Python for Rookies

A First Course in Programming

Answers to Selected Exercises

Sarah Mount, James Shuttleworth and Russel Winder

THOMSON™

Australia • Canada • Mexico • Singapore • Spain • United Kingdom • United States

Contents

1

Getting Started

Self-Review Questions

Self-review 1.1 Why is the following the case?

```
>>> print 4 + 9
13
>>>
```

Because the Python system evaluates the expression 4 + 9 which means adding 4 to 9 giving the result 13 which is the printed out.

Self-review 1.2 Why is the following the case – rather than the result being 13?

```
>>> print "4 + 9"
4 + 9
>>>
```

Because "4+9" is a string and so it is printed as is, there is no expression to be evaluated.

Self-review 1.3 Why does the expression 2 + 3 * 4 result in the value 14 and not the value 24?

Because the * operator has higher precedence than the operator +. The 3 * 4 subexpression is evaluated first giving 12 and then the expression 2 + 12 is evaluated resulting in 14.

Self-review 1.4 What is a module and why do we have them?

A module is a collection of bits of program that provide a useful service.

Self-review 1.5 What is a library and why do we have them?

The answer "A place where books are held, so that they can be borrowed" is clearly correct but not the one wanted. The answer wanted is"A collection of bits of software that we can make use of in our programs."

Self-review 1.6 What does the statement `turtle.demo()` mean?

The statement causes the function `demo` from the Turtle module to be executed.

Self-review 1.7 What is an algorithm?

It is a procedure/process for achieving a desired goal.

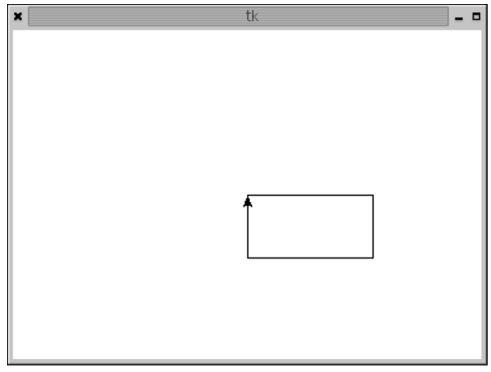
Self-review 1.8 Why are algorithms important in programming?

Programming is all about implementing algorithms in software so algorithms are critical and core to programming.

Self-review 1.9 What shape does the following program draw?

```
turtle.forward ( 100 )
turtle.right ( 90 )
turtle.forward ( 50 )
turtle.right ( 90 )
turtle.forward ( 100 )
turtle.right ( 90 )
turtle.forward ( 50 )
```

A rectangle 100 unit wide and 50 units deep:



Self-review 1.10 The Turtle module provides a function to draw circles. If this function was not available, what algorithm would you use to draw a circle?

Draw a large number of very short straight lines changing the direction of travel by 360° divided by the number of lines. So if the number is 3 we have a triangle. If the number is 4 we have a square. If the number is 5 we have a pentagon, with 6 a hexagon. If we have 100 sides then for a small 'circle' it may not be noticeable that the lines are straight.

Self-review 1.11 What does the function turtle.goto do?

Hint: Use the Python help system to help you!

```
Help on function goto in turtle:

turtle.goto = goto(*args)
  Go to the given point.

  If the pen is down, then a line will be drawn. The turtle's
  orientation does not change.

  Two input formats are accepted:

    goto(x, y)
    go to point (x, y)
```

```
  goto((x, y))
  go to point (x, y)

Example:
>>> turtle.position()
[0.0, 0.0]
>>> turtle.goto(50, -45)
>>> turtle.position()
[50.0, -45.0]
```

Self-review 1.12 What is stdin?

Self-review 1.13 What is 'script mode'?

Programming Exercises

Exercise 1.1 Estimate the size of your computer screen by importing the Turtle module and causing the turtle to move 1 pixel (though you may want to try 10 pixels!).

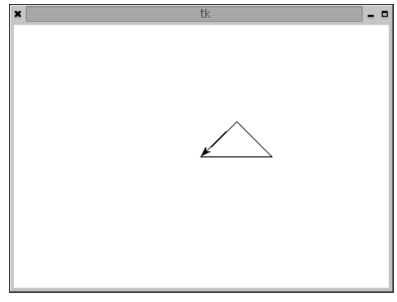
Using the program:

```
import turtle

turtle.forward ( 1279 )
turtle.right ( 90 )
turtle.forward ( 512 )

raw_input ( 'Press return to terminate the program: ' )
```

and the window manager to resize and reposition the drawing window, we saw that the screen width was 1280. By making the window wider than the screen and then locating one end of the horizontal line at one side of the screen, we saw that the down line was right on the other edge of the screen. This trick doesn't work so easily for vertical measurement. Hence we drew half what we thought was the screen height and saw whether it was half the height. It was so we estimate the height to be 1024.

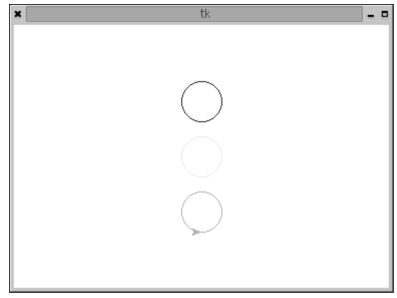
Exercise 1.2 Experiment with the 'square spiral' program, varying the lengths of various lines. Which values make spirals and which do not? Which values make the nicest-looking spiral?

Exercise 1.3 Write a program to draw a right-angled triangle looking something like:



```
import turtle

turtle.forward ( 70 )
turtle.left ( 135 )
turtle.forward ( 50 )
turtle.left ( 90 )
turtle.forward ( 50 )

# Prompt to stop the program from terminating and hence the display from disappearing.

raw_input ( 'Press return to terminate the program: ' )
```

Exercise 1.4 Write a program to draw a red circle, then a yellow circle underneath it and a green circle underneath that. The result should look something like:

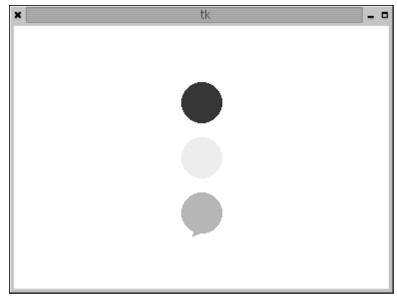


It is true that in this monochrome rendition, the colors just appears in various shades of grey, but the colors will work on your computer – unless you have a monochrome screen!

Hint: We used the function turtle.goto as well as the functions turtle.up, turtle.down, turtle.color and turtle.circle in our solution.

```
import turtle

turtle.up ( )
turtle.goto ( 0 , 35 )
turtle.down ( )
turtle.color ( 'red' )
turtle.circle ( 20 )
turtle.up ( )
turtle.goto ( 0 , -20 )
turtle.down ( )
turtle.color ( 'yellow' )
turtle.circle ( 20 )
turtle.up ( )
turtle.goto ( 0 , -75 )
turtle.down ( )
turtle.color ( 'green' )
turtle.circle ( 20 )

raw_input ( 'Press return to terminate the program: ' )
```

Exercise 1.5 Amend your program from the previous question so that the circles are filled and hence the result looks something like:

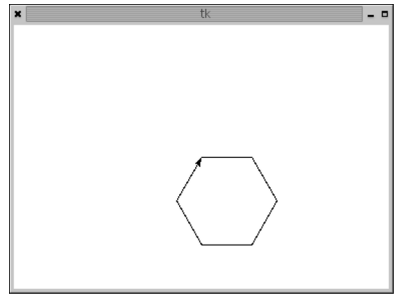


These circles are definitely filled with the right color when we run the program on our computers even though they just look greyish in this book.

Hint: We used the function turtle.fill in our solution.

```
import turtle

turtle.up ( )
turtle.goto ( 0 , 35 )
turtle.down ( )
turtle.color ( 'red' )
turtle.fill ( 1 )
turtle.circle ( 20 )
turtle.fill ( 0 )
turtle.up ( )
turtle.goto ( 0 , -20 )
turtle.down ( )
turtle.color ( 'yellow' )
turtle.fill ( 1 )
turtle.circle ( 20 )
turtle.fill ( 0 )
turtle.up ( )
turtle.goto ( 0 , -75 )
turtle.down ( )
turtle.color ( 'green' )
turtle.fill ( 1 )
turtle.circle ( 20 )
turtle.fill ( 0 )

raw_input ( 'Press return to terminate the program: ' )
```

Exercise 1.6 Write a program to draw a regular hexagon. The result should look something like:



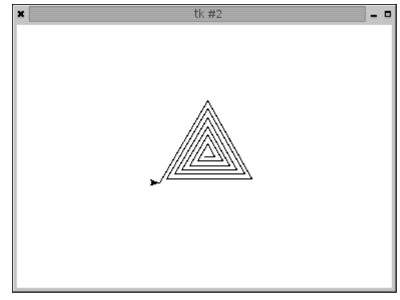
Hint: Regular hexagons have six sides, of equal length, which meet at an internal angle of 120°.

```
import turtle

turtle.forward ( 50 )
turtle.right ( 60 )
turtle.forward ( 50 )
turtle.right ( 60 )
turtle.forward ( 50 )
turtle.right ( 60 )
turtle.forward ( 50 )
turtle.right ( 60 )
turtle.forward ( 50 )
turtle.right ( 60 )
turtle.forward ( 50 )

raw_input ( 'Press return to terminate the program: ' )
```

Exercise 1.7 Write a program to draw the first letter of your name.

Challenges

Challenge 1.1 Write a program to create a triangular spiral, as in:

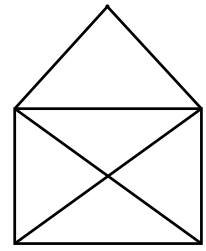


Hint: This is easiest if you work with equilateral triangles – it gets very complex otherwise.

```
import turtle

turtle.forward ( 10 )
turtle.left ( 120 )
turtle.forward ( 15 )
turtle.left ( 120 )
turtle.forward ( 20 )
turtle.left ( 120 )
turtle.forward ( 25 )
turtle.left ( 120 )
turtle.forward ( 30 )
turtle.left ( 120 )
turtle.forward ( 35 )
turtle.left ( 120 )
turtle.forward ( 40 )
turtle.left ( 120 )
turtle.forward ( 45 )
```

```
turtle.left ( 120 )
turtle.forward ( 50 )
turtle.left ( 120 )
turtle.forward ( 55 )
turtle.left ( 120 )
turtle.forward ( 60 )
turtle.left ( 120 )
turtle.forward ( 65 )
turtle.left ( 120 )
turtle.forward ( 70 )
turtle.left ( 120 )
turtle.forward ( 75 )
turtle.left ( 120 )
turtle.forward ( 80 )
turtle.left ( 120 )
turtle.forward ( 85 )
turtle.left ( 120 )
turtle.forward ( 90 )
turtle.left ( 120 )
turtle.forward ( 95 )
turtle.left ( 120 )

raw_input ( 'Press return to terminate the program: ' )
```

Challenge 1.2 Write a program to draw this 'envelope' figure:



Hint: You may want to start by doing some trigonometry to sort out all the angles and lengths. Pythagoras' Theorem will almost certainly come in useful: Pythagoras' Theorem states that for a right-angled triangle:

$$hypotenuse = \sqrt{x^2 + y^2}$$

where hypotenuse *is the longest side of the triangle and* x *and* y *are the lengths of the other two sides.*

Hint: To find the square root of a number in Python, you need to import the module ***math*** *and call the function* ***math.sqrt****: to find the square root of 2, you import the* ***math*** *module, then write* ***math.sqrt(2)****.*

```
#! /usr/bin/env python

import turtle
import math
```

```
# Assume the house body is a square and the roof is a right-angled triangle (half a square).
We
# can then get the various diagonals using Pythogoras' Theorem with all angles being multiples
# of 45.

# It seems almost totally unreasonable not to use variables to solve this problem.

sideLength = 40
halfDiagonalLength = 0.5 * math.sqrt ( 2 * ( sideLength ** 2 ) )

# Create the right orientation.

turtle.left ( 90 )

# Draw the house.

turtle.forward ( sideLength )
turtle.right ( 135 )
turtle.forward ( halfDiagonalLength )
turtle.right ( 90 )
turtle.forward ( halfDiagonalLength )
turtle.left ( 135 )
turtle.forward ( sideLength )
turtle.left ( 135 )
turtle.forward ( halfDiagonalLength )
turtle.right ( 90 )
turtle.forward ( halfDiagonalLength )
turtle.left ( 135 )
turtle.forward ( sideLength )
turtle.right ( 135 )
turtle.forward ( halfDiagonalLength )
turtle.right ( 90 )
turtle.forward ( halfDiagonalLength )
turtle.right ( 45 )
turtle.forward ( sideLength )

# Prompt to stop the program from terminating and hence the display from disappearing.

raw_input ( 'Press return to terminate the program: ' )
```

Challenge 1.3 Write a program to draw the following figure:

2

The Fundamentals

Self-Review Questions

Self-review 2.1 What are the types of the following literals? 1, 1., 1.0, ”1”, ”1.”, ”1.0”, ’1’, ’1.’, 100000000000000000, 100000000000000000., 100000000000000000.0.

int, float, float, str, str, str, str, str, long, float, float.

Self-review 2.2 Explain the difference between the following two statements:

```
print 12
print "12"
```

The first print the integer literal 12, the second prints the string comprising the character 1 followed by the character 2.

Self-review 2.3 Highlight the literals in the following program:

```
a=3
b='1'
c=a-2
d=a-c
e="dog"
f=', went to mow a meadow.'
g='man'
print a,g,"and his",e,f,a,g,",",d,g,",",c,g,"and his",e,f
```

The literals are: 3, ’1’, 2, "dog", ’, went to mow a meadow.’, ’man’, "and his", ",", "and his".

Self-review 2.4 What is the output of the above program?

This is best answered by actually executing the code, even though the question was really about ‘mental execution’, i.e. thinking through what the Python system would do when executing the code.

```
|> python meadowSong.py
3 man and his dog , went to mow a meadow. 3 man , 2 man , 1 man and his dog , went to mow a mea
|>
```

The important issue here is that ‘mental execution’ is an important skill so that you know when an error has occurred and can debug the error.

Self-review 2.5 What are the types of the variables a, b, c, d, e, f, g in the above program?

int, str, int, int, str, str, str

Self-review 2.6 Is there a difference between the *type* of the expressions "python" and 'python'?

Self-review 2.7 What is the result of 4+4/2+2?

8. the 4/2 is evaluated first then the left addition 4+2, then the right addition 6+2.

Self-review 2.8 Add brackets as needed to the above expression to make the answer:

a. 2

b. 5

c. 6

1. (4 + 4) / (2 + 2)
2. 4 + (4 / (2 + 2))
3. (4 + 4) / 2 + 2

Self-review 2.9 What is iteration?

It is the repeating of some action or sequence of actions.

Self-review 2.10 What is the difference between the expressions c = 299792458 and c = 2.99792458 * 10 ** 8?

In the first c is of type int, in the second, c is of type float. The value represented is the same.

Self-review 2.11 What does this expression n < 140 mean? I.e. what test is undertaken and what is the value and type of the result?

It tests whether the value of the variable n is less than the literal 140 and delivers **True** if it is and **False** if it is not. The result of the expression if of type bool.

Self-review 2.12 What is the value of the variable val after the executing the expression val = 1 / 2? What is the type of val?

Self-review 2.13 What is the value of the variable val after the executing the expression val = 1 / 2.0? What is the type of val?

Self-review 2.14 What are the possible values contained in a variable of type bool?

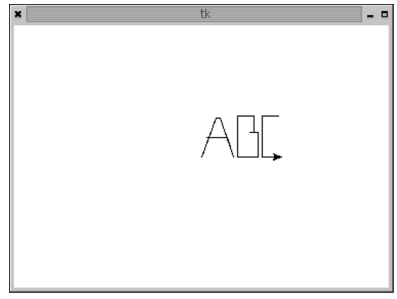
Programming Exercises

Exercise 2.1 The program:

```
import turtle

scale = 4

## Letter A
turtle.down ( )
# Point upwards to begin
turtle.left ( turtle.heading ( ) + 90 )
turtle.right ( 20 )
turtle.forward ( 10 * scale )
turtle.right ( 70 )
turtle.forward ( 1 * scale )
turtle.right ( 70 )
turtle.forward ( 10 * scale )
turtle.backward ( 5 * scale )
turtle.right ( 90 + 20 )
turtle.forward ( 5 * scale )
#Move to right of letter and over 1 * scale
turtle.up ( )
turtle.backward ( 5 * scale )
turtle.left ( 110 )
turtle.forward ( 5 * scale )
turtle.left ( 70 )
turtle.forward ( 1 * scale )

## Letter B
turtle.down ( )
# Point upwards to begin
turtle.left ( turtle.heading ( ) + 90 )
turtle.forward ( 10 * scale )
turtle.right ( 90 )
turtle.forward ( 4 * scale )
turtle.right ( 90 )
turtle.forward ( 4 * scale )
turtle.left ( 90 )
turtle.backward ( 1 * scale )
turtle.forward ( 2 * scale )
turtle.right ( 90 )
turtle.forward ( 6 * scale )
turtle.right ( 90 )
turtle.forward ( 5 * scale )
# Move to right of letter
turtle.up ( )
turtle.right ( 180 )
turtle.forward ( 6 * scale )

## Letter C
turtle.down ( )
# Point upwards to begin
turtle.left ( turtle.heading ( ) + 90 )
turtle.forward ( 10 * scale )
turtle.right ( 90 )
turtle.forward ( 4 * scale )
turtle.backward ( 4 * scale )
turtle.left ( 90 )
```

```
turtle.backward ( 10 * scale )
turtle.right ( 90 )
turtle.forward ( 4 * scale )
# Move to right of letter
turtle.up ( )
turtle.forward ( 1 * scale )

# Pause
raw_input ( "Press any key to end." )
```

causes the following when executed:



The code for writing each individual letter is fairly easy to spot, because of the comments. There's a much easier way to divide this code up though: functions. The exercise then is to:

1. Modify the code so that each letter is drawn by a function: `drawA ( )` for the letter A, for example.
2. Add the letters "D" and "E" as functions.
3. Amend the code to display the word "DECADE" rather than "ABCDE".

Exercise 2.2 Rewrite your answer to Exercise 1.4 (three colored circles one above the other) using a function.

Hint: The function should probably have three parameters.

```
import turtle

def drawAtWithColor ( x , y , c ) :
    turtle.up ( )
    turtle.goto ( x , y )
    turtle.down ( )
    turtle.color ( c )
    turtle.circle ( 20 )

drawAtWithColor ( 0 , 35 , 'red' )
drawAtWithColor ( 0 , -20 , 'yellow' )
drawAtWithColor ( 0 , -75 , 'green' )

raw_input ( 'Press return to terminate the program: ' )
```

Exercise 2.3 Rewrite your answer to Exercise 1.5 (three colored, filled circles one above the other) using a function.

Hint: The answer should be a trivial extension to the answer of the previous question.

```
import turtle

def drawAtWithColor ( x , y , c ) :
    turtle.up ( )
    turtle.goto ( x , y )
    turtle.down ( )
    turtle.color ( c )
    turtle.fill ( 1 )
    turtle.circle ( 20 )
    turtle.fill ( 0 )

drawAtWithColor ( 0 , 35 , 'red' )
drawAtWithColor ( 0 , -20 , 'yellow' )
drawAtWithColor ( 0 , -75 , 'green' )

raw_input ( 'Press return to terminate the program: ' )
```

Exercise 2.4 Rewrite your answer to question Exercise 1.6 (drawing a hexagon) using iteration.

```
import turtle

for i in range ( 6 ) :
    turtle.forward ( 50 )
    turtle.right ( 60 )

raw_input ( 'Press return to terminate the program: ' )
```

Exercise 2.5 Extend your answer to the previous question so that the number of sides to draw is a parameter – this is a programming solution for Self-review 1.10. How many sides do you need before it looks like a circle?

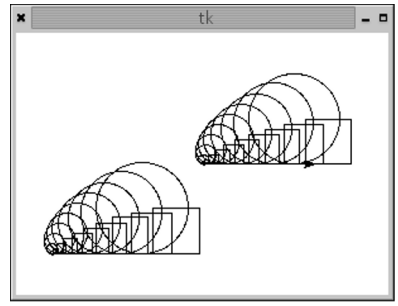
```
import turtle

circumference = 300

sides = input ( 'How many sides: ' )

length = circumference / sides
angle = 360 / sides

for i in range ( sides ) :
    turtle.forward ( length )
    turtle.right ( angle )

raw_input ( 'Press return to terminate the program: ' )
```

Experiment indicates that on a 1280×1024, 90dpi screen anything over about 20 sides cannot easily be distinguished from an actual circle.

Exercise 2.6 Rewrite your answer to Challenge 1.1 using a function.

```
import turtle

length = 10

def drawPair ( ) :
    global length
    turtle.forward ( length )
    length += 5
    turtle.left ( 120 )
    turtle.forward ( length )
    length += 5
    turtle.left ( 120 )

for i in range ( 10 ) :
    drawPair ( )

raw_input ( 'Press return to terminate the program: ' )
```

Exercise 2.7 Write a program to draw the following image:



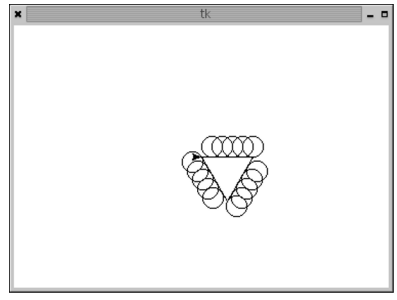

```
import turtle

def drawSquare ( x ) :
    for i in range ( 4 ) :
        turtle.forward ( x )
        turtle.left ( 90 )

def setPosition ( x , y ) :
    turtle.up ( )
    turtle.goto ( x , y )
    turtle.down ( )

def drawSequence ( ) :
    for i in range ( 10 ) :
        turtle.circle ( i * 4 )
        turtle.up ( )
        turtle.forward ( i )
        turtle.down ( )
        drawSquare ( i * 4 )
        turtle.up ( )
        turtle.forward ( i )
        turtle.down ( )
```

```
setPosition ( -120 , -70 )
drawSequence ( )
setPosition ( 0 , 0 )
drawSequence ( )

raw_input ( 'Press any key to terminate: ' )
```

Exercise 2.8 When the code:

```
import turtle

sides = 3
length = 5

for i in range ( sides ) :
    for j in range ( length ) :
        turtle.forward ( 10 )
        turtle.circle ( 10 )
    turtle.right ( 120 )

raw_input ( 'Press any key to end' )
```

is executed the result is:



i.e. it draws a triangle with circles along its edges. The *intention* was that modifying the variable sides would allow you to draw a square (by setting it to 4), a pentagon (setting it to 5) and so on. This doesn't currently work, however. The programmer clearly became distracted and left the program half finished. Your job is to make the code work as intended.

Hint: Remember that for any regular polygon, the external angles always add up to 360. If they didn't, it would either not be closed or it would overlap, or it would not be regular!

Challenges

Challenge 2.1 Go back and make sure you finish the programming exercise on creating polygons surrounded by circles. If necessary modify the program so that the drawing is undertaken by a function with parameters for side length and number of sides. Then add a further parameter to your function that allows the size of the circles to be

adjusted. Finally, write a program that asks the user to enter values for side length, number of sides and size of circles – this is now a general purpose regular-polygon-surrounded-by-circles generator. Not entirely useful per se, but fun!

Challenge 2.2 Can you spot a way of making money given the exchange rates in the last version of the currency conversion program? The trick is to write a program to show how much you could lose or make by exchanging between these currencies.

What stops people (as opposed to banks) exploiting this for profit?

(If you are a bank, the opportunities here are called *currency arbitrage*, but it is a high-risk activity.)

3

Controlling the Flow

Self-Review Questions

Self-review 3.1 The following expression is true when age is 21 and height is 120, or age is 21 and height is 140:

```
( age == 21 and height == 120 ) or ( age == 21 and height == 140 )
```

Can you write the expression with fewer terms?

Yes.

Self-review 3.2 Simplify the following Boolean expressions:

1. a **or** (b **and** c) **not** a
2. b **and** c **or False**
3. a **and** b **or** c **or** (b **and** a)
4. a == **True or** b == **False**

Self-review 3.3 'Nand' is an operator that is false only when both of its operands are true. Python doesn't have a nand operator, so how can we write an expression that is equivalent toA nand B?

Self-review 3.4 Under what conditions would the following code sequences print 'B'?

1.
```
if thing1 > 10 :
   print 'A'
else:
   print 'B'
```

2.
```
if thing1 > 10 :
   print 'A'
elif thing1 > 200 :
   print 'B'
```

3.
```
if thing1 > 10 :
   print 'A'
if thing1 > 200 :
   print 'B'
```

4.
```
if thing1 > 10 and thing1 < 10 :
  print 'A'
else:
  print 'B'
```

Self-review 3.5 How many times will Python execute the code inside the following while loops? You should answer the question *without* using the interpreter! Justify your answers.

1.
```
i = 0
while i < 0 and i > 2 :
    print "still going ..."
    i = i+1
```

2.
```
i = 25
while i < 100 :
    print "still going ..."
    i = i - 1
```

3.
```
i = 1
while i < 10000 and i > 0 and 1 :
    print "still going ..."
    i = 2 * i
```

4.
```
i = 1
while i < 10000 and i > 0 and 0 :
    print "still going ..."
    i = 2 * i
```

5.
```
while i < 2048 and i > 0 :
    print "still going ..."
    i = 2 * i
```

6.
```
i = 1
while i < 2048 and i > 0 :
    print "still going ..."
    i = 2 * i
```

7.
```
for i in [ ] :
    print "foobar!"
```

Self-review 3.6 What extra assert statement would be worth adding to the set of assertions that test the contains function on page ?? and page ???

To ensure there is no problem if the string is the last letters.

```
assert contains ( 'hellohello' , 'ohello' ) == True
```

To ensure there is no problem if the sub-string is a single letters.

```
assert contains ( 'hellohello' , 'h' ) == True
```

Self-review 3.7 Explain the difference between the iterative version of factorial on page ?? and the following:

```
def factorial ( x ) :
    if x < 0 :
        raise ValueError , 'Factorial not applicable to negative values.'
    elif x == 0 :
        return 1
    else :
        value = 1
        for i in range ( x , 1 , -1 ) :
            value *= i
        return value
```

The earlier one counts up, the above counts down.

Self-review 3.8 Compare the first power function (in Section ??) to the more efficient one in Section ??. How many multiplications does each implementation make to evaluate power (3 , 5)? Make sure you show your reasoning.

Self-review 3.9 Referring to the recursive function for generating numbers in the Fibonacci Sequence (page ??), how many calls to fibonacci are made to evaluate fibonacci (5)? Can you write down a general rule for how many calls must be made to evaluate fibonacci (n)?

Self-review 3.10 What lists do the following expressions evaluate to?

1. range (1 ,100)
2. range (1 , 50 , 2)
3. range (1 , 25 , 3)
4. range (10 , -10 , -2)

Self-review 3.11 The following code contains the names a, b and c. What is the scope of each of the them?

```
a = 10
for b in range ( 1 , 10 ) :
    c = b + 10
    print c
```

Self-review 3.12 What does the following code do? How might it be used in a larger program?

```
print 'Menu:'
print '1. Play game'
print '2. See high score table'
print '3. Exit'
i = -1
while i < 1 or i > 3:
    j = raw_input ( 'Enter choice: ' )
    try :
        i = int ( j )
    except :
        continue
```

Self-review 3.13 For the following recursive function, label the base cases and recursive cases. What do these functions do?

```
def m ( a , b ) :
  def mm ( a , b , c ) :
    if a == 0 : return c
    else : return mm ( a - 1 , b , c + b )
  return mm ( a , b , 0 )

def d ( a , b ) :
  def dd ( a , b , c ) :
    if a < b : return c
    else : return dd ( a - b , b , c + 1 )
  return dd ( a , b , 0 )
```

Hint: You are almost certainly already familiar with these algorithms, just not in this form! You might want to try working out what these functions would return for some simple input values.

Self-review 3.14 We are to write some assert statements of the form:

```
assert ( m ( ? , ? ) == ? )
```

and

```
assert ( d ( ? , ? ) == ? )
```

to create tests for the two functions from Self-review 3.13. What assert statements would it be useful to put into a test? What parameter values might cause problems for m and d?

Self-review 3.15 What do the Python keywords **break** and **continue** do?

Self-review 3.16 What does the following print out?

```
for i in range ( 1 , 10 ) :
    for j in range ( 1 , 10 ) :
        print i * j ,
    print
```

Self-review 3.17 What do they keywords **raise**, **try** and **except** mean?

Programming Exercises

Exercise 3.1 Implement a recursive Python function that returns the sum of the first *n* integers.

Exercise 3.2 Implement an iterative Python function that returns the sum of the first *n* integers.

Exercise 3.3 Implement an iterative Python function to generate numbers in the Fibonacci Sequence.

Exercise 3.4 While and for loops are equivalent: whatever you can do with one you can do with the other. In the following programs, convert the following while loops into for loops and the for loops into while loops. Of course, your new loops should give the same results as the old ones!

1.
```
for i in range ( 1 ,100 ) :
    if i % 3 == 2 :
        print i , "mod" , 3 , "= 2"
```

2.
```
for i in range(10):
    for j in range(i):
        print '*',
    print ''
```

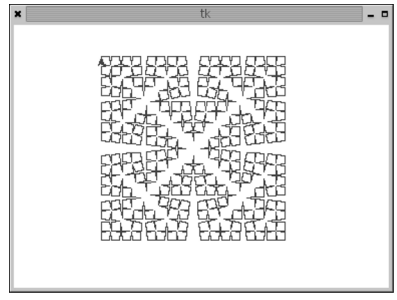
3.
```
i = 0
while i < 100 :
    if i % 2 == 0 :
        print i , "is even"
    else :
        print i , "is odd"
    i = i + 1
```

4.
```
char = ""
print "Press Tab Enter to stop and Enter to keep going ..."
iteration = 0
while not char == "\t" and not iteration > 99:
    print "Keep going?"
    char = raw_input()
    iteration += 1
```

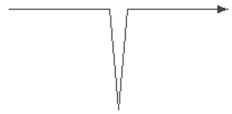
Exercise 3.5 Using books, the Web, or any other resource, find out what a logic gate is, and, in particular, what a half adder is. An output of the work should be a table showing the inputs and outputs for a half adder.

Implement a half adder as a few lines of Python code. Write a test program that shows that your implementation works as expected. You should be able to test that your code works for *all* possible input values.

Exercise 3.6 The following shows a Cesàro Fractal (also known as a Torn Square fractal). Write a program to draw this figure.



Hint: The Cesàro Fractal is based on a square. Each line, if it is longer than the minimum line length, is a line with an extra triangular section:



The triangular section is an isosceles triangle and is always on the 'inside' of the square.

```
#! /usr/bin/env python
# -*- coding: UTF-8 -*-

# Even in Python 2.4.3, you still cannot have Unicode characters in function names :-(

import turtle

def initialize ( color = 'blue' , smallest = 1.0 ) :
    '''Prepare to draw CesÃ¤ro Fractal'''
    turtle.clear ( )
    turtle.up ( )
    turtle.goto ( -100.0 , 100.0 )
    turtle.setheading ( 0 )
    turtle.color ( color )
    turtle.down ( )
    global smallestLineLength
    smallestLineLength = smallest

def cesaroLine ( overallLength ) :
    '''Draw a line of the CesÃ¤ro Fractal of a given length. The global variable smallestLineLength
    assumed to be set.'''
    halfSideLength = ( 0.9 * overallLength ) / 2.0
    if halfSideLength <= smallestLineLength :
        moveFunction = turtle.forward
    else :
        moveFunction = cesaroLine
    a = 85
    b = 180 - 2 * a
    moveFunction ( halfSideLength )
    turtle.right ( a )
    moveFunction ( halfSideLength )
    turtle.left ( 180 - b )
    moveFunction ( halfSideLength )
    turtle.right ( a )
    moveFunction ( halfSideLength )

def cesaroFractal ( overall , smallest = 1.0 , color = 'blue' ) :
    """Generate a CesÃ¤ro Fractal (a Torn Square Fractal) which is a square of lines with a 'triang
    initialize ( color , smallest )
    cesaroLine ( overall )
    turtle.right ( 90 )
    cesaroLine ( overall )
    turtle.right ( 90 )
    cesaroLine ( overall )
    turtle.right ( 90 )
```

```
    cesaroLine ( overall )

if __name__ == '__main__' :
    #initialize ( 'brown' , 100 ) ; cesaroLine ( 200 )
    cesaroFractal ( 200.0 , 10.0 , 'brown' )
    raw_input ( 'Press any key to terminate' )
```

Exercise 3.7 Self-review 3.13 contained recursive implementations of functions `m` and `d`. Write equivalent functions that use iteration rather than recursion to produce the same results

Exercise 3.8 Write a function that capitalizes all the vowels in a string.

Hint: We have generally used indexing and the ***range*** *function when using for loops. We can also use for loops for iterating over the characters in a string. For example:*

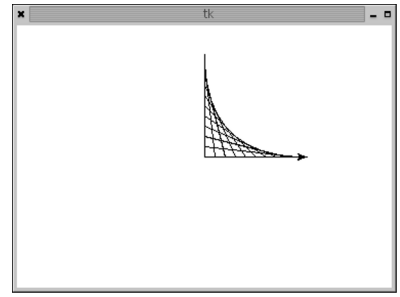
```
for char in 'foobar' :
    print char
```

Hint: Look at the documentation on strings: it has functions one or more of which are useful for this exercise. You might try using Python in interactive mode, importing the string module, and asking for help:

```
>>> import string
>>> help(string)
. . .
```

or look up the string module in the Python documentation.

Exercise 3.9 Use the Turtle module and nested loops to draw the following shape:




```
import turtle

turtle.goto ( ( 0 , 100 ) )
turtle.goto ( ( 0 , 0 ) )
turtle.goto ( ( 100 , 0 ) )
turtle.goto ( ( 0 , 0 ) )

for i in range ( 100 , -10 , -10 ) :
    turtle.up ( )
    turtle.goto ( ( 0 , i ) )
    turtle.down ( )
```

```
        turtle.goto ( ( 100 - i , 0 ) )

raw_input ( 'Press Enter to close.' )
```

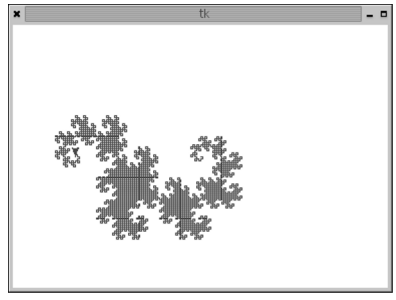
Exercise 3.10 Linux, Mac OS X, Solaris and other Unix-like systems have a command `cal` that, when run without a parameter, prints out the calendar for the current month. For example:

```
|> cal
      May 2007
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

|>
```

Use loops and nested loops to print out a similar table for this month. Try to make sure that your columns line up correctly, like the example above!

Challenges

Challenge 3.1 The following shows a Dragon Curve.



Find an algorithm for drawing a Dragon Curve and write a program to draw a figure such as the one shown.

Hint: The starting point for a Dragon Curve is a straight line, but you always have to deal with two lines at a time. So this is not a simple recursion – more than one recursive function is needed.

```
#! /usr/bin/env python
# -*- coding: UTF-8 -*-

# Even in Python 2.4.3, you still cannot have Unicode characters in function names :-(

import turtle

def initialize ( color = 'blue' , smallest = 1.0 ) :
```

```
    '''Prepare to draw a Dragon Curve'''
    turtle.clear ( )
    turtle.up ( )
    turtle.setheading ( 0 )
    turtle.color ( color )
    turtle.down ( )
    global smallestLineLength
    smallestLineLength = smallest

def dragonFirst ( overallLength ) :
    '''Draw the first part of a pair of lines in the Dragon Curve.'''
    if overallLength <= smallestLineLength :
        turtle.forward ( overallLength )
    else :
        overallLength /= 2.0
        dragonFirst ( overallLength )
        turtle.right ( 90 )
        dragonSecond ( overallLength )

def dragonSecond ( overallLength ) :
    '''Draw the second part of a pair of lines in the Dragon Curve.'''
    if overallLength <= smallestLineLength :
        turtle.forward ( overallLength )
    else :
        overallLength /= 2.0
        dragonFirst ( overallLength )
        turtle.left ( 90 )
        dragonSecond ( overallLength )

def dragonCurve ( overallLength , smallestLineLength , color ) :
    initialize ( color , smallestLineLength )
    dragonFirst ( overallLength )

if __name__ == '__main__' :
    #dragonCurve ( 2000.0 , 10.0 , 'purple' )
    #dragonCurve ( 4000.0 , 5.0 , 'purple' )
    dragonCurve ( 8000.0 , 2.5 , 'purple' )
    raw_input ( 'Press any key to terminate' )
```

Challenge 3.2 One application area in which iteration and recursion are particularly useful is user interfaces – programs that directly interact with people. These include websites, games, spreadsheets, and so on. For this challenge, you will create a text-based user interface to the Python turtle. The idea behind this is to let people use the turtle without having to learn Python first.

As a start, here is some code that allows you to move the turtle around using the keys a, s, w, and d on the keyboard:

```
import turtle

def turtle_interface ( ) :
    """Allow the user to control the turtle via the keyboard."""
    steps = 10
    angle = 5
    while True :
```

```
        i = raw_input ( )
        if i == 'w' : turtle.forward ( steps )
        elif i == 's' : turtle.backward ( steps )
        elif i == 'a' : turtle.left ( angle )
        elif i == 'd' : turtle.right ( angle )
        elif i == 'q' : break
        else : continue

print 'Control the turtle!'
turtle_interface()
```

Improve this code by adding:

1. A function to print out a menu before the user starts. This should explain which keys are used to control the turtle. (Make sure that you keep the key map up to date as you add new functionality to the program.)
2. User options to perform the following functions:
   (a) Clear the screen.
   (b) Lift the 'pen' of the turtle off the page.
   (c) Put the pen back down on the page.
   (d) Change the width of the line that the turtle draws.
   (e) Draw a circle. Better still, let the user decide how big a radius the circle should have.
   (f) Change the color of the turtle. Allow the user to choose from a small number of colors (say, red, blue, green and black).

   (You may well need to read through the documentation for the Turtle module to implement these.)
3. Undo capability. You will need to store the last key the user pressed and write some code (maybe in a separate function) to map keys onto their opposite actions. This might not be possible for some user options (such as drawing a circle) but others should be straightforward. For example, if the user has previously moved forward by 10 units, the opposite action is to move backwards 10 units.

4

Structuring State

Self-Review Questions

Self-review 4.1 What are the values of the variables a, b, c and d after the following statements have been executed?

```
a = 1
b = 2
c = a + b
d = a + c
```

a will be 1, b will be 2, c 3 and d 4.

Self-review 4.2 How many elements are in the dictionary someData after the following code has been executed?

```
someData = { }
someData['cheese'] = 'dairy'
someData['Cheese'] = 'dairy'
```

Self-review 4.3 Given the following assignment:

```
items = ( ( 3 , 2 ) , ( 5 , 7 ) , ( 1 , 9 ) , 0 , ( 1 ) )
```

without writing a Python program or using the Python interpreter, answer the following:

1. What value is returned by len (items)?
2. What is the value of items[1]?
3. **print** items[2][0] prints 1 but **print** items[3][0] causes a TypeError. Why is this given that items has more than 3 elements?
4. For items[x][y], what values of x and y cause the expression to evaluating to 9?
5. It might at first appear that the type of items[0] and the type of items[4] are the same.
   (a) What are the types of these two expressions?
   (b) Why is the type of items[4] not tuple?
   (c) If it was intended that items[4] be a tuple, how should the assignment be altered, without adding any new values?

Self-review 4.4 What is the output of the following code?

```
a = 1
def f ( ) :
    a = 10
print a
```

The code will print 1 to the console.

Self-review 4.5 Explain why the code in the previous question did *not* display 10.

Because the scope of the line `a = 10` is the method `f`, which is never called.

Self-review 4.6 Referring to the coffee/cola vending machine state transition diagram (page ??), what happens if the consumer puts three tokens into the machine without pressing any buttons?

Self-review 4.7 Which of the following are types mutable and which are immutable: `int`, `list`, `float`, `tuple`, `str`.

All the types listed are mutable except for `tuple` and `str`.

Self-review 4.8 What happens when the following statement is executed as the first statement of a program?

```
x , y , z = 1 , 2 , x ** 3
```

There is an error – the exception `NameError` is raised. All the expressions on the right-hand side are evaluated before any assignments are done so `x` is being used before it is assigned.

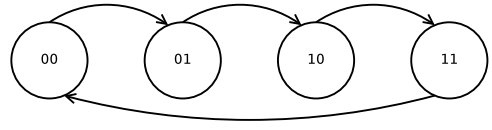
Self-review 4.9 For each of the following lines, using substitution, hand-evaluate the following sequence:

```
x , y , z = 1 , 2 , 3
z , y , x = x , z , y
x , y , z = ( z + 1 ) , ( x - 2 ) , ( y * 3 )
y , z , x = ( y / x ) , ( x * y ) , ( z ** x )
```

Self-review 4.10 Draw the state space of the following devices:

1. An MP3 or Ogg Vorbis player.
2. A microwave oven.

Self-review 4.11 Draw a state transition diagram for a two-bit binary counter. A binary counter is a digital circuit that has a clock input and an output that gives the number of clock cycles that have been counted. A two-bit binary counter only has two output connections, which can have the value of either 1 or 0. This means that a two-bit counter can count from 0 to $2^2 - 1$ and back to 0 again.



Self-review 4.12 Draw a state transition diagram for a PIN (Personal Identification Number) recognizer. If the user enters '1' followed by '8' followed by '5' followed by '0' followed by 'enter', the machine should enter a state called 'accept'. If the user enters any other combination of numbers, followed by 'enter' the machine should enter a state called 'reject'. Is this program a lexer?

Self-review 4.13 The implementation of cipherCharacter on page ?? has a bug in it even though this bug may never appear in normal use. What is the bug and how can it be fixed?

The problem is that there is no wrap around, and the chr function requires its parameter to be in range(256).

Self-review 4.14 What are the types of variables a and b in the following code:

```
a = ( 1 )
b = ( 1 , )
```

Self-review 4.15 What is the types and value of y after executing the following code:

```
x = [ 1 , 2 , 3 , 4 , 5 ]
y = x [ : ]
```

Both x and y are lists. They have the same value, which is [1, 2, 3, 4, 5].

Programming Exercises

Exercise 4.1 Examine the following code:

```
a = 10
b = 20
print "Before swapping a=%d, b=%d" % ( a , b )
#Swap values
a = b
b = a
print "After swapping a=%d, b=%d" % ( a , b )
```

1. Try running this small program. What are the values of a and b after the swap?
2. Why is the result not a having the value 20 and b having the value 10 as appears to be the intention from the use of the word swap in the code?
3. Modify the code so that the final result is a having the value 20 and b having the value 1 using simultaneous assignment to swap the values.
4. Many languages, such as C, C++, Java, do not have simultaneous assignment. Modify the program so that it successfully swaps the two values without using simultaneous assignment.

1. Both a and b have the value 20.
2. Because the line a = b sets a to 20, leaving b as it was.
3. a, b = b, a

4.
```
a = 10
b = 20
print "Before swapping a=%d, b=%d" % (a, b)
# Swap values
temp = a
a = b
b = temp
print "After swapping a=%d, b=%d" % (a, b)
```

Exercise 4.2 This question is based on a function for stepping through a representation of a pack of cards. The exact nature of the cards isn't important. We use a list to represent pack and in our example we use numbers between 1 and 5 as cards. They could equally be numbers and suits as tuples, strings of character names, etc. Here's the function:

```
# A hand of cards
cards = [ 1 , 5 , 3 , 4 , 2 , 3 , 2 ]

def nextCard ( cards ) :
    next = cards[0]
    newHand = cards[1:] + [ cards[0] ]
    return next , newHand
```

- What is the type of values returned by this function?
- Describe in words what the function does.
- It would be a simple matter for this function to alter the input list so that the first element becomes the last. This would simplify the return value, which is currently two items. Why might this be considered a poorer solution that the function as it is now?
- Write a loop that calls this function repeatedly, printing out the card from the top of the deck each time.
- Using the given function and your answer to the previous question as a starting point, write a program that checks a deck of cards for consecutive identical pairs (a 2 followed by a 2, for example). If a pair is found, a message should be displayed.
- What happens if the input list is empty? Correct the function so that it returns (`None` , [ ]) when an empty list is used as input.

Exercise 4.3 Write a program to create a frequency table using a dictionary from a set of input values. The program should repeatedly asks the user for an input. The input could be anything at all. As each input is gathered, see if it exists as a key in the dictionary. If it does not exist, associate the input as a key with the number 1. If it does exist, add one to the value already associated with it and add it back to the dictionary. If the value associated with this key is now greater than or equal to 3, print a message that looks something like "Dear user, you have entered the word 'discombobulated' 3 times!"

Frequency tables such as this are often used for storing histogram data. We will look at actually drawing histograms later (Chapter 8). Frequency tables are also used in some sorting algorithms.

Hint: You will need an infinite loop and to use the raw_input function.

Hint: someDict.has_key ('a key') returns a Boolean value indicating whether the dictionary someDict has already got a value associated with the key 'a key'.

```
def frequency ( ) :
    d = { }
    while True :
        user = raw_input ( 'Please enter the next value or QUIT to finish: ' )
        if user == 'QUIT' : return
        elif d.has_key ( user ) :
            d[user] += 1
            if d[user] > 2 : print 'Dear user, you have entered the value', user, d[user], 'times!'
        else :
            d[user] = 1
```

Exercise 4.4 Write the Python program that manages the state of the two-bit binary counter from Self-review 4.11

Exercise 4.5 The lexer in Section ?? can only recognize non-negative integers. Improve the program by adding code to deal with negative numbers.

```
def getIntegerValue ( buffer ) :
    isNegative = False
    if buffer[0] == '-' :
        isNegative = True
        buffer = buffer[1:]
    for i in range ( len ( buffer ) ) :
        if not buffer[i].isdigit ( ) : break
    if isNegative:
        return -1 * int ( buffer [:(i+1)] )
    else:
        return int ( buffer [:(i+1)] )
```

Exercise 4.6 Create a program for storing a week's worth of rainfall data. Use a list to store each day's value, entered sequentially by the user. When an entire week has been input, display the days with the minimum and maximum rainfall.

```
import math

def rainfall ( ) :
    days = [ 'Mon' , 'Tue' , 'Wed' , 'Thu' , 'Fri' , 'Sat' , 'Sun' ]
    rain = [ ]
    for i in range ( 7 ) :
        r = input ( 'Please enter the rainfall for ' + days[i] + ': ' )
        rain.append ( r )
    minimum = rain[0]
```

```
    maximum = rain[0]
    for i in rain :
        if i < minimum : minimum = i
        if i > maximum : maximum = i
    print 'Min rainfall for this week:' , minimum
    print 'Max rainfall for this week:' , maximum
    for i in range ( len ( rain ) ) :
        print 'Rainfall for' , days[i] , '=' , rain[i]

if __name__ == '__main__' :
    rainfall ( )
```

Exercise 4.7 Extend your answer to the previous exercise so that it also displays the mean and standard deviation of the values. The mean is the sum of all the values divided by the number of values. The standard deviation is the square root of the sum of the squares of the difference between each value and the mean, divided by the number of items.

```
import math

def rainfall ( ) :
    days = [ 'Mon' , 'Tue' , 'Wed' , 'Thu' , 'Fri' , 'Sat' , 'Sun' ]
    rain = [ ]
    for i in range ( 7 ) :
        r = input ( 'Please enter the rainfall for ' + days[i] + ': ' )
        rain.append ( r )
    # Minimum and Maximum
    minimum = rain[0]
    maximum = rain[0]
    for i in rain :
        if i < minimum : minimum = i
        if i > maximum : maximum = i
    mean = sum ( rain ) / len ( rain )
    sd = math.sqrt ( sum ( [ ( x - mean ) ** 2 for x in rain ] ) / ( len ( rain ) - 1 ) )
    # Print everything to the console
    print 'Min rainfall for this week:' , minimum
    print 'Max rainfall for this week:' , maximum
    print 'Mean rainfall for this week:' , mean
    print 'Standard deviation rainfall for this week:' , sd
    for i in range ( len ( rain ) ) :
        print 'Rainfall for' , days[i] , '=' , rain[i]

if __name__ == '__main__' :
    rainfall ( )
```

Exercise 4.8 Calculating statistics such as maximum, minimum, mean and standard deviation is a common task. Now that you have a program to do it, package the code up into a function that can be used again in future.

Exercise 4.9 A "wobbly" number is one in which the digits alternate between being higher and lower than the preceding one. Here are some wobbly numbers: 19284756242, 90909, 0909. Using what you have learned

about writing lexers, write a function that accepts a list of digits to be checked for wobbliness. If the sequence of digits is wobbly, the function should return **True**, otherwise **False**.

```
def isWobbly ( num ) :
    if num == '' : return False
    elif not ( len ( num ) > 2 ) : return False
    if not ( num[0].isdigit ( ) or num[1].isdigit ( ) ) : return False
    isLower = int ( num[0] ) < int ( num[1] )
    curr = int ( num[1] )
    for i in num[2:] :
        if not i.isdigit ( ) : return False
        d = int ( i )
        if ( curr < d ) == isLower : return False
        elif curr == d : return False
        isLower = curr < d
        curr = d
    return True
```

Exercise 4.10 The following code was written before the programmer had drunk their morning coffee, i.e. they were not properly awake at the time. It is supposed to add 3% to all the values in the variable salary_scale. However, there is one very important bug in this code. Find and fix the bug to make sure everyone gets their 3% pay rise!

```
salaryScale = ( 20000 , 21500 , 23000 , 24500 , 26000 , 27500 , 29000 )
for i in range ( len ( salaryScale ) ) :
    salaryScale[i] += salaryScale[i] * 0.03
print salaryScale
```

Tuples are immutable so the code generates a TypeError when executed.

Exercise 4.11 Starting with the list:

```
[ 43 , 1 , 2 , 99 , 54 , 78 , 22 , 6 ]
```

write a function to sort the list – i.e. your function should return the list:

```
[ 1 , 2 , 6 , 22 , 43 , 54 , 78 , 99 ]
```

You can use any method you like to sort the list. Good answers to this problem will work on any list, not just the one we've given here.

Challenges

Challenge 4.1 Imagine a fictional land where monetary units aren't based on decimal orders. In this land, we have 3 basic units of currency:

- The Blink. The smallest unit of currency.
- The Hoojim. Worth 12 Blinks.
- The Bung. Worth 20 Hooja (plural of Hoojim) or 240 Blinks.

1. Write a function called deBung that accepts an integer representing a number of Bungs and displays the number of Hooja and Blink it is worth. For example, calling the function like this:

   ```
   deBung ( 4 )
   ```

   Will produce the following output:

   ```
   4 Bungs is worth 80 Hoojim or 960 Blinks.
   ```

2. Write a function called enBlinkHoojaBung that takes a number of Blinks and outputs its equivalent in Blink, Hooja and Bung, using the smallest number of coins. Coins in our imaginary land are made of a very heavy metal, so this is important. If the function is called with the value 506, the output should be:

   ```
   506 Blinks is worth 2 Bung, 1 Hoojim and 6 Blinks.
   ```

   You will need the remainder operator (%).

3. Rewrite enBlinkHoojaBung so that it returns a tuple of Blinks, Hooja and Bung values instead of writing to the screen. The last example would return (2 , 1 , 6).

Challenge 4.2 Referring back to Exercise 4.2, write a program that uses two hands of cards (two lists) to play a simple game of "snap". No cards change hands, but when two identical cards are played, a message should be printed.

Challenge 4.3 Write a lexer to recognize floating point numbers.

Hint: If your input is the following string: '12345.9876' what do you have to do to each character in the string to generate your float? How are the numbers before the decimal point different to the numbers after the decimal point? Make sure you have an algorithm that works correctly on paper before you start coding and draw a state transition diagram for the scanning part of your lexer.

5

Functionally Modular

Self-Review Questions

Self-review 5.1 Which names are *local*, which are *global* and which are *built-in* in the following code fragment?

```
space_invaders = [ . . . ]
player_pos = ( 200 , 25 )
level = 1
max_level = 10

def play ( ) :
    . . .
    while ( level < max_level +1 ) :
        if len ( space_invaders ) == 0 :
            level += 1
            continue
        . . .
```

Global names:

- space_invaders
- player_pos
- level
- max_level

Built-in names:

- len

. . . and there are no local names.

Self-review 5.2 What is special about a *recursive* function?

Self-review 5.3 What is a *side effect*?

Self-review 5.4 What does the term *referential transparency* mean?

A function is *referentially transparent* if it returns the same result every time you execute it with the same arguments. In programming languages such as Python, this is the same as saying that the function does not update any variables – or that the function has no *side effects*. So, for example the following function is referentially transparent:

```
def sum ( l ) :
    if l == [ ] : return 0
    else : return 1 + sum ( l [ 1 : ] )
```

but this next function is not referentially transparent, because it updates the variable a:

```
a = 10
def example ( ) :
    global a
    a = a + 1
    return a * 10
```

Self-review 5.5 What list of lists does the following list comprehension evaluate to?

```
[ [ '#' for col in range ( 3 ) ] for row in range ( 3 ) ]
```

Self-review 5.6 What is the result of executing the following code?

```
def dp ( l1 , l2 ) :
    def p ( ll1 , ll2 , n ) :
        return ll1[n] * ll2[n]
    r = 0
    for i in range ( len ( l1 ) ) :
        r += p ( l1 , l2 , i )
    return r

print dp ( [ 1 , 2 , 3 ] , [ 4 , 5 , 6 ] )
```

This code prints 32.
The function p multiplies two elements in a particular index in two given lists. So, for example p ([1 , 2 , 3] , [4 , 5 , 6] , 1) will return the product of the second elements in the two lists, which is 2 * 5. The outer function dp uses p to multiply all the corresponding elements of the two lists in the arguments and totals the results. So, for the two lists in the function call, we get (1 * 4) + (2 * 5) + (3 * 6) which is 4 + 10 + 18, or 32.
The function dp is named after the idea of a "dot product" in mathematics – this is a very useful concept that is used in applications such as games and image processing.

Self-review 5.7 In Python modules, what is the following code idiom used for?

```
if __name__ == '__main__':
    . . .
```

Self-review 5.8 What is wrong with the following code?

```
from turtle import *
turtle.forward ( 100 )
```

Self-review 5.9 What are the following special variables used for?

1. __author__

2. __date__
3. __copyright__
4. __version__
5. __revision__
6. __license__
7. __credits__

Self-review 5.10 What is the result of executing the following Python code?

```
l = [ i for i in range ( 1 , 100 ) ]
map ( lambda x : x ** 3 , l )
```

Is there a better way of expressing the algorithm in Python?

The list l contains the integers from 1 to 99. The map function returns a list of integers which are the cubes of the list l, so [1 * 1 * 1 , 2 * 2 * 2 , 3 * 3 * 3 , . . .] or [1 , 8 , 27 , 64 , 125 , . . .].
There are several ways to do this in Python, but one way is to use list comprehensions:

```
cubes = [ i ** 3 for i in range ( 1 , 100 ) ]
```

Self-review 5.11 What is the result of the following Python code?

```
import string
v = [ 'a' , 'e' , 'i' , 'o' , 'u' , 'A' , 'E' , 'I' , 'O' , 'U' ]
filter ( lambda x : x in string.uppercase , v )
```

Self-review 5.12 What is a generator in Python?

Self-review 5.13 What does the Python keyword yield mean?

Self-review 5.14 What does the following Python function do?

```
def geni ( n ) :
    for i in range ( n ) :
        yield i
```

The function geni returns a *generator* which generates the integers from 1 to n. For example:

```
>>> def geni ( n ) :
...     for i in range ( n ) :
...             yield i
...
>>> g = geni ( 10 )
>>> g.next ( )
0
>>> g.next ( )
1
>>> g.next ( )
2
>>> g.next ( )
```

```
3
>>> g.next ( )
4
>>> g.next ( )
5
>>>
```

Self-review 5.15 Using the documentation on the Python website, find out what the exception StopIteration is used for.

Programming Exercises

Exercise 5.1 The following Python code represents a Tic-Tac-Toe board as a list of lists:

```
[ [ '#' , 'o' , 'x' ] , [ '#' , '#' , 'o' ] , [ 'x' , '#' , 'o' ] ]
```

The # symbols represent blank squares on the board. Write a function print_board that takes a list of lists as an argument and prints out a Tic-Tac-Toe board in the following format:

```
   | o | x
------------
   |   | o
------------
 x |   | o
```

```
def print_board ( board ) :
    for row in ( 0 , 1 , 2 ) :
        for col in ( 0 , 1 , 2 ) :
            if board[row][col] == '#' :
                print ' ' ,
            else : print board[row][col] ,
            if col < 2 : print ' | ' ,
        if row < 2 : print '\n-----------'
    print # Print an empty line
```

Exercise 5.2 Convert the following iterative functions into recursive functions:

Any iterative function can be recoded as a recursive function, and any recursive function can be recoded as an iterative function.

1.
```
def sum_even ( n ) :
    total = 0
    for i in range ( 2 , n + 1 , 2 ) :
        total += i
    return total
```

2.
```
def min ( l ) :
    m = 0
    for i in l :
        if i<m : m = i
    return m
```

3.
```
def prod ( l ) :
    product , i = 1 , 0
    while i < len ( l ) :
        product *= l[i]
        i += 1
    return product
```

Hint: You may find it useful to add an extra parameter to the recursive version of min.

Exercise 5.3 Convert the following recursive functions into iterative ones:

1.
```
def sum_odd ( n , total ) :
    if n == 1 : return total
    elif n % 2 == 0 : return sum_odd ( n - 1 , total )
    else : return sum_odd ( n - 2 , total + n )
```

2.
```
def max ( l , n ) :
    if l == [ ] : return n
    elif l[0] > n : return max ( l[1:] , l[0] )
    else : return max ( l[1:] , n )
```

3.
```
def mylen ( l , n ) :
    if l == [] : return n
    else : return mylen ( l[1:] , n +1 )
```

```
def sum_odd ( n ) :
    total = 0
    for i in range ( n + 1 ) :
        if i % 2 != 0 :
            total += i
    return total
```

```
def max ( l ) :
    m = 0
    if l == [ ] : return None # Dummy value
    else : m = l[0]
    for i in l :
        if i > m : m = i
    return m
```

```
def mylen ( l ) :
    if l == [ ] : return 0
    else : total = 0
    for i in l :
        total += 1
    return total
```

Exercise 5.4 The following code is a module that provides functions for drawing some simple shapes using the Python Turtle module. Copy the code into a file called shape.py and add in:

- Documentation for the module and its functions.
- Special variables such as __date__.

- Some simple testing.

Make sure you use pydoc to generate a webpage containing your documentation.

```
__author__ = 'Sarah Mount'

from turtle import *

def square ( n ) :
    for i in range ( 4 ) :
        forward ( n )
        left ( 90 )

def rectangle ( s1 , s2 ) :
    for i in range ( 2 ) :
        forward ( s1 )
        left ( 90 )
        forward ( s2 )
        left ( 90 )

def pentagon ( s ) :
    for i in range ( 5 ) :
        forward ( s )
        left ( 360 / 5 )
```

Exercise 5.5 Use map and **lambda** to turn a list of integers from 1 to 100 into a list of even numbers from 2 to 200.

```
map ( lambda x : x * 2 , range ( 1 , 101 ) )
```

Exercise 5.6 Use filter to generate a list of odd numbers from 0 to 100.

Exercise 5.7 Use a list comprehension to generate a list of odd numbers from 0 to 100.

```
[ i for i in range ( 101 ) if i % 2 != 0 ]
```

Exercise 5.8 Write a generator function (using the yield keyword) that generates factorial numbers.

Exercise 5.9 Ackermann's Function is defined as:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1,A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

Write a recursive Python function to implement Ackermann's Function. How many recursive calls will be required to evaluate $A(2,3)$?

```
def ackermann ( m , n ) :
    if m == 0 : return n + 1
    elif m > 0 and n == 0 : return ackermann ( m - 1 , 1 )
    else : return ackermann ( m - 1 , ackermann ( m , n - 1 ) )
```

We can find the number of recursive calls needed to compute ackermann (2 , 3) using a walk-through. This is similar to the work we did in Section **??** 3.3.1 when we looked at the recursive definition of the factorial function.

```
ackermann ( 2 , 3 )   ⟶   ackermann ( 1 , ackermann ( 2 , 2 ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 2 , 1 ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 1 , ackermann ( 2 , 0 ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 1 , ackermann ( 1 , 1 ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , ackermann ( 1 , 0 ) ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , 1 ) ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , 2 ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 1 , 3 ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , ackermann ( 1 , 2 ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 , 1 ) ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 , 0 ) ) ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , 1 ) ) ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , 2 ) ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , 3 ) ) ) )
                          ackermann ( 1 , ackermann ( 1 , ackermann ( 0 , 4 ) ) )
                          ackermann ( 1 , ackermann ( 1 , 5 ) )
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 1 , 4 ) ) )
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 , 3 ) ) ) )
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 , 2 ) ) ) ) )
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 , 2 ) ) ) ) ) )
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 ,
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 ,
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 ,
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , 2 ) ) ) ) ) ) )
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , 3 ) ) ) ) )
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , 4 ) ) ) )
                          ackermann ( 1 , ackermann ( 0 , ackermann ( 0 , 5 ) ) )
                          ackermann ( 1 , ackermann ( 0 , 6 ) )
                          ackermann ( 1 , 7 )
                          ackermann ( 0 , ackermann ( 1 , 6 ) )
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 1 , 5 ) ) )
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 , 4 ) ) ) )
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 , 3 ) ) ) ) )
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 , 2 ) ) ) ) ) )
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 1 ,
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 ,
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 ,
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 ,
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , 3 ) ) ) ) ) ) )
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , 4 ) ) ) ) )
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , 5 ) ) ) )
                          ackermann ( 0 , ackermann ( 0 , ackermann ( 0 , 6 ) ) )
                          ackermann ( 0 , ackermann ( 0 , 7 ) )
                          ackermann ( 0 , 8 )
                          9
```

Exercise 5.10 In Section ?? (page ??) we introduced palindromes as words or phrases that read the same forwards and backwards, and showed implementations using iteration and negative indexing. For this question we say that 'deed', 'peep' and 'a man, a plan, a canal, panama!' are all palindromes – so we do not make spaces significant. Write a recursive function implementation of isPalindrome to test whether or not a string is a palindrome.

Exercise 5.11 The first five rows of Pascal's Triangle are:

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
  . . .
```

Each number is generated by adding the two above it. Write a recursive function to generate the first *n* lines of Pascal's Triangle.

Challenges

Challenge 5.1 Create a module for playing Tic-Tac-Toe.

Hint: You may want to consider the following functions:

1. *print_board ( ) – from the programming exercise above, except that you will want to use a global board variable, rather than a function argument.*
2. *has_won ( ) – check to see whether either player has won. This function should return the string 'o' or 'x' if either the o or x player has won and '#' if neither player has won. A player wins in Tic-Tac-Toe if they have three of their counters in a row, a column or a diagonal.*
3. *place_counter (sq , counter) – place the counter on a particular square of the board. The first argument should be a number between 0 and 8 and the second argument should be either 'o' or 'x'. You should consider the squares on the board to be numbered as in the diagram below.*

```
 0 | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
```

   Using a numbering such as this one makes the answer to this challenge simpler!
4. *next_play ( ) – This function should ask the user for the next move they want to play. You should make sure that the user knows whether x or o is currently playing. You can assume that the user will enter an integer value. You should still check that the integer the player has provided is between 0 and 8 inclusive.*

5. *play ( ) – This is the top-level function that tells Python how to play Tic-Tac-Toe. The algorithm for game play is:*
   - *If one of the players has won the game, print a message on the screen to congratulate them.*
   - *If no-one has won the game, then:*
     - *Use the next_play function to get the current player's next move.*
     - *Play the next move by calling the place_counter function.*
     - *Print the new board.*
     - *Change the current player – either from 'x' to 'o' or from 'o' to 'x'.*

You will also need two global variables, one to store the current state of the board and one to say which player is playing next. Below is an outline of the module to get you started – make sure you add some documentation and make good use of Python's special variables, like __author__ and __version__.

```
board = [ ['#' for col in range ( 3 ) ] for row in range ( 3 ) ]
o_playing = True
def play ( ) :
    # Your code here!
def next_play ( ) :
    # Your code here!
def has_won ( ) :
    # Your code here!
def place_counter ( ) :
    # Your code here!
def print_board ( ) :
    # Your code here!

if __name__ == '__main__':
    play ( )
```

Challenge 5.2 One reason for selecting a recursive version of an algorithm over an iterative one, or vice versa, is running time. There are various ways of assessing whether one algorithm is faster than another. Big O notation is a very important one – we will come to this in Chapter 10. A technique for assessing the speed of programs implementing algorithms is to executing benchmarks. Benchmarks require us to be able to time the execution of a program. Python helps us here, as it provides a module called timeit that can be used to time the execution of Python expressions. Here is an example of the timeit module in use:

```
>>> import timeit
>>>
>>> def is_prime ( n ) :
...     return not bool ( filter ( lambda x: n%x == 0, range ( 2 , n ) ) )
...
>>> def primes ( n ) :
...     return filter ( is_prime , range ( 2 , n ) )
```

```
...
>>> t = timeit.Timer ( 'primes ( 100 )' , 'from __main__ import primes' )
>>> print t.timeit ( 500 )
1.47258400917
>>>
```

Here, we have written some code to time one of our prime number generators, and found out how long it has taken Python to execute the code 500 times. We did this by creating an object called timeit.Timer that took a call to our primes function as its first argument. The second argument, 'from __main__ import primes' tells the timeit module where it can find the primes function – in this case in the __main__ namespace.

The argument to the timeit method, 500, means 'execute the code 500 times'. Python code generally executes very quickly, so timing a very short piece of code that only runs once can be inaccurate. So, instead, we can ask timeit to time the code over several hundred or thousand runs, which will give us more accurate information. In the example above, the code took 1.47258400917 s to run.

For this challenge, test the time of execution of all the prime number generators we have presented in this book, timing them using the timeit module, to determine which is the fastest.

Challenge 5.3 Chess is played on an 8×8 board. A *queen* may attack any piece on the same row, column or diagonal as herself.

The Eight Queens Problem involves placing eight queens on a chess board in such a way that no queen is attacking any other – so, there must be at most one queen on any row, column or diagonal.

There are 92 solutions to the Eight Queens Problem and the challenge is to write a program that finds all of them.

Hint: Think about how to represent the board. A list of lists of Booleans might be a sensible choice, but are there other options? Whatever you use, you will probably want to make sure that you can print chess boards in a human-readable format, such as this:

```
Q X X X X X X X
X X Q X X X X X
X X X X X Q X X
X Q X X X X X X
X X X X X X X Q
X X X Q X X X X
X X X X X X Q X
X X X X Q X X X
```

The Eight Queens Problem is the subject of the classic paper, *Program Development by Stepwise Refinement* by Nicklaus Wirth. You can find his paper at the ACM 'classics' website: http://www.acm.org/classics/dec95/

6

Classy Objects

Self-Review Questions

Self-review 6.1 Explain the roles of r, random, and ( ) in:

```
r.random ( )
```

r is an object, random is a method which is a member of the object r and () is the (empty) list of arguments passed to the method random.

Self-review 6.2 What is an object?

Self-review 6.3 Why are classes important?

Self-review 6.4 What is a method?

A *method* is a piece of executable code, a bit like a function. Unlike a function, methods are members of objects and thier first argument is always the object to which the method belongs (this argument is usually called self).

Self-review 6.5 Why is a method different than a function?

Self-review 6.6 Why is the method name __init__ special, and for what reason(s)?

Self-review 6.7 What is an *abstract data type*?

An *abstract data type* is a type whose internal data structures are hidden behind a set of access functions or methods. Instances of the type may only be created and inspected by calls to methods. This allows the implementation of the type to be changed without changing any code which instantiates that type.
Queues and stacks are examples of abstract data types.

Self-review 6.8 Why is there an apparent difference between the number of parameters in the definition of a method and the number of arguments passed to it when called?

Self-review 6.9 What does self mean? Is it a Python keyword?

Self-review 6.10 Is the method bar in the following code legal in Python?

```
class Foo :
    def __init__ ( self ) :
        self.foo = 1
    def bar ( flibble ) :
        print flibble.foo
```

Self-review 6.11 What is *operator overloading*?

Operator overloading means creating a new definition of an operator (such as + or -) for a new datatype. For example the following class overloads the + and * operators, and the built-in function which tells Python how to print out a Vector object:

```
class Vector :
    def __init__ ( self , l ) :
        self.values = l
    def __add__ ( self , other ) :
        if len ( self.values ) != len ( other.values ) :
            raise AssertionError, 'Vectors have different sizes.'
        v = [ ]
        for i in range ( len ( self.values ) ) :
            v.append ( self.values[i] + other.values[i] )
        return Vector ( v )
    def __mul__ ( self , other ) :
        if len ( self.values ) != len ( other.values ) :
            raise AssertionError, 'Vectors have different sizes.'
        v = [ ]
        for i in range ( len ( self.values ) ) :
            v.append ( self.values[i] * other.values[i] )
        return Vector ( v )
    def __str__ ( self ) :
        return 'Vector ' + self.values.__repr__ ( )
```

Self-review 6.12 Which operators do the following methods overload?

1. __add__
2. __eq__
3. __lt__
4. __or__
5. __ne__
6. __div__
7. __ge__

Self-review 6.13 What is the difference between a queue and a stack?

Self-review 6.14 Why is it usually thought to be a bad thing to use **global** statements?

Self-review 6.15 What does the __ mean in the name __foobar in the following class?

```
class Foo :
    def __init__ ( self ) :
        self.__foobar ( )
    def __foobar ( self ) :
        print 'foobar'
```

The double underscore means that the method __foobar can only be called by code within the class Foo. For example:

```
>>> class Foo:
...     def __foobar(self):
...             print 'foobar'
...     def foobar(self):
...             self.__foobar()
...
>>> f = Foo()
>>> f.foobar()
foobar
>>> f.__foobar()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: Foo instance has no attribute '__foobar'
>>>
```

Self-review 6.16 What new facilities for encapsulation do classes offer?

Programming Exercises

Exercise 6.1 Create a class to represent a single die that can have any positive integer number of sides. This kind of die might be used when playing role-playing games (RPGs).

```
class AnyDie:
    def __init__ ( self , sides ) :
        assert type ( sides ) == type ( 0 )
        assert sides > 0
        self.sides = sides
        self.generator = random.Random ( )
        self.current = 1
        self.representations = map ( lambda x : '*' * x , range ( self.sides + 1 ) )
    def __str__ ( self ) :
        return self.representations[self.current]
    def roll ( self ) :
        self.current = self.generator.randint ( 1 , self.sides )
        return self.current
```

Exercise 6.2 Write a class to represent an RPG character's money pouch. Money should be stored as an integer, with methods for adding money and removing money. The removing method should take a value as parameter. If there is enough money, the value is removed from the money in the pouch and **True** is returned. If there is not enough money, **False** is returned.

Exercise 6.3 Write a simple RPG character class. The character should have a name, a money pouch and an inventory. The name should be stored as a string, the money pouch should be an instance of the pouch from the previous exercise and the inventory should be a dictionary in which keys are item names and values are the number of items held by the player.

Ensure that there are methods for adding items to, and removing items from, the inventory.

There should be a __str__ method that returns something that can be printed. For example:

```
print playerA
```

Might display:

```
-------------------------
Duncan Disorderly
-------------------------
Money: 235 gold pieces
-------------------------
Knapsack contains:

 Arrow: 12
 Rubber Sword: 1
 Felt tipped pen: 2
 Imitation fur coat: 23
-------------------------
```

```
class Pouch :
    def __init__ ( self , money ) :
        self.money = money
    def add ( self , money ) :
        self.money += money
    def remove ( self , money ) :
        if not ( self.money > money ) : return False
        self.money -= money
        return True
    def __str__ ( self ) :
        return 'Money: ' + str ( self.money ) + ' gold pieces.'

class Character :
    def __init__ ( self , name ) :
        self.name = name
        self.pouch = Pouch ( 0 )
        self.inventory = { }
    def pickupItem ( self , item ) :
        if self.inventory.has_key ( item ) :
            self.inventory[item] += 1
        else:
            self.inventory[item] = 1
    def dropItem ( self , item ) :
        if not self.inventory.has_key ( item ) : pass
        elif self.inventory[item] < 1: pass
        else : self.inventory[item] -= 1
    def pickupMoney ( self , money ) :
```

```
        self.pouch.add ( money )
    def dropMoney ( self , money ) :
        self.pouch.remove ( money )
    def __str__ ( self ) :
        sep = '----------\n'
        s = sep + self.name + '\n' + sep + str ( self.pouch ) + '\n' + sep + 'Knapsack contains:\n\n'
        for key in self.inventory.keys ( ) :
            if self.inventory[key] > 0 :
                s += ( '\t' + key + ': ' + str ( self.inventory[key] ) + '\n' )
        s += sep
        return s
```

Exercise 6.4 Implement the multiplication operation for the Matrix class.

The trick to implementing multiplication is to realize you need three, nested loops:

```
if len ( self.data[0] ) != len ( m.data ) :
    raise ValueError , 'Matrices not of the suitable shapes for multiplication.'
n = zeros ( len ( self.data ) , len ( m.data[0] ) )
for row in range ( len ( n.data ) ) :
    for column in range ( len ( n.data[0] ) ) :
        for i in range ( len ( self.data[0] ) ) :
            n.data[row][column] += self.data[row][i] * m.data[i][column]
return n
```

Exercise 6.5 Implement __getitem__ for the Matrix class so that we can rewrite the drawTriangle function to work with a triplet of 2×1 matrices:

```
def drawTriangle ( coordinateMatrix ) :
    turtle.up ( )
    turtle.goto ( coordinateMatrix[0][0][0] , coordinateMatrix[0][1][0] )
    turtle.down ( )
    turtle.goto ( coordinateMatrix[1][0][0] , coordinateMatrix[1][1][0] )
    turtle.goto ( coordinateMatrix[2][0][0] , coordinateMatrix[2][1][0] )
    turtle.goto ( coordinateMatrix[0][0][0] , coordinateMatrix[0][1][0] )
```

We can use __getitem__ in exactly the same way that we would use any list subscript. For example:

```
>>> m = Matrix(2,2)
>>> m[(0,0)]
0.0
>>>
```

Exercise 6.6 Write a class Account that stores the current balance, interest rate and account number of a bank account. Your class should provide methods to withdraw, deposit and add interest to the account. The user should only be allowed to withdraw money up to some overdraft limit. If an account goes overdrawn, there is fee charged.

Exercise 6.7 Write a small class to represent the light switch state machine from Section ??. Provide a single method to change the state of the switch and method called isOn which returns **True** if the switch is on and **False** if it is off. Make sure you override the __str__ method so that light switches can be printed to the console.

```
class Account:
    interest_rate = 0.05
    overdraft_limit = 500
    overdraft_charge = 50
    def __init__ ( self , number ) :
        self.number = number
        self.balance = 0
    def deposit ( self , amount ) :
        if amount < 0 : raise ValueError, "Can't deposit a negative amount of money."
        self.balance += amount
    def withdraw ( self , amount ) :
        if amount < 0 : raise ValueError, "Can't withdraw a negative amount of money."
        if amount > ( self.balance + Account.overdraft_limit ) : raise ValueError, 'Out of credit.'
        if amount > self.balance :
            self.balance -= amount
            self.balance -= Account.overdraft_charge
        else :
            self.balance -= amount
    def __str__ ( self ) :
        sep = '----------'
        s = '\nAccount\n' + sep + '\n'
        s += 'Account number: ' + str(self.number) + '\n' + sep + '\n'
        s += 'Overdraft limit: ' + str(Account.overdraft_limit) + '\n'
        s += 'Overdraft charge: ' + str(Account.overdraft_charge) + '\n' + sep + '\n'
        s += 'Account balance: ' + str(self.balance) + '\n' + sep + '\n'
        return s
```

Exercise 6.8 Write a program which *uses* the Stack class. Your program should begin by printing a menu to the user:

```
1. Add new data to stack
2. Print stack
3. Remove datum from stack
4. Exit
```

You should allow the user to enter 1, 2, 3 or 4 to select their desired action and you should write code to implement the four possible options.

Exercise 6.9 Amend your program from Exercise 6.8 to use a Queue as the data structure used to store data in.

```
menu = """1. Add new data to the queue.
2. Print queue.
3. Remove datum from the queue.
4. Exit.
"""

if __name__ == '__main__' :
    q = Queue ( )
    while True :
        print menu
        m = 0
        while m < 1 or m > 4 :
            m = input ( 'Enter: ' )
```

```
if m == 1 :
    datum = raw_input ( 'Enter datum: ' )
    q.add ( datum )
elif m == 2 :
    print 'Queue:' , q , '\n'
elif m == 3 :
    datum = q.remove ( )
    print 'Removed' , datum , 'from the queue.\n'
elif m == 4 :
    print 'Goodbye!'
    break
```

Exercise 6.10 A *priority queue* is an abstract data type similar to the queue introduced in Section ?? and Section ??. A priority queue associates a priority with each stored item and always stores the items so that the elements with the highest priority are at the 'top' of the queue and are the first to be removed – i.e. the items in a priority queue are sorted by priority. Create a PriorityQueue class based on the Queue class:

1. The add method should take two parameters, an item to store and a priority, which should be an integer.
2. The add method should ensure that when new data is added to the priority queue, it is added as a tuple which contains both the data and its priority. Make sure that data is always stored in priority order.
3. The remove method should return queue items, not tuples – i.e. the priority associated with the returned item can be ignored.

Challenges

Challenge 6.1 Currently, the *n*-faced die (see Exercise 6.1) is unable to display its spots and must instead rely on displaying a number.

Your task is to write a function that returns a string representation of the die face for a given number. For example, this:

```
print makeFace ( 9 )
```

might display:

```
***
***
***
```

The algorithm is up to you, but do remember about integer division and the remainder operator (%)

Now that you can make faces with an arbitrary number of spots, add this functionality to your *n*-sided die class.

Challenge 6.2 Extend your RPG character class to hold values for health points (HP), attack points (AP) and defence points (DP).

Add an attack method that takes a character instance as a parameter. This is your opponent.

If the character's AP is greater than the opponent's DP, the difference is subtracted from the opponent's HP. So, if I attack with a power of 7 and my opponent has a defence power of 6, they lose 1 health point. If they have a defence power of 9, they sustain no damage.

Write a program that demonstrates your character class by creating two characters that take it in turns to bop each other until one of them runs out of HP.

```
from pouch import Pouch
import random

class Character :
    def __init__ ( self , name ) :
        self.name = name
        self.pouch = Pouch ( 0 )
        self.inventory = { }
        self.hp = 100
        self.ap = random.randint ( 1 , 100 )
        self.dp = random.randint ( 1 , 100 )
    def attack ( self , enemy ) :
        if self.ap > enemy.dp :
            enemy.hp -= ( self.ap - enemy.dp )
    def pickupItem ( self , item ) :
        if self.inventory.has_key ( item ) :
            self.inventory[item] += 1
        else:
            self.inventory[item] = 1
    def dropItem ( self , item ) :
        if not self.inventory.has_key ( item ) : return
        elif self.inventory[item] < 1 : return
        else : self.inventory[item] -= 1
    def pickupMoney ( self , money ) :
        self.pouch.add ( money )
    def dropMoney ( self , money ) :
        self.pouch.remove ( money )
    def __str__ ( self ) :
        sep = '----------\n'
        s = sep + self.name + '\n' + sep
        s += 'Health:\t' + str(self.hp) + '\n'
        s += 'Attack:\t' + str(self.ap) + '\n'
        s += 'Defence:\t' + str(self.dp) + '\n'
        s += sep + str ( self.pouch ) + '\n' + sep + 'Knapsack contains:\n\n'
        for key in self.inventory.keys ( ) :
            if self.inventory[key] > 0 :
                s += ( '\t' + key + ': ' + str ( self.inventory[key] ) + '\n' )
        s += sep
        return s


if __name__ == '__main__' :
    player1 = Character ( 'Player 1' )
```

```
player2 = Character ( 'Player 2' )
print '***** Testing... Creating character\n'
print player1
print player2
print '***** Testing... Mortal Kombat!\n'
while player1.hp > 0 and player2.hp > 0 :
    # Player 1 always hits first -- grossly unfair :-)
    player1.attack ( player2 )
    player2.attack ( player1 )
    print player1
    print player2
```

7

Inheriting Class

Self-Review Questions

Self-review 7.1 What do the terms 'subclass' and 'superclass' mean in object-oriented programming?

Self-review 7.2 What does the following declaration mean?

```
class Foo ( Bar ) :
```

Self-review 7.3 What is the built-in name `object` in Python?

Self-review 7.4 What is the difference between single and multiple inheritance?

Self-review 7.5 What is the Diamond Problem in multiple inheritance?

Self-review 7.6 Define the terms:

1. Association.
2. Aggregation.

Self-review 7.7 In graphical user interfaces, what are:

1. Widgets.
2. Callbacks.
3. Events.
4. Event loops.
5. Event handler.

Self-review 7.8 What does the `pack` method do in Tkinter widgets?

Self-review 7.9 What is a modal dialog box?

Self-review 7.10 What is polymorphism? How is it related to object-oriented programming? Give a short example of a piece of code that exploits polymorphism.

Self-review 7.11 Using the expression evaluator from Section ?? (page ??), draw the object diagram that represents the following Python expression:

```
Mul ( 3 , Div ( Add ( 1 , 2 ) , Sub ( 5 , 8 ) ) )
```

What does that expression evaluate to?

Self-review 7.12 What is duck typing?

Self-review 7.13 What does the name self mean? Is self a Python keyword?

Self-review 7.14 Explain what is happening in the following code:

```
class Foo ( Bar ) :
    def __init__ ( self ) :
        Bar.__init__ ( self )
        return
```

Programming Exercises

Exercise 7.1 Extend the traffic light program to include a short red+amber 'prepare to go' state as is used for traffic lights in the UK, and other places.

Exercise 7.2 Create a subclasses of your Account class (from Exercise 6.6) called CreditAccount in which the user is charged a set amount for every withdrawal that is made. If the user is overdrawn, the withdrawal charge is doubled.

Exercise 7.3 Create a subclasses of your Account class (from Exercise 6.6) called StudentAccount in which new accounts start off with a balance of £500 and an overdraft of up to £3000 is allowed, with no charges for withdrawal.

Exercise 7.4 Create versions of:

1. DrawableRectangle
2. DrawableSquare
3. DrawableCircle

that work using the Tkinter package.

Exercise 7.5 Create a class called Number and two subclasses Binary and Roman for dealing with different sorts of number representation. The constructors to Binary and Roman should take a string argument so that you can use them like this:

```
b = Binary ( '11010101' )
r = Roman ( 'MCMLXXVIII' )
```

In your Number class you need to have methods that are common to all types of number representation:

1. You'll want a private method called __to_int that returns an ordinary integer representation of the calling object.
2. You'll need to override the built-in method __str__ that returns a string representation of the number which the calling object represents.
3. You'll need to override the __int__ method that is used by calls to the int function which converts its argument to an integer.

In all three classes you'll need an __int__ method that calls the __to_int method in the calling object – even though Binary and Roman are subclasses of Number, they still need an __int__ method of their own. Why is this? If you're not sure you have understood why this works, when you've finished this exercise, comment out the definition of __int__ in Binary and Roman and see what happens.

You'll need to override __to_int in the Binary and Roman classes. For the method in Roman, you may want to use the algorithm in Section ?? to parse a Roman numeral into an integer. Make the LUT (lookup table) a class variable (rather than an instance variable).

For the Binary class, you'll need to convert between binary and decimal. As a reminder of the algorithm, here's an example:

$$1010101 \rightarrow 1\times 2^6 + 0\times 2^5 + 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + +0\times 2^1 + 1\times 2^0$$

When you have written all your classes you'll need to test them. Below is the beginnings of a test program you can use to help you:

```
def test ( ) :
    data = [
        ( Binary ( '0' ) , 0 ) , ( Binary ( '1' ) , 1 ) ,
        ( Binary ( '10' ) , 2 ) , ( Binary ( '11' ) , 3 ) ,
        ( Binary ( '100' ) , 4 ) , ( Binary ( '101' ) , 5 ) ,
        ( Binary ( '10101010101' ) , 1365 ) ,
        ( Roman ( 'I' ) , 1 ) , ( Roman ( 'II' ) , 2 ) ,
        ( Roman ( 'IV' ) , 4 ) , ( Roman ( 'VI' ) , 6 ) ,
        ( Roman ( 'IX' ) , 9 ) , ( Roman ( 'X' ) , 10 ) ,
        ( Roman ( 'XI' ) , 11 ) , ( Roman ( 'MM' ) , 2000 ) ,
        ( Roman ( 'MCMLXXVIII') , 1978 )
        ]
    for entry in data :
        assert int ( entry[0] ) == entry[1]

if __name__ == '__main__' :
    test ( )
```

When your code works you should get no assertion failures when you execute the test program.

Exercise 7.6 On page ?? we introduced the Diamond Problem of object-oriented programming languages and explained how Python deals with it. In

this exercise you will explore how Python handles multiple inheritance.

Below are class definitions that exhibit the Diamond Problem. Start off by drawing a diagram to show the inheritance relationships between the five classes.

Create objects that are instances of classes D and E. Call the foobar and barfoo methods in both objects. What is the difference between the output from the two objects? Why does this happen?

```
class A :
    def foobar ( self ) :
        print 'Class A'
    def barfoo ( self ) :
        print 'barfoo from class A!'
class B ( A ) :
    def foobar ( self ) :
        print 'Class B inherits from A'
    def barfoo ( self ) :
        print 'barfoo from class B!'
class C ( A ) :
    def foobar ( self ) :
        print 'Class C inherits from A'
    def barfoo ( self ) :
        print 'barfoo from class C!'
class D ( B , C ) :
    def __init__ ( self ) :
        return
class E ( C , B ) :
    def __init__ ( self ) :
        return
```

Exercise 7.7 Write a program that takes a list of Point objects (from Section ??, page ??) and draws them with the Turtle module. Your code should be polymorphic. A Point2D object represents a point on the screen, but if you need to move the turtle to a point represented by a Point3D object, just ignore the z component.

Here is some test data to get you started:

```
square = [
    Point2D ( 0.0 , 0.0 ) , Point2D ( 100.0 , 0.0 ) , Point2D ( 100.0 , 100.0 ) ,
    Point2D ( 0.0 , 100.0 ) , Point2D ( 0.0 , 0.0 )
]
rectangle = [
    Point2D ( 0.0 , 0.0 ) , Point2D ( 200.0 , 0.0 ) , Point2D ( 200.0 , 100.0 ) ,
    Point2D ( 0.0 , 100.0 ) , Point2D ( 0.0 , 0.0 )
]
pentagon = [
    Point2D ( 0.0 , 0.0 ) , Point2D ( 100.0 , 0.0 ) , Point2D ( 131.0 , 95.0 ) ,
    Point2D ( 50.0 , 154.0 ) , Point2D ( -30.0 , 95.0 ) , Point2D ( 0.0 , 0.0 )
]
square3 = [
    Point3D ( 0.0 , 0.0 , 1.0 ) , Point3D ( 100.0 , 0.0 , 1.0 ) , Point3D ( 100.0 , 100.0 , 1.0 ) ,
    Point3D ( 0.0 , 100.0 , 1.0 ) , Point3D ( 0.0 , 0.0 , 1.0 )
```

```
]
square23 = [
    Point3D ( 0.0 , 0.0 , 1.0 ) , Point2D ( 100.0 , 0.0 ) , Point2D ( 100.0 , 100.0 ) ,
    Point3D ( 0.0 , 100.0 , 1.0 ) , Point2D ( 0.0 , 0.0 )
]
```

Hint: To draw these shapes elegantly, it would be sensible to lift the 'pen' of the turtle off the screen before moving from the origin to the first point.

Exercise 7.8 Write a version of the Caesar cipher from Section ?? (page ??) with a simple GUI. Allow users to enter text in a text box, then generate an enciphered version of the text when they press a button. Let users change the key that is used by the cipher either by providing a text box to enter the key, or a drop-down list of available keys.

You should research the HCI literature to see which of these two options the HCI community believe is a better choice of data entry in this context.

Exercise 7.9 Extend the Caesar cipher GUI by adding a button that causes deciphering (rather than enciphering) of the text in the text box.

Exercise 7.10 Write a simple color picker using Tkinter. You should provide the user with three sliders (using the Tkinter.Scale class) which they can use to set red, green and blue values. Whenever a slider is moved, a callback should be issued that changes the background color in a Canvas widget.

Exercise 7.11 Extend your answer to the previous exercise to allow users to also select specific colors from a drop-down list. Here is a random list of colors that you may want to include:

255 255 255	white	255 215 0	gold
0 0 0	black	139 69 19	SaddleBrown
47 79 79	dark slate gray	160 82 45	sienna
0 0 128	NavyBlue	205 133 63	peru
30 144 255	dodger blue	245 245 220	beige
70 130 180	SteelBlue	245 222 179	wheat
175 238 238	PaleTurquoise	244 164 96	SandyBrown
85 107 47	DarkOliveGreen	210 180 140	tan
124 252 0	LawnGreen	210 105 30	chocolate
0 255 0	green	178 34 34	firebrick
154 205 50	YellowGreen	255 140 0	dark orange
240 230 140	khaki	255 69 0	OrangeRed
255 255 224	LightYellow	255 0 0	red
255 255 0	yellow	255 105 180	hot pink

The three numbers are the red, green, and blue (RGB) values used by a screen to generate the color of the associated name name.

Challenges

Challenge 7.1 Write a GUI for the expression evaluator in Section ??. Let the user enter an expression and press a button to have that expression evaluated. To make this a bit simpler, you might want to start out with a GUI where the user enters a Python expression such as this:

```
Add(1, 2)
```

You'll need to turn the string from the user into a real Python object. Fortunately, Python provides a function called eval that does that. Here's an example of eval in action:

```
>>> class Foo :
...     def __init__ ( self ) :
...             return
...     def foo ( self ) :
...             print 'foobar!'
...
>>> eval ( 'Foo ( ).foo ( )' )
foobar!
>>>
```

Note that you can only use eval with expressions, not with statements!

Challenge 7.2 This challenge is to create a GUI that allows you to apply various image processing algorithms to images. To do this you will need an image processing library. Python doesn't come with one as standard, but you can download the Python Imaging Library (PIL) from http://www.pythonware.com/ – it is free. You will also need the PIL handbook http://www.pythonware.com/library/pil/handbook/, which is the best documentation available for the library.

The GUI should allow you to select an image, maybe using a list-box or a modal dialog from the tkFileDialog module. You will also need some buttons that apply transformations to the current image and display the processed image. To get you started, we have provided some code below that uses Tkinter to display an image, and some examples of image processing using PIL.

The *negative* of an image has the amount of red, green and blue in each image inverted. The negative of a red pixel (255, 0, 0) is a cyan pixel (0, 255, 255). The negative of a black pixel (0, 0, 0) is a white pixel (255, 255, 255). So to create the negative of an image, we just have to subtract the current red, green and blue values of each pixel from 255. Here's an example image and the code that implements the transformation:


```
""Produces the negative of a positive color image.""

__author__ = 'Sarah Mount'
__date__ = '2006-09'
__version__ = '1.0'
__copyright__ = 'Copyright (c) 2006 Sarah Mount'
__licence__ = 'GNU General Public Licence (GPL)'

import Image , sys , viewport

if __name__ == '__main__' :
    title = 'Color->Negative Using Python Image Library'
    filename = sys.argv[1]
    image = Image.open ( filename , 'r' )
    image.load ( )
    negative = image.point ( lambda pixel : 255 - pixel )
    viewport.display_image ( negative , title )
```

Two things to note here. Firstly, the PIL library stores the red, green and blue pixel values of an image as separate list elements (rather than storing a list of three-tuples). Secondly, each PIL image has a method called point that allows us to apply a transformation on each pixel value in the image. point can be used in two ways: you can either pass it a function or **lambda** expression to apply to each pixel value, or you can pass it a lookup table. Above we have used a **lambda** expression.

The PIL library also defines a number of *filters* that can be used to perform transformations on images. These are located in the ImageFilter module. To use a filter, you create an instance of a filter class and pass it to the filter method of an image object.

This example creates an *embossed* version of the image in which the edges of the objects in the image appear raised above the surface of the image:

```
    '''Emboss an image.'''

    __author__ = 'Sarah Mount'
    __date__ = '2006-09'
    __version__ = '1.0'
    __copyright__ = 'Copyright (c) 2006 Sarah Mount'
    __licence__ = 'GNU General Public Licence (GPL)'

    import Image , ImageFilter , sys , viewport

    if __name__ == '__main__' :
        filename = sys.argv[1]
        image = Image.open ( filename , 'r' )
        image = image.filter ( ImageFilter.EMBOSS ( ) )
        viewport.display_image ( image ,
                                 'Image Embossed Using Python Image Library' )
```

Below is the code for the viewport module that we have used in the examples above:

```
    '''View a PIL image on a Tkinter canvas.'''

    __author__ = 'Sarah Mount'
    __date__ = '2006-09'
    __version__ = '1.0'
    __copyright__ = 'Copyright (c) 2006 Sarah Mount'
    __licence__ = 'GNU General Public Licence (GPL)'

    import Tkinter , Image , ImageTk , sys

    def display_image ( pil_image, title ) :
        '''Take a PIL image and display it in a GUI.'''
        root = Tkinter.Tk ( )
        root.title ( title )
        im_width, im_height = pil_image.getbbox ( )[2:4]
        canvas = Tkinter.Canvas ( root, width=im_width, height=im_height )
        canvas.pack ( side=Tkinter.LEFT, fill=Tkinter.BOTH, expand=1 )
        photo = ImageTk.PhotoImage ( pil_image )
        item = canvas.create_image ( 0, 0, anchor=Tkinter.NW, image=photo )
        Tkinter.mainloop ( )

    if __name__ == '__main__' :
        filename = sys.argv[1]
        image = Image.open ( filename , 'r' )
        display_image ( image , 'Tk Viewport' )
```

8

Filing Things Away

Self-Review Questions

Self-review 8.1 We can store the contents of dictionaries, lists and tuples in a file and, using eval, retrieve them while keeping their type and structure. Does this mean that the operating system specifically supports these types?

Self-review 8.2 Why should we not use the write mode ('w') when we wish to add data to an existing file?

Self-review 8.3 What mode *should* be used when adding data to an existing file?

Self-review 8.4 If you search your computer's hard disk, you may find that many files have the same name. Readme.txt, for example, appears many times on our hard disks. These files are distinct and normally have different contents. How is it that the operating system can maintain separate files of the same name?

Self-review 8.5 What is 'wrong' with the histogram drawing algorithm on page ???

Hint: The problem is not in the code structure but in the assumption about coordinates.

The problem here is that we had assumed that the bottom left corner was the origin, (0, 0), and that positive x was rightward and positive y was upward. In fact the origin is the top left-hand corner with positive x rightward and positive y downward. So correcting the histogram drawing for a more expected appearance, we have to make two relatively trivial changes:

```
canvas.create_rectangle (
    ( 2 * i + 1 ) * barWidth , height - unitHeight ,
    ( 2 * i + 2 ) * barWidth , height - ( data[i][1] + 1 ) * unitHeight ,
    fill = 'black' )
```

Self-review 8.6 What are the special files . and .. for?

Self-review 8.7 What is the 'root' directory in a file system and what is it's .. file?

Self-review 8.8 In the os module Python has a variable called os.sep which has the value '/' on UNIX machines and '\' on Windows computers. What do you think this variable is used for?

Self-review 8.9 What are *buffers* and why does Python use them when opening files?

Self-review 8.10 What does the flush method do?

Self-review 8.11 What does the special method __repr__ do and why is it different to the __str__ method?

Self-review 8.12 What does Python's eval function do?

Self-review 8.13 What exception is raised when Python tries to open a file which does not exist?

Self-review 8.14 The check digit in ISBNs tells us whether an ISBN is valid. Can you think of any other computing related activities where check digits (or check sums) are used?

Self-review 8.15 Why do some ISBN numbers end with an X?

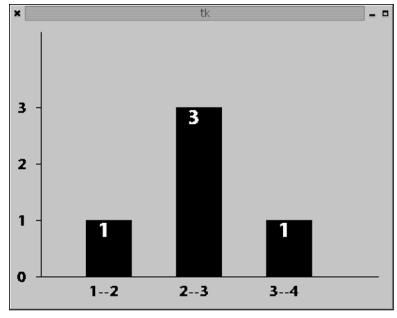
Programming Exercises

Exercise 8.1 Amend the getData method so that the parameter is a file object rather than a file name. Change the histogram drawing program removing the closing of the file and execute it to show that the amended getData method works as it should.

Exercise 8.2 Amend the getData method so that the parameter can be either a string giving the name of the file to be read, or a file object of an already open file.

Hint: Use the isinstance function to separate the three cases.

Exercise 8.3 Evolve the histogram drawing code (your corrected version from Self-review 8.5 rather than the faulty version on page ??) so that the bars have labels that show which bin the data is for, and so that there are indications of what the lengths of the bars represent. One approach would be to draw an axis down the side with tick marks and values. It may also be sensible to write the actual count of items in the bin above the bar, perhaps resulting in something like:



```
import Tkinter

def plot ( data ) :
    numberOfBuckets = len ( data )
    maximumValue = max ( [ datum[1] for datum in data ] )
    root = Tkinter.Tk ( )
    width , height = 400 , 300
    canvas = Tkinter.Canvas ( root , width = width , height = height )
    canvas.pack ( )
    originX , originY = 30 , height - 30
    canvas.create_line ( originX , originY , width - 10 , originY )
    canvas.create_line ( originX , originY , originX , 10 )
    barWidth = ( width - 2 * originX ) / ( 2 * numberOfBuckets + 1 )
    unitHeight = 300 / ( maximumValue + 2 )
    fontName = 'stonesans' # Use X names.
    axesFontDescriptor = ( fontName , 12 , 'bold' )
    numberOfTicks = 6
    for i in range ( numberOfTicks + 1 ) :
        value = i * maximumValue / numberOfTicks
        y = originY - value * unitHeight
        canvas.create_line ( originX - 5 , y , originX , y )
        canvas.create_text ( originX - 20 , y , text = str ( value ) , font = axesFontDescriptor )

    for i in range ( numberOfBuckets ) :
        x0 = ( 2 * i + 1 ) * barWidth + originX
        x1 = x0 + barWidth
        y0 = height - 30
        y1 = y0 - data[i][1] * unitHeight
        canvas.create_rectangle ( x0 , y0 , x1 , y1 , fill = 'black' )
        canvas.create_text ( x0 + 0.4 * barWidth , y0 + 15 , text = data[i][0] , font = axesFontDescriptor )
        canvas.create_text ( x0 + 0.4 * barWidth , y1 +10 , text = data[i][1] , fill = 'white' , font = ( fontNa
    root.mainloop ( )

if __name__ == '__main__' :
    plot ( [
        [ '1--2' , 1 ] ,
        [ '2--3' , 3 ] ,
        [ '3--4' , 1 ]
        ] )
```

Exercise 8.4 Look at the following program:

```
file = open ( 'words.dat' , 'w' )
word = ''
while word != 'END' :
    word = raw_input ( 'Enter a word (enter END to quit): ')
    file.write ( word + '\n' )
file.close ( )
```

This is a very simple program for storing a list of words in a file. When executed it expects the user to enter some words, one per line, finishing with the word "END".

1. What is the name of the file containing the words?
2. How many words are stored in the file?

3. Write a program to read the data file and display a numbered list of the contents, such as:

```
1: chicken
2: apple
3: fox
4: aubergine
5: END
```

4. Note that "END" is always the last word displayed. This isn't really the behavior we want, since "END" is meant as a command, not a word. Change the program so that it no longer stores "END" in the file.
5. There's still a problem: we can't put the word "END" into the file even if we wanted to. Maybe we should use the empty string ('') to signal the user's desire to quit? Modify the program to use the empty string to terminate user input.

Exercise 8.5 Using your answer to the previous question as a starting point, write a program that counts the number of times each word appears in the data file. Its output might look like this:

```
Carrot: 3
Cheese: 1
Egg: 6
Tomato: 2
```

Exercise 8.6 It would be nice if we had a graphical display of word frequency for the previous exercise. Use the histogram module from earlier in the chapter to display the word frequencies as a histogram.

Exercise 8.7 Write a program to say how many lines of text there are in a text file the name of which is obtained by prompting and reading input from the user.

Exercise 8.8 Write a program to read the contents of a text file and display it in a Tkinter text box.

Exercise 8.9 Write a program to store a list of contact names and telephone numbers, similar to the contact lists you might find on a mobile phone. The data needs to be stored in a file so that it is *persistent* – that is, the data available at the beginning of a new execution of the program is the same as at the end of the previous execution.

Exercise 8.10 Extend the program written for the previous exercise so that as new contacts are added, the program checks for duplicate names and numbers. If the new contact's name is already in the database, the message "This person is already entered." should appear. If the telephone number is already in the database, the program should display "This person's telephone number is already in the database. This could be an error, or a second contact at the same company. Add anyway?" If the user enters "Y", the contact is added.

Challenges

Challenge 8.1 Write a program to display a simple form of digital book. "Books" are text files in which each block (page) of text is followed by a double dash (--).

When a book is displayed, the first block of text is shown and the program should wait for the user to press the enter key before displaying the next.

Challenge 8.2 Extend the previous challenge so that it is possible to skip forward by an arbitrary number of pages. This should be achieved by allowing the user to enter a number before pressing the enter key. If the number is positive, the given number of pages are skipped. If there is no number, the next page is displayed.

Challenge 8.3 Further extend the book reader so that it can accept negative numbers for skipping pages. Entering -1 should go back to the previous page. There are many ways to achieve this.

9

Testing, Testing

Self-Review Questions

Self-review 9.1 What is unit testing?

It is testing the functions, classes and methods of our applications in order to ascertain whether there are bugs in the code.

Self-review 9.2 Why is unit testing important?

Without unit tests, we cannot tell whether the code we have written for our application does what it should.

Self-review 9.3 What is PyUnit?

PyUnit is the unit testing framework that comes as standard issue with the Python system.

Self-review 9.4 What is unittest?

unittest is the module name for PyUnit.

Self-review 9.5 What does this line of code mean?

```
class Fibonacci_Test ( unittest.TestCase ) :
```

It means that the class Fibonacci_Test is a subclass of unittest.TestCase ans will be a class containing unit tests to be executed as and when.

Self-review 9.6 What is an assertion?

It is a statement expressing the fact that a condition must be true.

Self-review 9.7 What does the following line mean:

```
self.assertEqual ( 55 , fibonacci ( 10 ) )
```

It is asserting that the result of executing fibonacci (10) should be the value 55. The unit test only succeeds if this is the case.

Self-review 9.8 When is a method a unit test method?

When the method is in a class that is a subclass of unittest.TestCase and the method has a name beginning with test.

Self-review 9.9 What do 'Red' and 'Green' mean in the context of unit testing?

'Red' means that the test has failed, i.e. one or more of the test methods failed. 'Green' means that all the test methods passed and the test passed.

Self-review 9.10 Why are the terms 'Red' and 'Green' used in the context of unit testing?

Because in tools originating with sUnit and JUnit, red was used to indicate a failed test and green used for a non-failing test. This tradition is continued in Eclipse tools for JUnit and TestNG with Java and PyDev with Python.

Self-review 9.11 What is 'coverage'?

It is a measure of the number of different cases associated with an application that the unit tests for that application test.

Self-review 9.12 What is 'refactoring'?

It is the process of changing the application to make is better internally without changing what it does as seen from the outside.

Self-review 9.13 Why is unit testing important for refactoring?

Without a unit test structure, it would not be possible to tell whether the behavior of the application after refactoring is the same as before.

Self-review 9.14 Using the Web, look up the term *eXtreme programming*, which has been mentioned in this chapter. What are the main features of this method and how does unit testing fit into it?

Self-review 9.15 Debugging code thoroughly usually gives rise to a few exceptions. To debug code efficiently, it's important that you understand what Python's different exception types mean. What are the following exceptions used for in Python:

- AssertionError
- ImportError
- IndexError
- KeyError
- NameError
- TypeError
- ValueError

Programming Exercises

Exercise 9.1 Although much of this chapter has been about testing, another fundamental skill is being able to identify and correct errors once your testing has uncovered them. The following pieces of code all contain simple errors. Identify the errors in each program and suggest possible tests to detect them and solutions to fix them:

1.
```
for i in range(1, -10, 1):
    print i
```

2.
```
l = [1, 2, 3, 4, 5]
l[6] = 6
```

3.
```
print [1, 2, 3].append([4, 5])
```

Hint: In most cases there will be more than one possible solution. Give some consideration to which fixes would be best to use in different situations.

Exercise 9.2 Complete the Matrix class test by adding test methods to deal with the subtraction and multiplication operations.

```
import unittest
from matrix import Matrix , zeros , unit

class Matrix_Test ( unittest.TestCase ) :

    def test_Zeros ( self ) :
        xRange = 2
        yRange = 4
        matrix = zeros ( xRange , yRange )
        for x in range ( xRange ) :
            for y in range ( yRange ) :
                self.assertEquals ( 0 , matrix[x][y] )

    def test_Unit ( self ) :
        xRange = 2
        yRange = 4
        matrix = unit ( xRange , yRange )
        for x in range ( xRange ) :
            for y in range ( yRange ) :
                if x == y :
                    self.assertEquals ( 1 , matrix[x][y] )
                else :
                    self.assertEquals ( 0 , matrix[x][y] )

    def test_AddIntegerTupleCorrect ( self ) :
        # data is a tuple of triplets, the first two are the matrices to be added, the third is the expected
        # answer.
        data = (
            ( Matrix ( ( ( 1 , 1 ) , ( 1 , 1 ) ) ) , Matrix ( ( ( 2 , 2 ) , ( 2 , 2 ) ) ) ,
              Matrix ( ( ( 3 , 3 ) , ( 3 , 3 ) ) ) ) ,
            ( Matrix ( ( ( 1 , ) , ( 1 , ) , ( 1 , ) , ( 1 , ) ) ) , Matrix ( ( ( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) ) ) ,
              Matrix ( ( ( 3 , ) , ( 3 , ) , ( 3 , ) , ( 3 , ) ) ) ) ,
            ( Matrix ( ( ( 1 , 1 , 1 , 1 ) , ) ) , Matrix ( ( ( 2 , 2 , 2 , 2 ) , ) ) ,
              Matrix ( ( ( 3 , 3 , 3 , 3 ) , ) ) )
            )
        for datum in data :
            # Addition should be commutative so try both ways round.
            self.assertEqual ( datum[2] , datum[0] + datum[1] )
            self.assertEqual ( datum[2] , datum[1] + datum[0] )

    def test_AddIntegerListCorrect ( self ) :
        # data is a tuple of triplets, the first two are the matrices to be added, the third is the expected
        # answer.
```

```
        data = (
            ( Matrix ( [ [ 1 , 1 ] , [ 1 , 1 ] ] ) , Matrix ( [ [ 2 , 2 ] , [ 2 , 2 ] ] ) ,
              Matrix ( [ [ 3 , 3 ] , [ 3 , 3 ] ] ) ) ,
            ( Matrix ( [ [ 1 ] , [ 1 ] , [ 1 ] , [ 1 ] ] ) , Matrix ( [ [ 2 ] , [ 2 ] , [ 2 ] , [ 2] ] ) ,
              Matrix ( [ [ 3 ] , [ 3 ] , [ 3 ] , [ 3 ] ] ) ) ,
            ( Matrix ( [ [ 1 , 1 , 1 , 1 ] ] ) , Matrix ( [ [ 2 , 2 , 2 , 2 ] ] ) ,
              Matrix ( [ [ 3 , 3 , 3 , 3 ] ] ) )
            )
        for datum in data :
            # Addition should be commutative so try both ways round.
            self.assertEqual ( datum[2] , datum[0] + datum[1] )
            self.assertEqual ( datum[2] , datum[1] + datum[0] )

    def test_AddDoubleTupleCorrect ( self ) :
        # data is a tuple of triplets, the first two are the matrices to be added, the third is the expe
        # answer.
        data = (
            ( Matrix ( ( ( 1.0 , 1.0 ) , ( 1.0 , 1.0 ) ) ) , Matrix ( ( ( 2.0 , 2.0 ) , ( 2.0 , 2.0 ) ) ) ,
              Matrix ( ( ( 3.0 , 3.0 ) , ( 3.0 , 3.0 ) ) ) ) ,
            ( Matrix ( ( ( 1.0 , ) , ( 1.0 , ) , ( 1.0 , ) , ( 1.0 , ) ) ) , Matrix ( ( ( 2.0 , ) , ( 2.0 , ) , ( 2.0 , ) , (
              Matrix ( ( ( 3.0 , ) , ( 3.0 , ) , ( 3.0 , ) , ( 3.0 , ) ) ) ) ,
            ( Matrix ( ( ( 1.0 , 1.0 , 1.0 , 1.0 ) , ) ) , Matrix ( ( ( 2.0 , 2.0 , 2.0 , 2.0 ) , ) ) ,
              Matrix ( ( ( 3.0 , 3.0 , 3.0 , 3.0 ) , ) ) )
            )
        for datum in data :
            # Addition should be commutative so try both ways round.
            self.assertEqual ( datum[2] , datum[0] + datum[1] )
            self.assertEqual ( datum[2] , datum[1] + datum[0] )

    def test_AddDoubleListCorrect ( self ) :
        # data is a tuple of triplets, the first two are the matrices to be added, the third is the expe
        # answer.
        data = (
            ( Matrix ( [ [ 1.0 , 1.0 ] , [ 1.0 , 1.0 ] ] ) , Matrix ( [ [ 2.0 , 2.0 ] , [ 2.0 , 2.0 ] ] ) ,
              Matrix ( [ [ 3.0 , 3.0 ] , [ 3.0 , 3.0 ] ] ) ) ,
            ( Matrix ( [ [ 1.0 ] , [ 1.0 ] , [ 1.0 ] , [ 1.0 ] ] ) , Matrix ( [ [ 2.0 ] , [ 2.0 ] , [ 2.0 ] , [ 2.0 ] ] )
              Matrix ( [ [ 3.0 ] , [ 3.0 ] , [ 3.0 ] , [ 3.0 ] ] ) ) ,
            ( Matrix ( [ [ 1.0 , 1.0 , 1.0 , 1.0 ] ] ) , Matrix ( [ [ 2.0 , 2.0 , 2.0 , 2.0 ] ] ) ,
              Matrix ( [ [ 3.0 , 3.0 , 3.0 , 3.0 ] ] ) )
            )
        for datum in data :
            # Addition should be commutative so try both ways round.
            self.assertEqual ( datum[2] , datum[0] + datum[1] )
            self.assertEqual ( datum[2] , datum[1] + datum[0] )

    def test_AddIntegerTupleError ( self ) :
        # data is tuple of pairs which should not be addable due to different shapes.
        data = (
            ( Matrix ( ( ( 1 , 1 ) , ( 1 , 1 ) ) ) , Matrix ( ( ( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) ) ) ) ,
            ( Matrix ( ( ( 1 , ) , ( 1 , ) , ( 1 , ) , ( 1 , ) ) ) , Matrix ( ( ( 2 , 2 , 2 , 2 ) , ) ) ) ,
            ( Matrix ( ( ( 1 , 1 , 1 , 1 ) , ) ) , Matrix ( ( ( 2 , 2 ) , ( 2 , 2 ) ) ) )
            )
        for datum in data :
            self.assertRaises ( ValueError , Matrix.__add__ , datum[0] , datum[1] )
```

```
    def test_AddIntegerListError ( self ) :
        # data is tuple of pairs which should not be addable due to different shapes.
        data = (
            ( Matrix ( [ [ 1 , 1 ] , [ 1 , 1 ] ] ) , Matrix ( [ [ 2 ] , [ 2 ] , [ 2 ] , [ 2 ] ] ) ) ,
            ( Matrix ( [ [ 1 ] , [ 1 ] , [ 1 ] , [ 1 ] ] ) , Matrix ( [ [ 2 , 2 , 2 , 2 ] ] ) ) ,
            ( Matrix ( [ [ 1 , 1 , 1 , 1 ] ] ) , Matrix ( [ [ 2 , 2 ] , [ 2 , 2 ] ] ) )
            )
        for datum in data :
            self.assertRaises ( ValueError , Matrix.__add__ , datum[0] , datum[1] )

    def test_AddDoubleTupleError ( self ) :
        # data is tuple of pairs which should not be addable due to different shapes.
        data = (
            ( Matrix ( ( ( 1.0 , 1.0 ) , ( 1.0 , 1.0 ) ) ) , Matrix ( ( ( 2.0 , ) , ( 2.0 , ) , ( 2.0 , ) , ( 2.0 , ) ) ) ) ,
            ( Matrix ( ( ( 1.0 , ) , ( 1.0 , ) , ( 1.0 , ) , ( 1.0 , ) ) ) , Matrix ( ( ( 2.0 , 2.0 , 2.0 , 2.0 ) , ) ) ) ,
            ( Matrix ( ( ( 1.0 , 1.0 , 1.0 , 1.0 ) , ) ) , Matrix ( ( ( 2.0 , 2.0 ) , ( 2.0 , 2.0 ) ) ) )
            )
        for datum in data :
            self.assertRaises ( ValueError , Matrix.__add__ , datum[0] , datum[1] )

    def test_AddDoubleListError ( self ) :
        # data is tuple of pairs which should not be addable due to different shapes.
        data = (
            ( Matrix ( [ [ 1.0 , 1.0 ] , [ 1.0 , 1.0 ] ] ) , Matrix ( [ [ 2.0 ] , [ 2.0 ] , [ 2.0 ] , [ 2.0 ] ] ) ) ,
            ( Matrix ( [ [ 1.0 ] , [ 1.0 ] , [ 1.0 ] , [ 1.0 ] ] ) , Matrix ( [ [ 2.0 , 2.0 , 2.0 , 2.0 ] ] ) ) ,
            ( Matrix ( [ [ 1.0 , 1.0 , 1.0 , 1.0 ] ] ) , Matrix ( [ [ 2.0 , 2.0 ] , [ 2.0 , 2.0 ] ] ) )
            )
        for datum in data :
            self.assertRaises ( ValueError , Matrix.__add__ , datum[0] , datum[1] )

# Methods added to answer question are below here ============================

    def test_SubtractCorrect ( self ) :
        data = (
            ( Matrix ( ( ( 1 , 1 ) , ( 1 , 1 ) ) ) , Matrix ( ( ( 2 , 2 ) , ( 2 , 2 ) ) ) ,
              Matrix ( ( ( -1 , -1 ) , ( -1 , -1 ) ) ) ) ,
            ( Matrix ( ( ( 1 , ) , ( 1 , ) , ( 1 , ) , ( 1 , ) ) ) , Matrix ( ( ( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) ) ) ,
              Matrix ( ( ( -1 , ) , ( -1 , ) , ( -1 , ) , ( -1 , ) ) ) ) ,
            ( Matrix ( ( ( 1 , 1 , 1 , 1 ) , ) ) , Matrix ( ( ( 2 , 2 , 2 , 2 ) , ) ) ,
              Matrix ( ( ( -1 , -1 , -1 , -1 ) , ) ) )
            )
        for datum in data :
            self.assertEqual ( datum[2] , datum[0] - datum[1] )

    def test_SubtractError ( self ) :
        data = (
            ( Matrix ( ( ( 1 , 1 ) , ( 1 , 1 ) ) ) , Matrix ( ( ( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) ) ) ) ,
            ( Matrix ( ( ( 1 , ) , ( 1 , ) , ( 1 , ) , ( 1 , ) ) ) , Matrix ( ( ( 2 , 2 , 2 , 2 ) , ) ) ) ,
            ( Matrix ( ( ( 1 , 1 , 1 , 1 ) , ) ) , Matrix ( ( ( 2 , 2 ) , ( 2 , 2 ) ) ) )
            )
        for datum in data :
            self.assertRaises ( ValueError , Matrix.__sub__ , datum[0] , datum[1] )

    def test_MultiplyCorrect ( self ) :
        data = (
```

```
                ( Matrix ( ( ( 1 , 1 ) , ( 1 , 1 ) ) ) , Matrix ( ( ( 2 , 2 ) , ( 2 , 2 ) ) ) ,
                  Matrix ( ( ( 4 , 4 ) , ( 4 , 4 ) ) ) ) ,
                ( Matrix ( ( ( 1 , ) , ( 1 , ) , ( 1 , ) , ( 1 , ) ) ) , Matrix ( ( ( 2 , 2 , 2 , 2 ) , ) ) ,
                  Matrix ( ( ( 2 , 2 , 2 , 2 ) , ( 2 , 2 , 2 , 2 ) , ( 2 , 2 , 2 , 2 ) , ( 2 , 2 , 2 , 2 ) ) ) ) ,
                ( Matrix ( ( ( 1 , 1 , 1 , 1 ) , ) ) , Matrix ( ( ( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) ) ) ,
                  Matrix ( ( ( 8 , ) , ) ) )
                )
            for datum in data :
                self.assertEqual ( datum[2] , datum[0] * datum[1] )

        def test_MultiplyError ( self ) :
            data = (
                ( Matrix ( ( ( 1 , 1 ) , ( 1 , 1 ) ) ) , Matrix ( ( ( 2 , ) , ( 2 , ) , ( 2 , ) , ( 2 , ) ) ) ) ,
                ( Matrix ( ( ( 1 , 1 , 1 , 1 ) , ) ) , Matrix ( ( ( 2 , 2 ) , ( 2 , 2 ) ) ) )
                )
            for datum in data :
                self.assertRaises ( ValueError , Matrix.__mul__ , datum[0] , datum[1] )

if __name__ == '__main__' :
    unittest.main ( )
```

Exercise 9.3 Write a test program for the queue implementation from Section ?? (page ??).

```
import sys
sys.path.append ( '../../../SourceCode/classyObjects' )

import unittest
from queue_2 import Queue

class Queue_Test ( unittest.TestCase ) :

    def setUp ( self ) :
        self.queue = Queue ( )

    def test_Create ( self ) :
        self.assertEquals ( 0 , len ( self.queue ) )

    def test_One ( self ) :
        self.queue.add ( 10 )
        self.assertEquals ( 1 , len ( self.queue ) )
        self.assertEquals ( 10 , self.queue.remove ( ) )
        self.assertEquals ( 0 , len ( self.queue ) )

    def test_Many ( self ) :
        data = ( 10 , 'fred' , 10.4 , [ 10 , 20 ] )
        for datum in data : self.queue.add ( datum )
        self.assertEquals ( len ( data ) , len ( self.queue ) )
        for i in range ( len ( data ) ) :
            self.assertEquals ( data[i] , self.queue.remove ( ) )
        self.assertEquals ( 0 , len ( self.queue ) )

    def test_RemoveEmpty ( self ) :
        self.assertRaises ( IndexError , self.queue.remove )
```

```
if __name__ == '__main__' :
    unittest.main ( )
```

Exercise 9.4 Write a test program for the stack implementation from Section ?? (page ??).

Exercise 9.5 Write a test program for the Account class from Exercise 6.6 (page 49), the CreditAccount class from Exercise 7.2 (page 56), and the StudentAccount class from Exercise 7.3 (page 56).

Exercise 9.6 Below is the start of a test suite; the exercise is to write a module that passes it. Your code will provide functions to calculate various sorts of average on lists of data:

- mode – the most frequent datum appearing in the list.
- median – the middle value in the list of (sorted) data.
- mean – the total of the data divided by the number of values.

```
import average, unittest

class TestMe ( unittest.TestCase ) :

    def setUp ( self ) :
        self.zeros = [ 0.0 for i in range ( 1 ,10 ) ]
        self.ints = [ i for i in range ( -10 , 10 ) ]
        self.foobar = [ i for i in range ( -10 , 10 ) ] + [ 2 , 2 , 2 , 2 , 2 , 2 , 2 , 2 ]

    def test_mode ( self ) :
        self.assertEquals ( 0.0 , average.mode ( self.zeros ) )
        self.assertEquals ( 0 , average.mode ( self.ints ) )
        self.assertEquals ( 2 , average.mode ( self.foobar ) )

    def test_mean ( self ) :
        self.assertEquals ( 0.0 , average.mean ( self.zeros ) )
        self.assertEquals ( -1 , average.mean ( self.ints ) )
        self.assertEquals ( 0 , average.mean ( self.foobar ) )

    def test_median ( self ) :
        self.assertEquals ( 0.0 , average.median ( self.zeros ) )
        self.assertEquals ( 0 , average.median ( self.ints ) )
        self.assertEquals ( 2 , average.median ( self.foobar ) )

if __name__ == '__main__':
    unittest.main ( )
```

Exercise 9.7 The test suite for the previous task is a good start, but not very complete. Add some more test cases. In particular, you should test for:

- Handling data that isn't in a sequence.
- Handling data that is in a sequence but isn't numeric.

Exercise 9.8 Use PyUnit to test the following functions in the random module:

- randint
- random
- choice

Exercise 9.9 If you have completed the exercise above, you will have noticed that the random module contains both functions and classes. The classes represent various sorts of random number generator, and many of the available functions are duplicated as methods in each of the classes. Write a test suite to test the randint, random and choice methods in the random.Random class.

Exercise 9.10 Some programs cannot be easily tested using unit test modules such as PyUnit. GUIs, in particular, need both unit testing (to exercise application logic) and testing that exercises the graphical components of the program.

For this exercise you need to thoroughly test the Turtle module. Although you are already familiar with this code, you should make sure that you plan your testing carefully and exercise as many of the available functions as possible. In particular, you should note that many functions in turtle are not graphical, and so *can* be tested with PyUnit.

Exercise 9.11 Some data is more interesting to test than others. UK postcodes (unlike US zip codes) are particularly interesting because they have lots of special cases. Most postcodes consist of two letters, one number, a space, a number and two letters. For example, James' office postcode is CV1 5FB. However, many postcodes are slightly different, such as PO10 0AB, SW1A 0AA and SAN TA1.

Read about British postcodes on Wikipedia http://en.wikipedia.org/wiki/UK_postcodes and write a table of test data that would thoroughly exercise a postcode parser.

Challenges

Challenge 9.1 We left these questions as questions for the reader in Section ?? (page ??):

1. Should we be stopping numbers such as IIIIIIIIIII, for example, from being legal?
2. Should we be forbidding mixed case numbers?
3. Are MCMIC, MCMXCIX and MIM all valid representations of 1999? Why does MCMXCIX appear to be the preferred representation?
4. Is MCMCXCVIII a valid representation of 2098?
5. Is MXMVII a valid representation of 1998?

1. Should we be stopping numbers such as IIIIIIIIIII, for example, from being legal?
2. Should we be forbidding mixed case numbers? (Probably yes.)
3. Are MCMIC, MCMXCIX and MIM all valid representations of 1999? Why does MCMXCIX appear to be the preferred representation?
4. Is MCMCXCVIII a valid representation of 2098?
5. Is MXMVII a valid representation of 1998?

Challenge 9.2 In this challenge you will be developing a program using Test-Driven Development (TDD), which we introduced in Section ?? (page ??). Make sure you follow the principles of TDD strictly. The purpose of this challenge is not just to give you more programming practice, but to give you experience of developing a whole program using TDD.

In Chapters 6 and 7 we developed the `Die` class and its various subclasses. For this challenge you will write a game of Snap, which should be played on the console (i.e. you don't need to write a GUI unless you want to). You will probably want to include the following components, but the detailed design of the code is up to you:

- A pack of cards. You will probably want to use a subclass of `Die` for this. You don't have to model a standard pack of 52 cards, it's up to you.
- Some part of your program should control the game. This component should be responsible for knowing whose turn it is to play and when the game has been won (or drawn if you run out of cards).
- An interface. Some component of your program should be responsible for communicating with the user. Following good HCI principles, you should make sure that the user is always aware of the state of the game (including their score).

Note that because users are playing with a pack of cards, when one card is (randomly) played, it cannot be used again. So you need some way to ensure that cards are not played twice – perhaps by keeping track of which cards have already been played. This code could be placed in various components of the program. Think about where would be most sensible to put it (but remember, you can always refactor your code later on).

Challenge 9.3 In Section ?? (page ??) of this chapter, we wrote a test class for the matrix module from Section ?? (page ??). In this challenge we will give you code for another module that uses matrices to perform some simple graphics operations. The code comes with some simple tests, your job is to test the code thoroughly using PyUnit.

In graphics applications such as the GIMP or Photoshop and in animations, shapes and objects are often moved, scaled and rotated

automatically. This is often done using a technique called homogeneous coordinates, in which 3D matrices are used to manipulate 2D images.

To convert a 2D point into a homogeneous 3D point we make the following transformation:

$$(x, y) \mapsto (x, y, 1)$$

and to convert back to an ordinary 2D point the following transformation needs to be performed:

$$(x, y, h) \mapsto \left(\frac{x}{h}, \frac{y}{h}\right)$$

To perform a transformation (rotation, translation or scaling) on a point, the point needs to be multiplied by a transformation matrix. In the code below we have created a class called HomogMatrix which is a subclass of the Matrix class you have seen before. This class has methods to convert vectors to and from their homogeneous counterparts. The module also has functions to perform transformations on vectors and code to draw shapes (made from lists of vectors) with the Python turtle.

Make sure you spend some time looking through this code in order to understand it, before you begin to write some test cases for the HomogMatrix class and the later functions.

```
""Provides functions to rotate, scale and translate points
    using homogeneous matrices.""

__author__ = 'Sarah Mount <s.mount@wlv.ac.uk>'
__date__ = '2007-05-30'
__version__ = '1.0'
__copyright__ = 'Copyright (c) 2007 Sarah Mount'
__licence__ = 'GNU General Public Licence (GPL)'

from matrix import Matrix
import copy, math, turtle

class HomogMatrix ( Matrix ) :
    def __init__ ( self , data ) :
        Matrix.__init__ ( self , data )
    def __ensure2D ( self ) :
        if len ( self.data ) != 1 or len ( self.data[0] ) != 2 :
            raise ValueError , 'Matrix not a 2D vector.'
    def __ensure3D ( self ) :
        if len ( self.data ) != 1 or len ( self.data[0] ) != 3 :
            raise ValueError , 'Matrix not a 3D vector.'
    def de_homo ( self ) :
        ""Convert a homogeneous 3D point into a 2D point.""
        self.__ensure3D ( )
        m1 = self.data[0][0] / self.data[0][2]
        m2 = self.data[0][1] / self.data[0][2]
        return HomogMatrix ( [ [ m1 , m2 ] ] )
    def en_homo ( self ) :
        ""Convert a 2D point into a homogeneous 3D point.""
```

```
            self.__ensure2D ( )
            d = copy.copy ( self.data )
            d[0] = list ( d[0] )
            d[0] += [1.0]
            return HomogMatrix ( d )
        def __mul__ ( self , m ) :
            mm = HomogMatrix ( Matrix.__mul__ ( self , m ) )
            return HomogMatrix ( mm )
        def __add__ ( self , m ) :
            mm = HomogMatrix ( Matrix.__add__ ( self , m ) )
            return HomogMatrix ( mm )
        def __sub__ ( self , m ) :
            mm = HomogMatrix ( Matrix.__sub__ ( self , m ) )
            return HomogMatrix ( mm )

    def translate ( v , ( tx , ty ) ) :
        '''Translate 2D vector v by (t1, t2)'''
        if not isinstance ( v , HomogMatrix ) :
            raise ValueError , 'Matrix not homogeneous'
        t_data = [
            [ 1.0 , 0.0 , 0.0 ] ,
            [ 0.0 , 1.0 , 0.0 ] ,
            [ tx , ty , 1.0 ] ]
        t = Matrix ( t_data )
        v_h = v.en_homo ( )
        translated = v_h * t
        return translated.de_homo ( )

    def scale ( v , ( sx , sy ) ) :
        '''Scale vector v by (sx, sy).'''
        if not isinstance ( v , HomogMatrix ) :
            raise ValueError , 'Matrix not homogeneous'
        s_data = [
            [ sx , 0.0 , 0.0 ] ,
            [ 0.0 , sy , 0.0 ] ,
            [ 0.0 , 0.0 , 1.0 ] ]
        s = HomogMatrix ( s_data )
        v_h = v.en_homo ( )
        scaled = v_h * s
        return scaled.de_homo ( )

    def rotate ( v , theta ) :
        '''Scale vector v by (sx, sy).'''
        if not isinstance ( v , HomogMatrix ) :
            raise ValueError , 'Matrix not homogeneous'
        r_data = [
            [ math.cos ( theta ) , math.sin ( theta ) , 0.0 ] ,
            [ -math.sin ( theta ) , math.cos ( theta ) , 0.0 ] ,
            [ 0.0 , 0.0 , 1.0 ] ]
        r = Matrix ( r_data )
        v_h = v.en_homo ( )
        rotated = v_h * r
        return rotated.de_homo ( )

    def draw ( points ) :
```

```
    '''Draw a shape made from a list of points.'''
    turtle.clear ( )
    turtle.up ( )
    turtle.goto ( points[0][0] )
    turtle.down ( )
    for point in points[1:]:
        turtle.goto ( point[0] )

if __name__ == '__main__':
    # Test data -- some simple shapes made from lists of points.
    # A point is a 2D vector [[x, y]]
    shapes = {
        'square' : [
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ) ,
            HomogMatrix ( [ ( 100.0 , 0.0 ) ] ) ,
            HomogMatrix ( [ ( 100.0 , 100.0 ) ] ) ,
            HomogMatrix ( [ ( 0.0 , 100.0 ) ] ) ,
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ) ] ,
        'rectangle' : [
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ) ,
            HomogMatrix ( [ ( 200.0 , 0.0 ) ] ) ,
            HomogMatrix ( [ ( 200.0 , 100.0 ) ] ) ,
            HomogMatrix ( [ ( 0.0 , 100.0 ) ] ) ,
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ) ] ,
        'pentagon' : [
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ) ,
            HomogMatrix ( [ ( 100.0 , 0.0 ) ] ) ,
            HomogMatrix ( [ ( 131.0 , 95.0 ) ] ) ,
            HomogMatrix ( [ ( 50.0 , 154.0 ) ] ) ,
            HomogMatrix ( [ ( -30.0 , 95.0 ) ] ) ,
            HomogMatrix ( [ ( 0.0 , 0.0 ) ] ) ]
        }
    for shape in shapes.keys ( ) :
        print shape , ':'
        for vector in shapes[shape] :
            print vector
        draw ( shapes[shape] )
        print

    def draw_transform ( trans , data , s , t ) :
        '''Apply matrix transformation to a shape and draw the transformed
        shape with the turtle. trans should be a function. data is used by the
        trans function. s should be a key (for the shapes dict). t should be a
        string.'''
        ss = map ( lambda v : trans ( v , data ) , shapes[s] )
        print t , s , ':'
        for v in ss : print '\t' , v
        print
        draw ( ss )

    # Draw transformed shapes
    draw_transform ( translate , ( -100.0 , -100.0 ) , 'square' , 'translated' )
    draw_transform ( translate, ( -200.0 , -200.0 ) , 'pentagon' , 'translated' )
    draw_transform ( scale, ( 1.5 , 0.2 ) , 'rectangle' , 'scaled' )
    draw_transform ( scale, ( 0.25 , 0.25 ) , 'pentagon' , 'scaled' )
```

```
draw_transform ( rotate , 45.0 , 'square' , 'rotated' )
draw_transform ( rotate , 20.0 , 'pentagon' , 'rotated' )

raw_input ( 'Press any key to exit.' )
```

10 Algorithms and Data Structures

Self-Review Questions

Self-review 10.1 Why are data structures other than sequence and dictionary useful?

Because they have properties that are different and many algorithms need data structures that have properties different to those of sequence and dictionary.

Self-review 10.2 Why is Binary Search better than Linear Search?

Because the performance of Binary Search is $O(\log_2 n)$ whereas the performance of Linear Search is $O(n)$.

Self-review 10.3 Why is it not always possible to use Binary Search for all searches in sequences?

Because the Binary Search algorithm requires the data in the sequence to be sorted.

Self-review 10.4 What alternatives to sequence are there for storing data to be searched?

The data could be in a dictionary, in which case the search is $O(1)$, or in a search tree in which case the performance is $O(\log_2 n)$. Dictionaries are clearly the best solution. The exception is where the data has to be held in a sorted order.

Self-review 10.5 What is a comparator?

It is a predicate (a Boolean function) that compares values and imposes a total order on the set of values.

Self-review 10.6 Why is Big O notation useful?

It gives us a tool for measure performance of algorithms and software and so gives us a tool for comparing.

Self-review 10.7 Why might we want to sort a list?

Two possible reasons:

1. The data is required to be sorted for some external reason, i.e. data has to be printed out in sorted order.
2. An algorithm requires use of a sequence rather than a dictionary, and the data has to be sorted.

Self-review 10.8 Why are 'divide and conquer' algorithms important?

The time complexity of 'divide and conquer' algorithms is invariably better than any linear algorithm. Binary Search ($O(\log_2 n)$) is better than Linear Search ($O(n)$). Quicksort and Merge Sort ($O(n \log_2 n)$) are better (at least in general) than Bubble Sort ($O(n^2)$).
Of course, 'divide and conquer' should not be used if 'direct access' ($O(1)$) methods are available. So don't use sequences if dictionaries can do the job.

Self-review 10.9 Why investigate tree data structures at all when we have sequences and dictionaries?

Dictionaries are implemented using open hash tables which give fine performance but the data is not stored in a sorted form. There are map-related algorithms that can happily suffer decreased performance for access and update because the need is for the data to be sorted. For these algorithms maps implemented using trees are very important.

Self-review 10.10 What is output when the following code is executed?

```
a = [ 1 , 2 , 3 , 4 ]
b = a
b[0] = 100
print a
```

What is output when the following code is executed?

```
a = [ 1 , 2 , 3 , 4 ]
b = a
b = a + [ 5 , 6 , 7 ]
b[0] = 100
print a
```

Explain why in the first code sequence there is a change to the list referred to by `a` when there is a change to `b`, but in the second code sequence there is not.

In the first code sequence `a` and `b` refer to the same list, i.e. the list object is shared by two variables. In the second code sequence `b` refers to a different list object.

Self-review 10.11 Is an algorithm that has time complexity $O(n)$ faster or slower that one that has time complexity $O(\log_2 n)$? How does a time complexity of $O(n \log_2 n)$ compare to them?

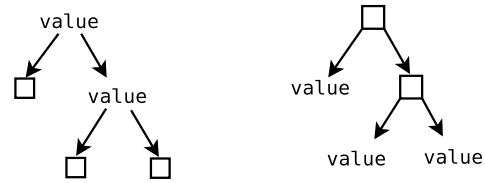
Self-review 10.12 In the recursive version of Quicksort, what are the base cases and recursive cases of the algorithm?

Self-review 10.13 When is it better to use Bubble Sort rather than Quicksort?

Self-review 10.14 The following class represents a tree structure in Python.

```
class MyTree :
    def __init__ ( self , value , left , right ) :
        self.value = value
        self.left = left
        self.right = right
```

Which of the figures below correctly represents trees constructed with this class?



Tree 1 Tree 2

Self-review 10.15 What is the *root node* of a tree data structure?

Programming Exercises

Exercise 10.1 Extend the unit test program for testing the search functions so that its coverage is far better.

Exercise 10.2 Extend the unit test program for testing the sort functions so that its coverage is far better.

Exercise 10.3 Create a quicker Quicksort by extending the implementation in this chapter so that Bubble Sort is used instead of a recursive call of Quicksort when the parameter is a sequence with less than 10 items.

Exercise 10.4 Create an improved Quicksort implementation by changing the algorithm for selecting the pivot so that the two sub-sequences created by the partition are approximately the same length.

Exercise 10.5 This exercise is about using the ideas from the binary search algorithm to play a simple "20 Questions" type game. The (human!) player thinks of a number between 0 and 100. The program asks the player a series of yes/no questions. The first question will be "Is the number between 0 and 50?", the rest follows the binary chop approach.

What is the maximum number of questions the program will have to ask the player in order to obtain the correct answer?

Exercise 10.6 Different comparators can be used to sort different kinds of data. Write comparators implementing the following comparisons and test them by using them to sort some appropriate data:

1. A lexicographical comparator – i.e. one that compares two strings based on their alphabetic ordering.
2. An ASCII comparator that can be used to sort characters based on their ASCII value. *Hint: remember the ord function!*
3. A comparator that can be used to sort names based on their surname. This comparator should use strings of the form 'Sarah Mount', 'James Shuttleworth', 'Russel Winder', and 'James T Kirk' and compare them alphabetically on the *last* word in the string.

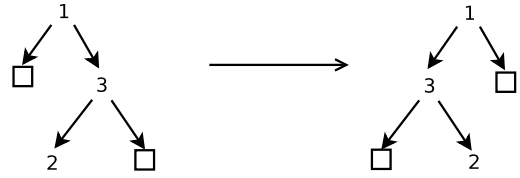
Exercise 10.7 When deciding what sorting algorithm to use on a list, it is useful to find out whether a list is nearly sorted. There are several ways to do this. Write functions which implement the following measures – if any of the functions return 0 the list is already sorted. Make sure that you compare the different functions on a range of sorted and unsorted data:

1. Return 1 if the list is unsorted.
2. Consider every adjacent pair of list elements. Add one to the 'unsortedness' score for every pair which is unsorted.
3. Find the longest sorted sub-sequence of elements in the list. Return the length of the list minus this number. For example, if the list is [2, 9, 3, 5, 6, 7, 1, 4], the longest sorted sub-sequence is 3, 5, 6, 7, which is of length 4. So your function should return 8-4 which is 4.

Exercise 10.8 In Section ?? we introduced the built-in class set which efficiently implements sets (using hash tables). This class contains many useful set operations such as intersection, union and difference, but it does not have a Cartesian product (aka cross product) operation. The Cartesian product of two sets is the set of all pairs created by taking an item from the first set and an item from the second set. For example if we have the set { 1 , 2 } and the set { 'a' , 'b' } then the Cartesian product is the set { (1 , 'a') , (1 , 'b') , (2 , 'a') , (2 , 'b') }.

For this exercise write a subclass of set that provides a method for calculating Cartesian products.

Exercise 10.9 Subclass the OrderedBinaryTree class from Section ?? to create a class that has a method reflect which exchanges the left and right subtrees of each branch, like this:



Exercise 10.10 The timeit module can be used to time the execution of code. For example, the following interpreter session shows that the built-in sort method for Python lists takes 1 s to sort the list [1 , 2 , 3 , 4 , 5] one million times on the machine we tried it on:

```
>>> import timeit
>>> t = timeit.Timer ( stmt = '[ 1 , 2 , 3 , 4 , 5 ].sort ( )' )
>>> t.timeit ( 1000000 )     # Run the statement 1000000 times.
0.99292302131652832
>>>
```

Use timeit to compare how long the various sorting algorithms covered in this chapter take to sort a given list.

Challenges

Challenge 10.1 Create a type that has the same methods as a dictionary but which uses an ordered binary tree to store the data.

Challenge 10.2 Other well-known sorting algorithms are:

Insertion Sort
Shell Sort
Heapsort
Radix Sort

Research these algorithms, extend the unit test program to test implementations of these algorithms, then add implementations of these algorithms.

11

Threading the Code

Self-Review Questions

Self-review 11.1 What is a 'thread' and what is a 'process'? What is the difference between the two?

Self-review 11.2 What does the 'scheduler' in an operating system take responsibility for?

Self-review 11.3 What does the following line of code do?

```
time.wait ( 5 )
```

Self-review 11.4 Threading is used extensively in interactive programming in applications such as games, GUIs, Web browsers, and so on. Why is threading so important for programs that need to interact with humans?

Self-review 11.5 What is happening in the following code?

```
import threading , time

class Foobar ( threading.Thread ) :
    def __init__ ( self ) :
        threading.Thread.__init__ ( self )
    def run ( self ) :
        time.sleep ( 1 )
        print self.getName ( )

if __name__ == '__main__' :
    pool = [ Foobar ( ) for i in range ( 10 ) ]
    map ( lambda t : t.start ( ) , pool )
```

Self-review 11.6 On one occasion when we ran the code above, we got the following output:

```
Thread-1
Thread-2
Thread-7
Thread-3
```

```
Thread-9
Thread-8
Thread-6
Thread-10
Thread-4
Thread-5
```

Why don't the numbers appear sequentially?

Self-review 11.7 To the nearest second, how long will the code in Self-review 11.5 take to execute?

Self-review 11.8 The second threaded guessing game program can take up to 5 s to terminate after the user has entered Ctlr+d. Why is this?

When the user presses Ctrl+d, the main thread sets the keepGoing variable in the generator thread to **False** and then terminates. The generator thread could just have tested the value and entered a sleep of up to 5 s. Only then will it see the changed value, terminate the loop, terminate the run method and hence terminate.

Self-review 11.9 'Livelock' and 'deadlock' are two problems that are commonly associated with multi-threaded programs. Define 'deadlock' and 'livelock'.

Self-review 11.10 What are 'critical sections' in programs?

Self-review 11.11 What does it mean for a piece of code to be 'atomic'?

Self-review 11.12 What does the threading.Thread.start method do?

Self-review 11.13 Why must the run method be overridden in subclasses of threading.Thread?

Self-review 11.14 What does the following code do?

```
print threading.currentThread ( )
```

What output would it give if used outside a thread object, like this:

```
if __name__ == '__main__':
    print threading.currentThread()
```

Self-review 11.15 Imagine you are writing a multi-player game. Your game will have some interesting graphics that need to be generated quickly to provide a smooth animation. There will be a sound track to the game-play and several players will be interacting at once. Inside the game there will be both player and non-player characters. What parts of this program would you want to separate out into their own threads?

Programming Exercises

Exercise 11.1 Write a program that creates two threads, each of which should sleep for a random number of seconds, then print out a message which includes its thread ID.

Exercise 11.2 Starting with your code from the previous exercise, add a counter to your program. Each thread should include the value of the counter in the message it prints to the screen and should increment the counter every time it wakes up. So, the counter will hold the total number of messages printed to the screen by all the running threads.

You will need to use some form of concurrency control to ensure that the value of the counter is always correct. Several of these are available in Python. One simple solution is to use locks. The idea here is that if a variable is locked, it cannot be changed by code in another thread until it has been unlocked. In this way we can use locks to create critical sections in our code.

Here's a code fragment to get you started:

```
import threading

foobar = 10 # Shared data!
foobar_l = threading.Lock ( )
. . .
# Start critical section
foobar_l.acquire ( ) # Acquire lock on foobar
foobar = . . .
foobar_l.release ( ) # Let other threads alter foobar
# End critical section
. . .
```

Locking isn't everyone's favorite method of concurrency control. Acquiring and releasing locks is relatively very slow – remember concurrency is often used in time-critical applications such as games.

Exercise 11.3 Python provides a special data structure in a class called `Queue.Queue` that is specifically for use in multi-threaded programs. You have already learned about queue types in Sections ?? (page ??) and ?? (page ??). In Python's queues, data can be added to a queue using a method called `put` and removed from it using the method `get`. Here's an example:

```
>>> import Queue
>>> q = Queue.Queue ( )
>>> q.put ( 5 )
>>> q.put ( 4 )
>>> q.get ( )
5
>>> q.get ( )
4
>>> q.get ( )
```

Here we have added the objects 5 and 4 to a queue we have created, then retrieved those two values. Note that we have called `q.get` a third time, even though the queue is now empty. You might have expected Python to raise an exception here, because we tried to access a value that doesn't exist. In fact, what is happening here is that Python is waiting (blocking) on the queue to provide a value. Because queues are expected to be used in concurrent programs, the authors of the `Queue.Queue` class knew that other threads might be placing data on the queue even when one thread is waiting to fetch data from it.

For this exercise, write a small program in which several threads access a single queue at the same time. Each thread should randomly either place some random data on the queue, or take data off it. Make sure you print out lots of useful information about what each thread is doing to the queue so you can watch how the program proceeds!

Challenges

Challenge 11.1 Tkinter provides a number of methods to control concurrency and scheduling in GUIs, including:

- widget.after
- widget.update
- widget.wait_visibility

Read the Tkinter documentation to find out what these methods do.

Start out by creating a simple GUI that implements an alarm. To help with this, Tkinter buttons have methods called bell and flash that 'beep' and make the button flash, respectively. Use the after method to schedule an alarm to sound after a given number of milliseconds. Make sure the user has a way of turning the alarm off!

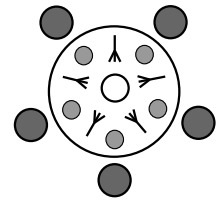
When you have a basic alarm program working, implement the following improvements:

- Allow the user to schedule the alarm to go off at a particular time.
- Allow the user to schedule the alarm to go off at given intervals (for example, every half hour).
- Allow the user to schedule several alarms to go off at various times during the day.

Challenge 11.2 The classic problem in the area of concurrent programming, was proposed by Edsgar Dijkstra in 1971, and recast by Tony Hoare as the The Dining Philosophers Problem.

Five philosophers are sitting at a round table. Philosophers are quite eccentric, all they do all day is think and eat, but they can't

do both at once. Between each pair of philosophers is a fork and each philosopher needs two forks to eat. What would happen if all the philosophers organize themselves to pick up their left fork and then their right fork whenever they stop thinking? Can you think of a better way for philosophers to eat?



Research this problem and write a program that uses threads to represent each philosopher to explore how to avoid deadlock.

12

The Life of the Game

Self-Review Questions

Self-review 12.1 Why did we create a class for this application rather than just leave it as a set of functions in a module? Is this actually needed?

Self-review 12.2 What do the terms *event*, *event handler* and *event loop* mean.

Self-review 12.3 Which are the event handler/callback methods in the Life class?

Self-review 12.4 Why have we used multiple threads in this program?

Self-review 12.5 What is the reason for changing from using a list of list of Booleans as the representation of the state of the universe?

Self-review 12.6 Why have buttons been used to present the cells of the game universe?

Self-review 12.7 What have the test programs tested?

Self-review 12.8 What have the test programs not tested?

Self-review 12.9 What was the run-time error that had to be avoided in Section ??? Why did it need to be avoided?

Self-review 12.10 Why is 'separation of concerns' a good approach to program design?

Self-review 12.11 The Game of Life has many stable patterns. In the text, we mentioned Block, Boat and Blinker. Other stable shapes are *Toad*, *Glider* and *Lightweight Spaceship* (*LWSS*). What are these shapes and why are they stable?

Hint: The answer to the question isn't in this book. You will need to research another sources.

Self-review 12.12 What are *observer variables* and how are they used in GUIs?

Self-review 12.13 What is the Tkinter 'scale' widget used for?

Self-review 12.14 What does the join method (in the Thread class) do?

Self-review 12.15 We've said that the universe which starts off with no life is stable. What would happen if we started the game with every cell being live?

Programming Exercises

Exercise 12.1 Extend the test program for the Game of Life code to include *Toad*, *Glider* and *LWSS* as part of the test.

Exercise 12.2 Extend the test program for the Game of Life code to include *Gosper's Gliding Gun* as part of the test.

Exercise 12.3 Amend the Game of Life program to use an icon rather than # as the displayed symbol.

Exercise 12.4 HighLife is a variation on Conway's Game of Life. Instead of the 23/3 rule (a live cell stays alive only if it has two or three neighbors otherwise it dies, and dead cell is made live if it has exactly three live neighbors), HighLife uses the 23/36 rule (a live cell stays alive only if it has two or three neighbors otherwise it dies, and dead cells become alive if they have either three or six live neighbors). Create a version of the Game of Life program that uses the HighLife rules.

Exercise 12.5 As presented in this chapter, the user can use the Game of Life GUI to populate the universe randomly. Add a list box which gives the user options to try out the Block, Boat and Blinker societies.

Hint: Storing these societies as lists in the program should make things straightforward.

Exercise 12.6 In the exercise above, you allowed the user to try out some preselected societies. Storing the societies as data structures in the code makes it difficult to extend the range of possibilities. For this exercise extend the program so that the societies are stored in files that are read in when the program starts. The list box of possible societies for the user to try should be calculated from the files that were read in and not predefined.

Exercise 12.7 Add the functionality to the Game of Life program of allowing users to store the *current* game state in a file, and to reload it when they next start the program.

Exercise 12.8 The Game of Life program in this chapter is an example of a program which can have multiple user interfaces – in this case, one on the console and a GUI. This exercise is to write a program that implement the Caesar Cipher from Section ?? (page ??) but which has multiple user interfaces. The program should have two interfaces to the cipher – one using the console and one which is a GUI.

Hint: Make sure that the GUI classes, the classes to control console interaction and the actual cipher are all separate. This will mean that you should be able to swap between the two interfaces and add new ones without changing the Caesar Cipher.

Challenges

Challenge 12.1 Investigate the technique of using mock objects to help test the user interface.

Challenge 12.2 Create a class called LifeCell that removes the need for dealing directly with StringVars and button construction (for the grid, anyway). That is, something that can be used to set state (cella.makeAlive, or cella.setState(**True**), perhaps) and expose an associated button for adding to the GUI (cella.getButton might be appropriate).

Challenge 12.3 Exercise 12.4 introduces the HighLife rules and a simple notation for rules for this kind of cellular automata. The standard Game of Life is described in this notation as 23/3, whilst HighLife is described as 23/36. Modify the program so that the rule is set by the user by entering two sequences of numbers into two text boxes.

When your new version of the game is working, enter the rule in the image above. Try it on a single live cell.

Challenge 12.4 Sudoku is a popular single-player puzzle game played on a 9×9 grid (see the example below). The object of the game is to fill every square with a digit from 1–9 such that each row and column contains exactly one instance of each digit, as does every 3×3 square.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

For this challenge you should implement a stand-alone GUI-based Sudoku. Sudoku puzzles are quite hard to generate so it might

be sensible to download a set of puzzles and solutions from a free online source rather than making your own in the first instance. Doing random creation of puzzles is a natural evolution of the program. From a usability point of view being able to pause a game and restart it later part solved is a good idea so storing puzzles and solutions in a sensible format that you can easily parse is an important part of the solution.

Wikipedia has a good article on Sudoku at http://en.wikipedia.org/Sudoku.

13

PyGames

Self-Review Questions

Self-review 13.1 Define the term *API*.

Self-review 13.2 What is 'blitting'? Why is it important for animation?

Self-review 13.3 What is a Rect object in PyGame? What does it represent?

Self-review 13.4 What is the difference between the Rect.move and Rect.move_ip methods in the pygame.Rect class? Give two examples of how these methods can be used in a line of code.

Self-review 13.5 Most programs that manipulate images and animations put the origin, (0, 0), of the screen at the top-left hand corner of the application window. Why is this?

Self-review 13.6 What does the pygame.init function do?

Self-review 13.7 In the Snake game, what did the following names represent and what are they used for:

clock frame SnakePart.UP fps dirty body bodysprite

Self-review 13.8 What does the pygame.Surface.convert do? Why did we use it in our code?

Self-review 13.9 What does the following code do?

```
for event in pygame.event.get ( ) :
    if event.type == pygame.locals.QUIT :
        sys.exit ( )
```

Self-review 13.10 What is the difference between a KEYDOWN and a KEYUP event?

Self-review 13.11 What names in pygame.locals are used to represent the following keys on the keyboard:

q Enter Shift Tab Space

Self-review 13.12 How can we find the current position of the mouse when using PyGame?

Self-review 13.13 What is a 'sprite' and what is a 'sprite group'?

Self-review 13.14 What does it mean to 'render' a sprite?

Self-review 13.15 What does the pygame.Sprite.update method do? Why is it usually overridden in subclasses?

Self-review 13.16 Briefly describe the technique of 'dirty rect animation'. Why is it more efficient than other methods?

Self-review 13.17 What is a 'blit buffer'? What name did we give the blit buffer in the Snake code in Section ???

Programming Exercises

Exercise 13.1 Change the bouncing ball animation so that the animation begins with the ball in a random position on the screen.

Exercise 13.2 The speed at which the ball travels will make a big difference to how *smooth* the animation looks. Allow the user to vary the speed of the animation by using the 'up' and 'down' arrow keys on the keyboard. Experiment with different speeds – which looks best?

Exercise 13.3 Add another ball to the bouncing ball animation and have them both begin in random positions.

Exercise 13.4 Having completed the previous exercise, make sure that the two balls bounce off each other whenever they collide.

Exercise 13.5 Write a simple demonstration program to show how PyGame allocates numbers (*scancodes*) to different keys on the keyboard. You should start with a blank screen, and whenever the user presses a key, display its scancode (which you can get from event.key).

Hint: Have a look at pygame.time.wait, which might be useful to you!

Exercise 13.6 Write a PyGame program to simulate an Etch A Sketch. The user should be able to use the arrow keys on the keyboard to draw a continuous line on a blank screen.

Hint: Wikipedia has a short article on Etch A Sketch at http://en.wikipedia.org/wiki/Etch_A_Sketch.

Hint: Look in the pygame.draw module. What does the aa mean in the name pygame.draw.aaline?

Exercise 13.7 The Bouncing Ball animation has almost everything necessary to create a game of single-player Pong, except a bat. Add a new subclass of Sprite to represent the bat, and make sure that the ball collides correctly with it. Use the display_score function in the game_functions module to display the score (which should increment every time the ball hits the bat).

Hint: You might want to make the ball a Sprite too, to make collision detection easier.

Exercise 13.8 Change the Snake program so that the snake only grows in length when the head collides with randomly placed sprites (snake food), which should be the same size as a SnakePart. Make sure that after a random period, new food is made available to the snake.

Exercise 13.9 Augment the game_functions module with a high score table. Use either the shelve or pickle module to store the scores.

Exercise 13.10 Add a second snake to the Snake game that is controlled by the computer. This second snake should move in random directions around the screen. If the player's snake collides with the computer's, the player loses the game. Allow the computer's snake to eat food too – preventing the player's snake from gaining valuable points!

Exercise 13.11 Add 'lives' to the Snake game. The player should lose a life if the snake collides with itself or the sides of the screen, but only lose the game when all the player's lives are exhausted. Make sure that the player is always aware of how many lives they have left.

Hint: Think about which parts of your code should go in the snake.py file and which should go in the game_functions module.

Challenges

Challenge 13.1 Implement a single-player Space Invaders arcade game (see Space Invaders on Wikipedia for a description: http://en.wikipedia.org/wiki/Space_Invaders).

Hint: You should start by creating a game with a single level. Then you can evolve this to a multi-level game, for which you will probably want to write code to use configuration files to describe what should happen on each level.

You should create any images and sounds you need to use in the game. We recommend GIMP (http://www.gimp.org/) or Inkscape (http://www.inkscape.org) for graphics.

Challenge 13.2 Python can run on a variety of devices apart from PCs. These include PDAs, embedded computers and mobile phones. Any Linux or Symbian based phone should be able to run Python: the phone

manufacturer's support site should have details. We used a Nokia Series 60 phone for our solution to this challenge, having downloaded a port of the Python interpreter from the Nokia website. This port of Python comes with some good documentation which should be read before trying to load the Python system and run Python programs on your phone.

For this challenge, write a single-player Pong game for your phone. Make sure you read the documentation carefully. In particular, you will need to know about the canvas (in `appuifw.Canvas`), the `Keyboard` class, the `Image` class and (of course!) rects. You will also want to read about the `e32.ao_yield` function which provides a very simple form of concurrency control.