // assign Y’s value to X
    • Y=temp   // assign temp’s value to Y
Swap Using Multiple Assignment

• $x, y = 2, 3$
  – print $x, y$  # prints 2 3

• $x, y = y, x$
  – print $x, y$  # prints 3 2
Chaining

- \( x = y = 5 \)
- `print x, y`  # prints 5  5
More Control: Selection
Have Seen 2 Forms of Selection

if boolean expression:
    suite

if boolean expression:
    suite
else:
    suite
Python Selection, Round 3

if boolean expression1:
    suite1
elif boolean expression2:
    suite2
(as many elif’s as you want)
else:
    suiteLast
if, elif, else, the Process

• evaluate boolean expressions until:
  – the boolean expression returns True
  – none of the boolean expressions return True

• if a boolean returns True, run the corresponding suite. Skip the rest of the if

• if no boolean returns True, run the else suite, the default suite
Code Listing 2.20
Updated Perfect Number classification
# classify the number based on its divisor sum
if theNum == sumOfDivisors:
    print theNum,"is perfect"
elif theNum < sumOfDivisors:
    print theNum,"is abundant"
else:
    print theNum,"is deficient"
theNum += 1
More Control: Repetition
While Loop, Round Two

• while loop, oddly, can have an associated else statement
• else statement is executed when the loop finishes under normal conditions
  – basically the last thing the loop does as it exits
While with Else

```
while booleanExpression:
    suite
    suite
else:
    suite
    suite
rest of the program
```
FIGURE 2.9 while-else.
Break Statement

• A break statement in a loop, if executed, exits the loop
• It exists immediately, skipping whatever remains of the loop and the else statement (if it exists) of the loop
Continue Statement

• A continue statement, if executed in a loop, means to immediately jump back to the top of the loop and re-evaluate the conditional.

• Any remaining parts of the loop are skipped for the one iteration when the continue was executed.
Change in Control: Break and Continue

- While loops are easiest read when the conditions of exit are clear.
- Excessive use of continue and break within a loop suite make it more difficult to decide when the loop will exit and what parts of the suite will be executed each loop.
- Use them judiciously.
While Overview

```
while test1:
    statement_list_1
    if test2: break  # Exit loop now; skip else
    if test3: continue  # Go to top of loop now
    # more statements
else:
    statement_list_2  # If we didn’t hit a ‘break’

# ‘break’ or ‘continue’ lines can appear anywhere
```
Range and for Loop
Range Function

- The range function generates a sequence of integers

- range(5) => [0, 1, 2, 3, 4]
  - assumed to start at 0
  - goes up to, **but does not include**, the provided number argument.

- range(3,10) => [3, 4, 5, 6, 7, 8, 9]
  - first argument is the number to begin with
  - second argument is the end (exclusive)
Iterating Through the Sequence

for num in range(1, 5):
    print num

• range generates the sequence [1, 2, 3, 4]
• for loop assigns num each of the values in the sequence, one at a time in sequence
• prints each number (one number per line)
Range Function (Cont.)

• range(1, 12, 3) => [1, 4, 7, 10]
  – first argument is the number to begin with
  – second argument is the end (optional)
  – third argument is the span (optional)
Hailstone Example
Hailstone Sequence

- The Hailstone sequence is a simple algorithm applied to any positive integer.
- In general, by applying this algorithm to your starting number you generate a sequence of other positive numbers, ending at 1.
- Unproven whether every number ends in 1 (though strong evidence exists).
Algorithm

while the number does not equal one

• If the number is odd, multiply by 3 and add 1

• If the number is even, divide by 2

• Use the new number and reapply the algorithm
Even and Odd

Use the remainder operator

- if num % 2 == 0:  # even
- if num % 2 == 1:  # odd
Code Listing 2.26
Hailstone Sequence, Loop
while num > 1:  # stop when the sequence reaches 1
    if num%2 == 1:        # num is odd
        num = num*3 + 1
    else:            # num is even
        num = num/2
    print num","", # add num to sequence
    count +=1        # add to the count

else:
    print    # blank line for nicer output
    print "Sequence is ",count," numbers long"