

d. 118 *Ans: 'v'*
 e. 32 *Ans: ' ' (space)*

19. Use Appendix A to find the characters which correspond to these hexadecimal numbers:

a. 0x68 *Ans: 'h'*
 b. 0x37 *Ans: '7'*
 c. 0x44 *Ans: 'D'*
 d. 0x25 *Ans: '%'*

e. 0x2E *Ans: '.'*

20. Use Appendix A to find the hexadecimal equivalents of the following characters:

a. 'L' *Ans: 0x4C*
 b. *Ctrl+H* (backspace) *Ans: 0x08*
 c. '=' *Ans: 0x3D*
 d. 'U' *Ans: 0x55*
 e. '?' *Ans: 0x3F*

Lesson 18, While Statements

Important Concepts

Before starting this lesson, review the vocabulary words from Lessons #8, #9, #10 and #11. A verbal ungraded quiz in which the student fills in the blank is sufficient to emphasize that the vocabulary words from the previous lessons remain important throughout the course.

Conditional logic is the backbone of software programming. This topic is presented over a period of three lessons and two test modules to give students an opportunity to fully comprehend and assimilate this information.

New Definitions

break statement: A simple statement which, when placed within the while clause of a while statement, immediately stops processing of statements in the suite, and resumes processing of statements beyond the while statement without executing the suite of statements controlled by the else clause.

continue statement: A simple statement which, when placed within the while clause of a while statement, immediately stops processing of statements in the suite and returns control to the header of the while clause for the next evaluation of the boolean expression.

loop: In software, a portion of code which jumps back to a specific location to execute over and over.

refactor: A process by which source code is changed to improve maintenance or to prepare for new features without changing the capability of the program.

while statement: A compound statement which allows looping back to the top of the suite of controlled statements as long as the boolean expression in the header of the while clause evaluates to True.

Possible Roadblocks

Ensure that students work through all steps of the examples in this lesson until any uncertainty vanishes. All of the skills displayed in this lesson are foundation skills to be required throughout the remainder of this course.

Practice Exercises

A. List the two ways for a while loop to terminate. Which of these allows the else clause to execute?

Solution:

A while loop terminates when the boolean expression evaluates to False, or when a break statement is encountered. Only the boolean expression option allows the else clause to execute.

B. Consider the following while statement. Predict how many times 'did it' will print to the console. Check your prediction with the interactive Python console.

```
x = 3
while x > 0:
    print 'did it'
    x = x - 1
```

Solution:

Three times.

The first time the boolean expression evaluates, x contains 3. After a print statement, 1 is then subtracted leaving x with the value 2. After a second print statement, 1 is subtracted again leaving x at 1. The last time around x becomes zero after the third print statement. Upon the next evaluation of x, the boolean expression returns False and the loop terminates.

Exercises

1. Bob thinks that his source code is too messy to leave to a maintenance programmer to add features. He should **refactor** it.

2. A **while** statement is a compound statement which allows looping back to the top of the suite of controlled statements as long as the boolean expression in its header evaluates to True.

3. In software terms, a **loop** is a portion of code which jumps back to a specific location to execute over and over.

4. A **break** statement abnormally terminates a while

statement, and skips over the else clause, if one exists.

5. A **continue** statement causes an immediate re-evaluation of the boolean expression at the top of a while statement to see if processing the statement should resume.

6. Consider the following while statement. How many times will 'yep' print to the console? Confirm your answer with the Python console.

```
t = 4
while t < 6:
    print 'yep'
    t = t + 1
```

Solution:

Twice.

After the first 'yep' t is set to 5. After the second 'yep' t is set to 6. The boolean expression then evaluates to False and processing of the while statement terminates.

7. How many times would 'yep' print if the boolean expression in Exercise #6 was changed to `t <= 6`?

Solution:

Three times.

The boolean expression would evaluate to True one more time.

8. A **pass** statement is a do-nothing statement which acts as a placeholder where a statement is syntactically required.

9. **Code coverage** testing is a source code test method which involves testing each branch of code execution, as well as the boundary conditions between each branch.

10. An **elif** clause is used in the interior of an if statement to optionally execute a suite of statements based on a new boolean expression in the header of the clause.

11. An **else** clause identifies statements to be executed when the boolean expression of the associated if clause, and any preceding elif clauses, evaluate to False.

12. What will print on the console when the following code executes?

```
x = 3
if x <= 4:
    print 'Fee'
elif x < 8:
    print 'Fi'
elif x <= 12:
    print 'Foe'
else:
    print 'Fum'
```

Solution:

'Fee'

The if expression evaluates to True.

13. What prints to the console when the following code executes?

```
a = 4
b = 6.3
c = False

if a < 3:
    print 'pig'
elif not c:
    print 'wig'
elif c and (b > 5.1):
    print 'big'
else:
    print 'fig'
```

Solution:

'wig'

The if expression evaluates to False so the first suite is skipped. Next, the first elif expression evaluates to True so that suite executes.

14. A **hexadecimal** number is a number in the base-16 number system, commonly used by computer hardware, where individual place values are represented by the numerals 0 through 9 and A through F.

15. The **ASCII** character set is the 128 characters, numerals and console control codes which have been adopted as standard computer characters.

16. A **sequence** type is the class of Python types, each of which contains an ordered set of subordinate data elements.

17. Predict the result of the following string operations given the following variables:

```
s = 'pepperoni'
t = 'pizza'
```

a. <code>min(s)</code>	<i>Ans: 'e'</i>
b. <code>len(t)</code>	<i>Ans: 5</i>
c. <code>3 * t</code>	<i>Ans: 'pizzapizzapizza'</i>
d. <code>s[5]</code>	<i>Ans: 'r'</i>
e. <code>s[1:5]</code>	<i>Ans: 'eppe'</i>

18. Given the string `s` from Exercise #16, write a Python expression statement using a slice operation to return the string 'peron':

Solution:

`s[3:8]`

19. Use Appendix A to find the characters which correspond to these hexadecimal numbers:

a. <code>0x3E</code>	<i>Ans: '>'</i>
b. <code>0x32</code>	<i>Ans: '2'</i>
c. <code>0x46</code>	<i>Ans: 'F'</i>
d. <code>0x26</code>	<i>Ans: '&'</i>

e. `0x5D` *Ans: 'J'*

20. Use Appendix A to find the hexadecimal equivalents of the following characters:

a. `'P'` *Ans: 0x50*

b. `Ctrl+J` (linefeed) *Ans: 0x0A*

c. `'+'` *Ans: 0x2B*

d. `'u'` *Ans: 0x75*

e. `'''` *Ans: 0x22*

Lesson 19, Console Menu

Important Concepts

Before starting this lesson, review the vocabulary words from Lessons #8, #14, #16, #17 and #18. A verbal ungraded quiz in which the student fills in the blank is sufficient to emphasize that the vocabulary words from the previous lessons remain important throughout the course.

Requirements analysis is an essential element of professional software development. This lesson gives an overview of the subject. Numerous texts are available to students who wish to know more about the topic.

New Definitions

requirement: A description, usually written, of a specific task the software must perform.

requirements analysis: A software activity which involves deciding what the software should do, and capturing these ideas in a requirements document.

requirements document: A written list of requirements, usually in numbered outline form.

Possible Roadblocks

This lesson presents more code to be typed in a single lesson than any in this course so far. However, the code is straight-forward and is suitable for students to finish on their own as part of homework if they cannot finish within the lecture period.

Practice Exercises

A. Joe, a Python programmer, is talking to the sales manager about a program needed by the salesmen to delete wrong numbers from the customer list so that future calls won't be wasted. After he finishes taking notes on his notepad, what should he do? Afterward, who should he share this information with?

Solution:

He should write down his ideas as a requirements document and let the sales manager review it.

B. Predict the results of the following statements given the string `s` below. Test your predictions with the interactive Python console. Characters which look like a lower case L are a lower case L.

```
s = 'alligator'
```

i. `'l' == s[2]` *Ans: True*

ii. `s[1] == s[2]` *Ans: True*

iii. `'o' == s[4]` *Ans: False*

iv. `'gat' == s[4:7]` *Ans: True*

v. `'a' == s[0]` *Ans: True*

Exercises

1. A **requirement** is a description, usually written, of a specific task software must perform.

2. A **requirements document** is a list of requirements, usually in numbered outline form.

3. When we ask future users of our software what they need the software to do we are performing **requirements analysis**.

4. The programmers at your company can't agree about what exactly constitutes a requirements document. To avoid potential personal issues and arguments the software manager decided to not write a requirements document at all and just count on verbal understanding of what the program must do. Was this a wise decision? Why or why not?

Solution:

No, this was not a wise decision. Arguing about the contents of a requirements document indicates a better situation than not having a requirements document of any kind.

5. Predict the results of the following statements given the string `s` below. Test your predictions with the interactive Python console.

```
s = 'chicken'
```

i. `'i' == s[2]` *Ans: True*

ii. `s[0] == s[3]` *Ans: True*

iii. `'h' == s[4]` *Ans: False*

iv. `'ick' == s[2:4]` *Ans: False*

v. `'e' == s[5]` *Ans: True*

6. Sally updated the software to get ready for a major feature to be added. While doing so, she went ahead and implemented some of those features. Was she purely refactoring? Why or why not?