```
define turnright:
    turnleft
    turnleft
    turnleft

move
turnright
move
turnoff
```

```
define putbeeper_and_move:
    putbeeper
    move

putbeeper_and_move
putbeeper_and_move
putbeeper_and_move
putbeeper_and_move
putbeeper_and_move
turnoff
```

```
define putbeeper_and_move:
    putbeeper
    move

do 5:
    putbeeper_and_move

turnoff
```

```
define putbeeper_and_move:
    putbeeper
    move

define put_down_line_of_beepers:
    do 5:
        putbeeper_and_move

put_down_line_of_beepers
turnoff
```

```
define move_if_not_blocked:
    if front_is_clear:
        move

move_if_not_blocked
turnoff
```

```
define pick_up_and_move:
    if next_to_a_beeper:
        pickbeeper
    move

pick_up_and_move
turnoff
```

```
define turnright:
    turnleft
    turnleft
    turnleft

if next_to_a_beeper:
    turnleft
else:
    turnright

turnoff
```

```
define turnright:
    turnleft
    turnleft
    turnleft

define face_north:
    if facing_east:
        turnleft
    elif facing_south:
        turnleft
        turnleft
    elif facing_west:
        turnright

face_north
turnoff
```

```
define face_north:
    do 3:
        if not_facing_north:
            turnleft

face_north
turnoff
```

```
define empty_beeper_bag:
    while any_beepers_in_beeper_bag:
        putbeeper

empty_beeper_bag
move
turnoff
```

```
define move_to_beeper:
    while not_next_to_a_beeper:
        move

move_to_beeper
pickbeeper
move
turnoff
```

```
define pick_up_all_beepers:
    while next_to_a_beeper:
        pickbeeper

pick_up_all_beepers
turnoff
```

```
define pick_up_all_beepers:
    while next_to_a_beeper:
        pickbeeper

define clean_mile:
    do 8:
        pick_up_all_beepers
        move

clean_mile
turnoff
```

```
define pick_up_all_beepers:
    while next_to_a_beeper:
        pickbeeper

define clean_intersection:
    pick_up_all_beepers
    move

define clean_mile:
    do 8:
        clean_intersection

clean_mile
turnoff
```

```
# Demonstrate basic use of Python variables and values

# The # symbol creates a comment
# Everything on a line after a # will be ignored by Python
# Use comments to document and explain your programs


#--- Basic variables

# A variable is a name that is associated with a particular value
# We use variables to represent and manipulate data in programs

# An assignment statement creates a variable and gives it a value

# The = symbol is used to perform assignment
# The left side of the = is the variable's name
# The right side is the value assigned to the variable

# Python automatically creates variables when you use
# them in assignments

# Create a variable named x and assign it the value 5
x = 5

# Display the value of x on the screen
print x

# Create a variable named y and assign it the value 10
y = 10

# Print both x and y to the screen
print x, y

# We can change the value assigned to variable by using
# another assignment statement
# After this line executes, x will hold the value 0
x = 0

# We can assign the value of one variable to another variable
# The next line assigns the value of x (currently 0) to y
y = x

# Print both x and y a second time
print x, y

# Variable names can be longer than single letters
# They must start with a lowercase letter
# Multiple words in a name should be separated by the _ character
circle_radius = 4
boiling_point_in_degrees_F = 212.0
boiling_point_in_degrees_C = 100.0
```

```
# Using descriptive variable names is good programming practice


#--- More on values

# There are three basic value types we use in Python programs

# ints (short for integers) are the positive and negative whole numbers
# Any number without a decimal point is an int
a = 4
b = -10000001

# floats are fractional numbers
# Any number with a decimal point is a float
c = .5
d = 3.14
e = -.0003

# Note that floats and ints are different types, even when they
# would be mathematically equivalent
f = 2
g = 2.0

# The name float comes from "floating point" the name for the
# technique used to represent decimal numbers in the computer's memory

# Finally, strings are sequences of characters
# A string is enclosed in quotation marks

# This line prints the word Hello to the screen
print "Hello"

# The quotation marks are not part of the string
# They just mark its beginning and ending

# We can assign strings to variables the same way we assign
# numbers to variables
example_message = "Hello"

# Printing the variable example_message will print the word Hello
print example_message

# The characters in a string can be letters, numbers, and punctuation
longer_message = "Hello, world!  I'm pleased to meet you."
print longer_message
```

```
#--- Demonstrate the use of arithmetic in Python

# There are four basic arithmetic operations in Python
#     Addition using the + symbol
#     Subtraction using the - symbol
#     Multiplication using the * symbol
#     Division using the / symbol

# All of these behave the way you would expect, except for
# division, which has some quirks we'll study in the lab

# Create two variables and print their sum
a = 10
b = 5
print a + b

# The result of an arithmetic operation can be used on the right
# side of an assignment statement
c = a + b
print c

# We can mix variables, values, and multiple operations in
# one assignment statement
d = 2 * a + c
print d

# We can use parentheses to group operations
e = (2 * 3 + 5) * 10 + 6
print e

# Python uses the standard order of operations
#     Expressions within parentheses have higher priority
#     Division and multiplication from left to right
#     Then addition and subtraction from left to right
f = (5 + 10 * (8  - 5)) * 2
print f
```

```
#--- Temperature conversion example

# Create a variable representing a Celcius temperature
temp_in_degrees_C = 100.0

# Perform the calculations to convert to Fahrenheit
temp_in_degrees_F = 1.8 * temp_in_degrees_C + 32

print temp_in_degrees_C, temp_in_degrees_F


# This program works, but is not very user-friendly
#    It only prints two numbers, with no labels or context
#    Using a different input requires changing the code

# We'll work on correcting these problems in lab
```

```
#--- Getting input from the user

# Python includes a built-in function called raw_input,
# which can be used to get input from the user

# The following lines ask the user to type something, then
# echo it back
response = raw_input("Type something and press ENTER. ")
print response

# When you run this code, the prompt
#
#    Type something and press ENTER.
#
# will be displayed on the screen
#
# The user types a message, then presses ENTER
# The message the user typed is stored in the variable response

# raw_input is our first example of a built-in function
# Python contains many functions that can be used for common tasks
# Later, we'll learn how to write our own custom functions

# We "call" functions, sometimes passing in "arguments" that control what
# what the function does

# The call to the raw_input function has four parts
#    the name of the function, raw_input
#    a left parenthesis, (
#    a string containing the message we want to display to the user
#    a right parenthesis, )

# Functions can return results, such as the response typed by the user
# The response is stored in a variable using an assignment statement

# Changing the string argument will print a different prompt
user_name = raw_input("What's your name? ")
print "Pleased to meet you!"


#--- More complex printing

# We often want to print a message that combines a message with a variable's value

# There are multiple ways to do this, but an easy way is to use the + symbol
# to combine the two strings

user_name = raw_input("What's your name? ")
print "Pleased to meet you, " + user_name

# The + operation joins together the message "Pleased to meet you, " and the
```

```
# variable containing the user's name


#--- Getting numbers as input

# By default, the response the user types is a string

# In many cases, we want the user to input a number, so that we can
# use it for a calculation

# The problem: the value returned by raw_input is a string, and we can
# only perform calculations on ints and floats

# We need to change it to a number before we can use it in a calculation
# This is called "type conversion"

# Python provides a function called int() that can take a string of numbers as
# its input and return the corresponding integer value

# Convert the string "1234" to the number 1234
int_value = int("1234")

# To convert the user's input to an integer value we get a response using raw_input,
# then use the int function to convert the string to an integer
user_string = raw_input("Type an integer value and press ENTER. ")
user_int = int(user_string)
user_int_plus_one = user_int + 1

# There is another type conversion function called float() that will convert
# a string of numbers into a float

# Finally, a function called str() will take a number as input and convert it into
# a string.  This is useful for printing.

# Convert the number 6789 into a string and print the string
int_value = 6789
str_value = str(int_value)
print "The number is " + str_value

# These statements can be combined into one line
print "The number is " + str(int_value)
```

```
#--- An introduction to Python's for loop

#--- Example 1: printing a message multiple times

# The basic for loop
for i in range(5):
    print "Hello"

# Changing the argument to range() changes the number of
# times the loop body executes
for i in range(10):
    print "Hello"

# The loop body can contain multiple statements
#
# All of the statements in the loop body execute on each
# pass through the loop
for i in range(10):
    print "Hello"
    print "Friend"


#--- Example 2: using the index variable

# The variable i is called the "index variable"
# Its value increases by 1 on each pass through the loop

# Print the index variable on each step
for i in range(5):
    print i

# Loops using range(N) -- where N is an integer -- start at
# 0 and go up to N-1

# The loop prints the values from 0 to 4

# It's common to use a single letter for the index variable -- like
# i, j, or k -- but it can have any name


#--- Example 3: using a loop to find a sum

total = 0
for k in range(5):
    total = total + k
print total


#--- Example 4: using another variable to control the loop

n = 100
```

```
total = 0
for j in range(n):
  total = total + j

# This loop will run 100 times, as if we had used range(100)
```

```
#--- Implement the Babylonian square root algorithm

# The Babylonian method is an ancient technique for estimating
# square roots

# It is an example of an iterative algorithm

# The method has two inputs
#     the number n that we wish to take the root of
#     an initial guess of the root, called x

# Each iteration updates the guess x with a better estimate

# The update calculation is
#
#    x_new = (x + n / x) / 2

# In this implementation, we'll use a for loop to repeat the
# update for a fixed number of iterations

# A more sophisticated version would examine the change in x
# at each step and stop when x has converged to a stable value


#--- Implementation

input_str = raw_input("Enter the number for the square root calculation. ")
n = int(input_str)

# The initial guess -- we'll just start at 1.0
# Making the intial value of x a float ensures that all results will be floats
x = 1.0

# Perform a fixed number of updates to x
for i in range(25):
    x = (x + n/x) / 2

# Print the final result
print "The estimate of the square root is " + str(x)
```

```
#--- Relational operators make comparisons between values

# There are six basic relational operators
#     ==  (equal)
#     !=  (not equal)
#     <
#     >
#     <=  (less than or equal)
#     >=  (greater than or equal)

# Relational comparison yield one of two special values
#     True
#     False

# These values represent logical truth or falsity

# They are not the same as the strings "True" and "False"


#--- Example: comparisons between two variables

a = 0
b = 1
print a == b
print a != b
print a < b
print a > b
print a <= b
print a >= b
```

```
#--- Introduction to Conditional Statements

# Conditional statements selectively execute code
# The body of the statement executes if a condition is True


#--- Example 1: basic if statement
a = 0
b = 0
if a == b:
    print "a equal to b"

if a <= b:
    print "a less than or equal to b"

# If the condition is not True, the body does not execute
if a > b:
    print "a greater than b"


#--- Example 2: if-else statement
if a != b:
    print "a not equal to b"
else:
    print "a equal to b"


#--- Example 3: if-elif-else statement

# The elif keyword allows us to create conditionals
# with more than three cases

# Test if a value is greater than, less than, or equal to zero
if a > 0:
    print "greater than zero"
elif a < 0:
    print "less than zero"
else:
    print "equal to zero"
```

```
#--- Testing if a number is even or odd

# The % symbol is the modulus operator
# It returns the remainder after division

# This example will print 2, because 2 is the remainder after
# 5 is divided by 3
x = 5 % 3
print x

# Prints 1
x = 65 % 8
print x

# Prints 0, 100 is evenly divisible by 10
x = 100 % 10
print x


#--- Test if a user-supplied number is even or odd
input_str = raw_input("Enter a number. ")
a = int(input_str)

if a % 2 == 0:
    print "Even"
else:
    print "Odd"


#--- Example: find the sum of all the even numbers less than 1000

total = 0

for i in range(1000):
    if i % 2 == 0:
        total = total + i

print "The sum of the even numbers less than 1000 is " + str(total)
```

```
#--- More complex logical operations

# Sometimes we want to perform tests that combine multiple relations
#
#     Is a number even and greater than 100?
#     Is a number divisible by either 2 or 3?

# There are three logical operators that can be used to combine multiple
# relational operations into more complex tests
#
#     and
#     or
#     not

# Test if a number is even and greater than 100
a = 102
if (a % 2 == 0) and (a > 100):
    print "Even and greater than 100."

# Test if a number is divisible by 2 or by 3
a = 8
if (a % 2 == 0) or (a % 3 == 0):
    print "Divisible by either 2 or 3."

# These operations behave the way you would expect
#
#     and is True only if both of its inputs are True
#     or is True if at least one of its inputs is True

# The not operator inverts the result of a relational test
a = 1
if not (a <= 0):
    print "Greater than zero."

# Using parentheses around the tests is not required, but is helpful


#--- Example: Sum of the numbers less than 1000 divisible by 3 but not by 5

total = 0

for i in range(1000):
    if (i % 3 == 0) and (i % 5 != 0):
        total = total + i

print total
```

```
#--- Number guessing game with a fixed number of tries

# The target value that the user will guess
target = 79

# The number of tries
n_tries = 10

# A Boolean variable to keep track of the user's success
success = False

# Print starting message
print "Can you guess a number between 1 and 100?"
print "I'll tell you if your guess is too high or too low."
print ''

# Loop for the number of tries
for n in range(n_tries):

    # Print how many tries are left
    print "You have " + str(n_tries - n) + " guesses remaining."

    # Get a guess from the user
    input_str = raw_input("Enter a guess. ")
    guess = int(input_str)

    # If the guess is correct, end the loop
    # The break command will immediately terminate a for loop
    if guess == target:
        success = True
        break

    # Otherwise, offer a hint
    elif guess > target:
        print "Your guess is too high."
    else:
        print "Your guess is too low."


# After the loop has ended, check the value of success
if success == True:
    print "Correct!"
else:
    print "Sorry!  You are out of guesses.  Please try again."
```

```
#--- Introduction to simple functions

# Define a function that prints a message
def print_message():
    print "Hello, World!"


# The program starts executing here, after the def statement

# Call the print_message() function
print_message()
```

```
#--- Print a message twice

def print_message():
    print "Hello, World!"

def print_twice():
    print_message()
    print_message()


# The program begins executing here, after the defines
print_twice()
```

```
#--- A function that simulates a die roll

# Import the bulit-in randint functions
from random import randint

def die_roll():

    # This is a docstring
    # It describes what the function does, its inputs, and outputs

    # The docstring is marked by triple quotes

    """
    Simulate a die roll

    Returns:
        a randomly chosen value between 1 and 6, inclusive
    """

    # Generate the random value
    roll = randint(1, 6)

    # The return statement sends the value of roll
    # back to the point of the original function call
    return roll


# Simluate the sum of two dice
roll_1 = die_roll()
roll_2 = die_roll()

sum_of_two_rolls = roll_1 + roll_2

print "The sum of the two dice is " + str(sum_of_two_rolls)
```

```
#--- Calculate the area of a circle given its radius

# Import Python's built-in value of pi
from math import pi

def area(r):
    """
    Calculate the area of a circle

    Args:
        r: the radius of the circle

    Returns:
        the area, pi * r * r
    """

    a = pi * r * r
    return a


circle_area = area(3.0)
print circle_area
```

```
#--- Function that calculates the area of a rectangle

def rect_area(width, height):

    """
    Calculate the area of rectangle

    Args:
        width: the width of the rectangle
        height: the height of the rectangle

    Returns:
        the area
    """

    # Simple calculations can be combined with the return
    return width * height


a = rect_area(6, 5)
print a
```

```
#--- Implementation of absolute value function

def absolute_value(x):

    """
    Calculate the absolute value of a number

    Args:
        x: the number to calculate the absolute value of

    Returns:
        x if x >= 0 or -x if x < 0
    """

    if x >= 0:
        return x
    else:
        return -x


abs_val_1 = absolute_value(50)
print abs_val_1

abs_val_2 = absolute_value(-75)
print abs_val_2
```

```
#--- Accessing individual characters in a string

# Use a for loop with the in keyword
#
# The variable c steps through each character in s

s = "Hello, World!"

for c in s:
    print c


# Example: count the vowels in a string
num_vowels = 0

for  c in s:
    if (c == "a") or (c == "e") or (c == "i") or (c == "o") or (c== "u"):
        num_vowels += 1

print num_vowels


# You can also use the in keyword to test if a character is in a string
num_vowels = 0

for c in s:
    if c in "aeiou":
        num_vowels += 1

print num_vowels
```

```
#--- Accessing individual characters using square brackets

s = "Hello, World!"

# You can also access single characters using square brackets
# The first character has index 0

first_char = s[0]
print first_char  # prints H

second_char = s[1]
print second_char  # prints e

# The function len() returns the length of the string
s_length = len(s)

# The final character is at index len(s) - 1
final_char = s[len(s) - 1]

# Attempting to access s[len(s)] would cause an error


# Counting the vowels in a string using a for loop over the indices
num_vowels = 0

for i in range(len(s)):
    if s[i] in "aeiou":
        num_vowels += 1

print num_vowels
```

```
#--- Opening and reading from a file

# Open the file called words.txt for reading
#
# The open() function takes two arguments
#     the full path to the file you want to open
#     a string describing what operations you want to perform ("r" for reading)
#
# The variable f contains a reference to the file

f = open("words.txt", "r")

for line in f:
    print line

f.close()
```

```
#--- Example word testing program

def starts_with_q_but_not_qu(w):

    """
    Return True if the word starts with "q" but not "qu"

    Args:
        w: the word to test

    Returns:
        True if w starts with "q" but not "qu", False otherwise
    """

    if w[0] == "q" and w[1] != "u":
        return True
    else:
        return False


# The program begins executing here, after the definitions

f = open("words.txt", "r")

for line in f:

    # strip() removes whitespace at the start or end of the line
    word = line.strip()

    if starts_with_q_but_not_qu(word):
        print word
```

```
#--- Example word testing program

def contains_no_vowels(w):

    """
    Return True if the word contains no vowels

    Args:
        w: the word to test

    Returns:
        True if w contains no vowels, False otherwise
    """

    # Loop through the vowels one at a time
    #
    # If any vowel is in w, return False immediately
    #
    # If we don't return in the loop, the word contains no vowels,
    # so we can return True

    for vowel in "aeiou":
        if vowel in w:
            return False

    return True


# The program begins executing here, after the definitions

f = open("words.txt", "r")

for line in f:

    # strip() removes whitespace at the start or end of the line
    word = line.strip()

    if contains_no_vowels(word):
        print word
```

```
#--- Example word testing program

def number_of_z(w):

    """
    Count the number of z's in a word

    Args:
        w: the word to count

    Returns:
        the number of z's
    """

    num_z = 0

    for c in w:
        if c == "z":
            num_z += 1

    return num_z


# The program begins executing here, after the definitions

# Use two variables to track the largest number of z's found
# so far and the word that contains those z's
#
# At the end of the for loop, best_word will be the word with the
# largest number of z's

best_count = 0
best_word = ""

f = open("words.txt", "r")

for line in f:

    # strip() removes whitespace at the start or end of the line
    word = line.strip()

    num_z = number_of_z(word)

    if num_z > best_count:
        best_count = num_z
        best_word = word

print best_word
```

```
#--- A simple while loop

count = 5

while count > 0:
    print count
    count = count - 1

print "Blastoff!"
```

```
#--- An echo program that uses a while loop

running = True

while running:

    input_str = raw_input("Type something: ")
    print input_str

    if input_str == "quit":
        running = False
```

```
#--- An infinite loop

# It's possible to write a while loop that never terminates
#
# Press Control-C on your keyboard to kill a program that's
# stuck in an infinite loop

while True:
    print "I AM GOING CRAZY!"
```

```
#--- Implement the Babylonian square root algorithm

input_str = raw_input("Enter the number for the square root calculation: ")
n = int(input_str)

x = 1.0  # The initial guess

# Use a loop to update until the change in x is small
running = True

while running:
    print x

    old_x = x
    x = (x + n/x) / 2

    if abs(x - old_x) < .0001:
        running = False

# Print the final result
print "The estimate of the square root is " + str(x)
```

```
#--- Introduction to lists


# Create a list of numbers
num_list = [1, 2, 3, 4, 5]

# The elements of a list can be of any type
string_list = ["fee", "fie", "fo", "fum"]

# A list can contain elements of different types, including other lists
mixed_list = [1, 3.14, "d", "The city of Nauvoo, Illinois", [0, 1, 2]]


# The range() function creates a list of numbers
x = range(4)  # Creates the list [0, 1, 2, 3, 4]
y = range(2, 6)  # Creates the list [2, 3, 4, 5]


# A for loop steps over the elements of a list in order
for s in string_list:
    print s

# We've secretly been using lists whenever we used a for loop
# with the range() function
#
# This loop is the same as one using range(4)
for i in x:
    print i


# As with strings, we can access individual elements using []
first_element = mixed_list[0]
second_element = mixed_list[1]
last_element = mixed_list[len(mixed_list) - 1]

# Unlike strings, we change the elements in a list using assignments
mixed_list[3] = "The city of Kingsport, Tennessee"


# There are two ways to add new elements to an existing list
#
# First, the list's append() method
#
# This method takes the existing list and adds one more element to the end

num_list.append(6)
print num_list

# Second, the + operator, which concatenates two lists
#
# This operation has two quirks
```

```
#
#     It creates a new list, so it must be used in an assignment
#     The element being appended must be a list

num_list = num_list + [7]
print num_list


# Use the del keyword to delete an element from a list
del mixed_list[1]
print mixed_list
```

```
#--- Find the minimum element of a list

def min_of_list(a):

    """
    Find and return the minimum element of a list

    Args:
        a: the input list (we assume elements of a can be compared)

    Returns:
        the minimum element of a
    """

    min_value = a[0]
    min_index = 0

    for i in range(1, len(a)):
        if a[i] < min_value:
            min_value = a[i]
            min_index = i

    return min_value



num_list = [101, 456, 782, 34, 99, 2, 8846]
min_element = min_of_list(num_list)
print min_element
```

```
#--- Create a list of random integers

from random import randint

def random_list(n, min_val, max_val):

    """
    Create and return a list of randomly generated integers

    Args:
        n: number of elements in the random list
        min_val: smallest possible random value
        max_val: maximum possible random value

    Returns:
        a: the list of randomly generated values
    """

    # Create the random list by starting with an empty list
    # and appending randomly generated ints to it

    a = []

    for i in range(n):
        a.append(randint(min_val, max_val))

    return a



rand_list = random_list(25, 0, 100)
print rand_list
```

```
#--- Timing of linear search

from time import clock

def linear_search(a, value):

    """
    Linear search a list for a given value

    Args:
        a: the list
        value: the value to search for

    Returns:
        True if the value is in the list, False otherwise

        In some applications, we want to search the list for a given
        value, then return a particular piece of data associated with
        the value if it is found.  For example, we search a phone directory
        for a particular name, then return the phone number associated with
        that name.  This code could be modified to accommodate that case.
    """

    for element in a:
        if element == value:
            return True

    return False



# Create a large list of numbers
n_elements = 1000000
a = range(n_elements)

# Number of iterations to use for timing experiment
n_iterations = 1000

# To measure the worst-case performance of linear search, we need
# to search for a value that can't be in the list
#
# This will force the search to examine every element in the list
value = -1

# Run the linear search test many times
#
# Measure the total elapsed time between the beginning and end of all the tests,
# the calculate the average elapsed time per call

start_time = clock()
```

```
for i in range(n_iterations):
    linear_search(a, value)

end_time = clock()

total_elapsed_time = end_time - start_time
avg_time_per_search = total_elapsed_time / n_iterations

print "The average time per linear search with list length %d = %f" % (n_elements, avg_time_per_sear
```

```
#--- Timing of binary search

from time import clock

def binary_search(a, value):

    """
    Binary search a list for a given value

    Args:
        a: the sorted list
        value: the value to search for

    Returns:
        True if the value is in the list, False otherwise

        In some applications, we want to search the list for a given
        value, then return a particular piece of data associated with
        the value if it is found.  For example, we search a phone directory
        for a particular name, then return the phone number associated with
        that name.  This code could be modified to accommodate that case.
    """

    upper = len(a) - 1
    lower = 0

    while upper >= lower:
        mid = (upper + lower) / 2

        if a[mid]  == value:
            return True

        elif a[mid] > value:
            upper = mid - 1

        elif a[mid] < value:
            lower = mid + 1


    return False



n_elements = 10000000
a = range(n_elements)
n_iterations = 1000
value = -1



# Run the binary search test many times
```

```
#
# Measure the total elapsed time between the beginning and end of all the tests,
# the calculate the average elapsed time per call

start_time = clock()

for i in range(n_iterations):
    binary_search(a, value)

end_time = clock()

total_elapsed_time = end_time - start_time
avg_time_per_search = total_elapsed_time / n_iterations

print "The average time per binary search with list length %d = %f" % (n_elements, avg_time_per_sear
```

```
Pseudocode for Selection Sort

Input: a list to sort
Output: the sorted list


Selection sort intially finds the smallest element and swaps it
into the first position of the list

It repeats this procedure for every position, finding the smallest
remaining element and swapping it into its appropriate position in
the sorted list

The list is sorted after all positions have been processed


for i in 0 to len(list) - 1

    min_value = list[i]
    min_index = i

    # Find the smallest remaining element in the list
    #
    # Elements at locations less than i are already in their
    # correct positions

    for j in i to len(list)
        if list[j] < smallest_value
            min_value = list[j]
            min_index = j

    swap list[i] and list[smallest_index]
```

```
#--- Example of local variables and scope in functions

def bar(x):

    x = x + 1
    print "The value of x in bar() = " + str(x)


def foo(x):

    # Change x, then call bar(), which has its own local x
    #
    # If bar() changes its local x, what is the effect on the
    # x here in foo()?

    x = x + 1
    print "The value of x in foo() before calling bar() = " + str(x)
    bar(x)
    print "The value of x in foo() after calling bar() = " + str(x)


# Create a variable called x and print it
#
# Call foo(), which contains a local variable named x
#
# If foo() changes its value of x, what effect does that
# have on the x defined here, in the main part of the program?

x = 0
print "The value of x in __main__ before calling foo() = " + str(x)
foo(x)
print "The value of x in __main__ after calling foo() = " + str(x)
```

```
#--- A recursive countdown program

def countdown(n):

    if n == 0:
        print "Blastoff!"
        return

    print n
    countdown(n-1)


countdown(5)
```

```
#--- Recursive implementation of factorial calculation

def factorial(n):

    """
    Recursively calculate n!

    Args:
        n: the input factorial value

    Returns:
        n!
    """

    if n == 1:
        return 1

    n_factorial = n * factorial(n - 1)
    return n_factorial


print factorial(5)
```

```
#--- Recursive calculation of the Fibonacci numbers

def fib(n):

    """
    Calculate the nth Fibonacci number

    Args:
        n: the number in the Fibonacci sequence to return

    Returns:
        The nth Fibonacci number
    """

    # There are two base cases, fib(1) = 1 and fib(2) = 1
    if n == 1:
        return 1

    if n == 2:
        return 2


    fib_n = fib(n - 1) + fib(n - 2)
    return fib_n



# Print the first ten Fibonacci numbers
for i in range(1, 11):
    print fib(i)
```

```
#--- Using Python's dictionaries to count letter frequencies

f = open("words.txt", "r")

counts = {}

for line in f:
    word = line.strip()

    for letter in word:

        # If the letter is not already in the dictionary
        # intialize its corresponding count to zero

        if letter not in counts:
            counts[letter] = 0

        counts[letter] += 1


# Print each dictionary entry
for letter in counts:
    print "%s: %d" % (letter, counts[letter])
```

```
#--- Implementation of Caesar cypher encryption

def caesar_encrypt(plaintext, shift):

    """
    Encrypt a message using the Caesar cypher

    Args:
        plaintext: the message string to encrypt
                   assume this consists of only uppercase letters

        shift: the integer amount to rotate the letters

    Returns:
        The enciphered message
    """

    ciphertext = ""

    for c in plaintext:

        # Convert the character to a number using ord()
        c_ord = ord(c) - ord("A")

        # Shift it by the desired amount, using the modulus to
        # wrap around the end of the alphabet
        c_ord_shift = (c_ord + shift) % 26

        # Convert back using chr() and append to the ciphertext
        ciphertext = ciphertext + chr(c_ord_shift + ord("A"))

    return ciphertext



plaintext = "ATTACKATDAWN"
ciphertext = caesar_encrypt(plaintext, 1)

print "plaintext: " + plaintext
print "ciphertext: " + ciphertext
```

```
#--- Create a window and draw a rectangle

# Import pygame and call the init() method
import pygame
pygame.init()

# Load color constants
blue = pygame.Color('blue')
white = pygame.Color('white')

# The overall window size
size = 320, 240

# Initialize the screen
screen = pygame.display.set_mode(size)

# Fill the screen with one color
screen.fill(white)

# Fill a specific rectangle with one color
#
# The tuple (100, 100, 100, 50) represents a rectangle
# with its top left corner at position (100, 100)
# a width of 150 and a height of 50
#
# Remember that position (0,0) is the upper-left corner!
screen.fill(blue, (100, 100, 150, 50))

# Call pygame.display.update() to actually
# make changes to the visible window
pygame.display.update()

# Run until the user tries to quit the application
# This is mandatory to get Pygame to work correctly with IDLE
running = True
while running:
    event = pygame.event.wait()
    if event.type == pygame.QUIT:
        running = False
pygame.quit()
```

```
#--- Flashing rectangle

# Import and initialize pygame
import pygame
pygame.init()

# Initialize a clock object
fps_clock = pygame.time.Clock()

# Color definitions
blue = pygame.Color('blue')
red = pygame.Color('red')

# Window size
size = 120, 240

# Initialize the screen
screen = pygame.display.set_mode(size)

running = True
screen_color = 0

while running:

    # Change the screen color
    if screen_color == 1:
        screen.fill(blue)
        screen_color = 0
    else:
        screen.fill(red)
        screen_color = 1

    # A more flexible way of closing the window
    #
    # pygame.event.get() returns a list of events that
    # have happened since the last time it was called.
    # Later, we'll use other events to interact with
    # the keyboard and mouse.
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Update the display to make the changes
    pygame.display.update()

    # Wait long enough to run at 10 frames per second
    # Call this after pygame.display.update()
    fps_clock.tick(10)

pygame.quit()
```

```
#--- Bouncing ball

# Import and initialize pygame
import pygame
pygame.init()

# Initialize a clock object
fps_clock = pygame.time.Clock()

# Color definitions
black = pygame.Color('black')
white = pygame.Color('white')

# Window size
size = 320, 240

# Initialize the screen
screen = pygame.display.set_mode(size)

# Ball parameters
ball_x = 100
ball_y = 100
ball_r = 10

# Move the ball by this much on each step
ball_dx = 1
ball_dy = 1

# Fill the screen with white background
screen.fill(white)

#--- Main loop
running = True
while running:

    # Clear the screen
    screen.fill(white)

    # Draw the ball
    pygame.draw.circle(screen, black, (ball_x, ball_y), ball_r)

    # Move the ball
    ball_x = ball_x + ball_dx
    ball_y = ball_y + ball_dy

    # Change the ball direction if it hits the edge
    #
    # screen.get_width() and screen.get_height() return the
    # width and height of the window
    if ball_x + ball_r >= screen.get_width() or ball_x - ball_r < 0:
        ball_dx = -ball_dx
```

```
    if ball_y + ball_r >= screen.get_height() or ball_y - ball_r < 0:
        ball_dy = -ball_dy

    # Check for events
    # The only event we care about is closing the window
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Update the display
    pygame.display.update()
    fps_clock.tick(20)

pygame.quit()
```

```
#--- Using the mouse in Pygame

# Import and initialize pygame
import pygame
pygame.init()

# This line imports the constants that define
# keyboard and mouse events
from pygame.locals import *

# FPS clock
fps_clock = pygame.time.Clock()

# Color definitions
white  = pygame.Color('white')
red  = pygame.Color('red')
blue  = pygame.Color('blue')

# Initialize the screen
size = 320, 240
screen = pygame.display.set_mode(size)
screen.fill(white)

# This application will move a colored square around with the mouse,
# and change color when the mouse button is clicked
square_x = 100
square_y = 100
square_size = 10
current_color = blue

# The main loop
running = True
while running:

    # Draw a square on the window
    screen.fill(white)
    screen.fill(current_color, (square_x, square_y, square_size, square_size))

    for event in pygame.event.get():

        if event.type == QUIT:
            running = False

        # Moving the mouse over the window creates a
        # MOUSEMOTION event.
        #
        # The coordinates of the mouse cursor are stored
        # in event.pos
        elif event.type == MOUSEMOTION:
            mouse_x, mouse_y = event.pos
```

```
            square_x = mouse_x
            square_y = mouse_y

        # Clicking the mouse creates a MOUSEBUTTONDOWN, then
        # a MOUSEBUTTONUP event
        #
        # This event triggers when the user releases the mouse button
        elif event.type == MOUSEBUTTONUP:

            if current_color == red:
                current_color = blue
            else:
                current_color = red

    # Update the display
    pygame.display.update()
    fps_clock.tick(20)

# Exit when the loop ends
pygame.quit()
```

```
#--- Moving a player around

# Create a "player" square and a "goal" square
#
# Arrow keys move the player
#
# The program ends when the player touches the goal square

import pygame
from random import randint
from pygame.locals import *
pygame.init()

# Initialize a clock object
fps_clock = pygame.time.Clock()

# Color definitions
black = pygame.Color('black')
white = pygame.Color('white')
blue = pygame.Color('blue')
red = pygame.Color('red')

# Window size
size = 640, 480

# Initialize the screen
screen = pygame.display.set_mode(size)

# Set the keyboard to accept repeats
#
# This allows multiple keypress signals to be sent
# when the user holds down a keyboard key
#
# Parameters control the rate of key events
pygame.key.set_repeat(10,10)

player_x = 100
player_y = 100
goal_x = 150
goal_y = 150
square_size = 10
step = 2

running = True
while running:

    screen.fill(white)
    screen.fill(blue, (player_x, player_y, square_size, square_size))
    screen.fill(red, (goal_x, goal_y, square_size, square_size))

    # Create two Rect objects from the player and goal square positions
```

```
    #
    # pygame.Rect.colliderect() returns True if the two Rects overlap
    #
    # If the collision test succeeds, the program loop ends

    player_rect = Rect(player_x, player_y, square_size, square_size)
    goal_rect = Rect(goal_x, goal_y, square_size, square_size)

    collide_test = pygame.Rect.colliderect(player_rect, goal_rect)

    if collide_test:
        running = False

    # Check for events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        # Key presses are signified by a KEYDOWN event
        #
        # The key that was pressed is stored in event.key
        # Look at the Pygame documentation for the complete list of keys
        #
        # Here, arrow key presses adjust the player square's location

        elif event.type == KEYDOWN:
            if event.key == K_UP:
                player_y -= step
            if event.key == K_DOWN:
                player_y += step
            if event.key == K_LEFT:
                player_x -= step
            if event.key == K_RIGHT:
                player_x += step

    # Update the display
    pygame.display.update()
    fps_clock.tick(30)

pygame.quit()
```

```
#--- Bouncing ball

class Ball(object):

    """A class representing a bouncing ball"""

    pass


# Import and initialize pygame
import pygame
pygame.init()

# Initialize a clock object
fps_clock = pygame.time.Clock()

# Color definitions
black = pygame.Color('black')
white = pygame.Color('white')

# Window size
size = 320, 240

# Initialize the screen
screen = pygame.display.set_mode(size)

# Initialize a single Ball object
ball = Ball()

# We can assign parameters to the ball using dot notation
ball.x = 100
ball.y = 100
ball.r = 10
ball.dx = 1
ball.dy = 1

# Fill the screen with white background
screen.fill(white)

#--- Main loop
running = True
while running:

    # Clear the screen
    screen.fill(white)

    # Draw the ball
    pygame.draw.circle(screen, black, (ball.x, ball.y), ball.r)

    # Move the ball
    ball.x = ball.x + ball.dx
```

```
    ball.y = ball.y + ball.dy

    # Change the ball direction if it hits the edge
    #
    # screen.get_width() and screen.get_height() return the
    # width and height of the window
    if ball.x + ball.r >= screen.get_width() or ball.x - ball.r < 0:
        ball.dx = -ball.dx
    if ball.y + ball.r >= screen.get_height() or ball.y - ball.r < 0:
        ball.dy = -ball.dy

    # Check for events
    # The only event we care about is closing the window
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Update the display
    pygame.display.update()
    fps_clock.tick(20)

pygame.quit()
```

```
#--- Bouncing ball

class Ball(object):

    """
    A class representing a bouncing ball

    Attributes:
        x, y: position of the ball's center
        dx, dy: ball velocity
        r: ball radius
    """

    # The constructor method has the special name __init__
    #
    # The constructor is used to set attributes of an object
    # at the moment of its creation
    #
    # The constructor always takes the special value self as its
    # first argument
    def __init__(self, arg_x, arg_y, arg_r, arg_dx, arg_dy):
        self.x = arg_x
        self.y = arg_y
        self.r = arg_r
        self.dx = arg_dx
        self.dy = arg_dy



# Import and initialize pygame
import pygame
pygame.init()

# Initialize a clock object
fps_clock = pygame.time.Clock()

# Color definitions
black = pygame.Color('black')
white = pygame.Color('white')

# Window size
size = 320, 240

# Initialize the screen
screen = pygame.display.set_mode(size)

# The constructor makes it easy to create multiple ball objects
ball = Ball(100, 100, 10, 1, 1)
ball2 = Ball(25, 25, 10, -2, -2)

# Fill the screen with white background
```

```
screen.fill(white)

#--- Main loop
running = True
while running:

    # Clear the screen
    screen.fill(white)

    # Draw the balls
    pygame.draw.circle(screen, black, (ball.x, ball.y), ball.r)
    pygame.draw.circle(screen, black, (ball2.x, ball2.y), ball2.r)

    # Move the balls
    ball.x = ball.x + ball.dx
    ball.y = ball.y + ball.dy
    ball2.x = ball2.x + ball2.dx
    ball2.y = ball2.y + ball2.dy

    # Change the ball direction if it hits the edge
    #
    # screen.get_width() and screen.get_height() return the
    # width and height of the window
    if ball.x + ball.r >= screen.get_width() or ball.x - ball.r < 0:
        ball.dx = -ball.dx
    if ball.y + ball.r >= screen.get_height() or ball.y - ball.r < 0:
        ball.dy = -ball.dy

    if ball2.x + ball2.r >= screen.get_width() or ball2.x - ball2.r < 0:
        ball2.dx = -ball2.dx
    if ball2.y + ball2.r >= screen.get_height() or ball2.y - ball2.r < 0:
        ball2.dy = -ball2.dy

    # Check for events
    # The only event we care about is closing the window
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Update the display
    pygame.display.update()
    fps_clock.tick(20)

pygame.quit()
```

```
#--- Bouncing ball

class Ball(object):

    """
    A class representing a bouncing ball

    Attributes:
        x, y: position of the ball's center
        dx, dy: ball velocity
        r: ball radius
    """

    # The constructor method has the special name __init__
    #
    # The constructor is used to set attributes of an object
    # at the moment of its creation
    #
    # The constructor always takes the special value self as its
    # first argument
    def __init__(self, arg_x, arg_y, arg_r, arg_dx, arg_dy):
        self.x = arg_x
        self.y = arg_y
        self.r = arg_r
        self.dx = arg_dx
        self.dy = arg_dy


def draw(b):

    """ Draw ball b to the Pygame window """
    pygame.draw.circle(screen, black, (b.x, b.y), b.r)


def move(b):

    """
    Move a ball object, changing direction at the screen edge

    Args:
        b: the ball to move

    Returns:
        Nothing
    """
    b.x = b.x + b.dx
    b.y = b.y + b.dy

    if b.x + b.r >= screen.get_width() or b.x - b.r < 0:
        b.dx = -b.dx
    if b.y + b.r >= screen.get_height() or b.y - b.r < 0:
```

```
        b.dy = -b.dy


# Import and initialize pygame
import pygame
pygame.init()

# Initialize a clock object
fps_clock = pygame.time.Clock()

# Color definitions
black = pygame.Color('black')
white = pygame.Color('white')

# Window size
size = 320, 240

# Initialize the screen
screen = pygame.display.set_mode(size)

# The constructor makes it easy to create multiple ball objects
ball = Ball(100, 100, 10, 1, 1)
ball2 = Ball(25, 25, 10, -2, -2)

# Fill the screen with white background
screen.fill(white)

#--- Main loop
running = True
while running:

    # Clear the screen
    screen.fill(white)

    # Draw the balls
    draw(ball)
    draw(ball2)

    # Move the balls and change direction at the screen edge
    move(ball)
    move(ball2)

    # Check for events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Update the display
    pygame.display.update()
    fps_clock.tick(20)
```

```
pygame.quit()
```

```
#--- Pong using classes

# Importing pygame.locals lets us use many names
# without putting "pygame." at the front
import pygame
from pygame.locals import *


class Paddle(object):

    """
    A paddle controlled by the user

    Attributes:
        x, y = the upper left corner of the paddle
        width, height = the extent of the paddle
    """

    def __init__(self, arg_x, arg_y, arg_width, arg_height):
        self.x = arg_x
        self.y = arg_y
        self.width = arg_width
        self.height = arg_height


class Ball(object):

    """
    The Pong ball

    Attributes:
        x, y: ball center
        dx, dy: ball velocity
        r: ball radius
    """

    def __init__(self, arg_x, arg_y, arg_r, arg_dx, arg_dy):
        self.x = arg_x
        self.y = arg_y
        self.r = arg_r
        self.dx = arg_dx
        self.dy = arg_dy


def draw_paddle(p):

    """ Draw rectangular paddle p to the screen """
    screen.fill(black, (p.x, p.y, p.width, p.height))


def draw_ball(b):
```

```
    """ Draw ball b to the screen """

    # Change the ball's center position to an int
    #
    # Pygame will not draw at fractional pixel positions
    b_x_int = int(round(b.x))
    b_y_int = int(round(b.y))

    pygame.draw.circle(screen, black, (b_x_int, b_y_int), b.r)


def move_ball(b):

    """
    Move a ball object, changing direction at top or bottom
    of the screen, or at the paddles

    Args:
        b: the ball to move
        left_p: the left paddle
        right_p: the right paddle

    Returns:
        Nothing
    """

    b.x = b.x + b.dx
    b.y = b.y + b.dy

    # The ball can bounce off the top or bottom of the screen
    if b.y + b.r >= screen.get_height() or b.y - b.r < 0:
        b.dy = -b.dy


def test_paddle_reflection(b, p):

    """
    Test for a reflection off of a paddle

    Args:
        b: the ball
        p: the paddle

    Returns:
        Nothing
    """

    # Create Rect objects containing the ball and paddle
    #
    # Remember, Rect() uses the upper left corner position,
```

```
    # width, and height as its arguments, so we need to calculate
    # these using the ball center and radius
    ball_rect = Rect(b.x - b.r, b.y - b.r, 2 * b.r, 2 * b.r)
    paddle_rect = Rect(p.x, p.y, p.width, p.height)

    # If there is a collision, reflect the ball
    if ball_rect.colliderect(paddle_rect):
        b.dx = -b.dx


def ball_at_edge(b):

    """
    Check if the ball has reached either side of the screen

    Args:
        b: the ball to test

    Returns:
        True if b is at left or right edge, False otherwise
    """

    if b.x - b.r <= 0 or b.x + b.r >= screen.get_width():
        return True
    else:
        return False


# Initialize pygame
pygame.init()

# Initialize a clock object
fps_clock = pygame.time.Clock()

# Color definitions
black = pygame.Color('black')
white = pygame.Color('white')

# Window size
size = 320, 240

# Initialize the screen
screen = pygame.display.set_mode(size)

# Initialize key repeat
pygame.key.set_repeat(10, 10)

# Create two paddles and a ball
left_paddle = Paddle(0, 100, 10, 40)
right_paddle = Paddle(310, 100, 10, 40)
ball = Ball(100, 100, 5, 2, 2)
```

```
#--- Main loop
running = True
while running:

    # Clear the screen
    screen.fill(white)

    # Draw the paddles and the ball
    draw_paddle(left_paddle)
    draw_paddle(right_paddle)
    draw_ball(ball)

    # Move the ball, checking for reflection
    move_ball(ball)

    # Test for ball and paddle reflections
    test_paddle_reflection(ball, left_paddle)
    test_paddle_reflection(ball, right_paddle)

    # Check to see if the ball has reached the edge of the screen
    #
    # If so, end the game
    if ball_at_edge(ball):
        running = False

    # Check for events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        # Key events move the paddles up and down
        #
        # Use up and down arrows for the right paddle
        # w and s keys for the left paddle
        #
        # Moving the paddle up DECREASES its y coordinate
        # Moving down INCREASES the y coordinate

        elif event.type == KEYDOWN:
            if event.key == K_UP:
                right_paddle.y -= 5
            if event.key == K_DOWN:
                right_paddle.y += 5
            if event.key == K_w:
                left_paddle.y -= 5
            if event.key == K_s:
                left_paddle.y += 5

    # Update the display
```

```
    pygame.display.update()
    fps_clock.tick(20)

pygame.quit()
```

```
#--- ASTEROIDS

import pygame
from pygame.locals import *
from random import random, randint
from math import sin, cos, radians


class Asteroid(object):

    """
    A floating space rock, drawn as a circle

    Attributes:
        x, y: center
        dx, dy: velocity
        r: radius
        color: color of this asteroid

    Methods:
        draw: draw the Asteroid
        move: move the Asteroid
        get_rect: helper method to return bounding rectangle
        is_large_enough_to_split: returns True if radius is not too small
        split: creates two new smaller Asteroids
    """

    def __init__(self, arg_x, arg_y, arg_r, arg_dx, arg_dy, arg_c):

        """ Initialize attributes for a new asteroid """
        self.x = arg_x
        self.y = arg_y
        self.r = arg_r
        self.dx = arg_dx
        self.dy = arg_dy
        self.color = arg_c


    def draw(self):

        """ Draw this asteroid to the screen """

        x_int = int(round(self.x))
        y_int = int(round(self.y))
        pygame.draw.circle(screen, self.color, (x_int, y_int), self.r)


    def move(self):

        """
        Move the asteroid
```

```
    If the asteroid leaves one side of the screen it will float
    back in on the other side
    """

    self.x += self.dx
    self.y += self.dy

    # Leaving the left side
    if self.x + self.r < 0:
        self.x = screen.get_width() + self.r

    # Leaving the right side
    if self.x - self.r > screen.get_width():
        self.x = 0 - self.r

    # Leaving the top
    if self.y + self.r < 0:
        self.y = screen.get_height() + self.r

    # Leaving the bottom
    if self.y - self.r > screen.get_height():
        self.y = 0 - self.r


def get_rect(self):

    """ Return a Rect containing this Asteroid """

    # Make the Rect a little smaller than the true bounding box
    # to help the player
    r = Rect(self.x - .9 * self.r, self.y - .9 * self.r,
             1.8 * self.r, 1.8 * self.r)
    return r


def is_large_enough_to_split(self):

    """ The Asteroid can split if its radius is >= 20 """
    return self.r >= 20


def split(self):

    """
    Split this asteroid into two new Asteroids

    Each split will have the same starting location and half the size
    and a randomly chosen velocity
    """
```

```
        new_dx_1 = 8 * (random() - .5)
        new_dy_1 = 8 * (random() - .5)
        ast_1 = Asteroid(self.x, self.y, self.r / 2, new_dx_1,
                         new_dy_1, self.color)

        new_dx_2 = 8 * (random() - .5)
        new_dy_2 = 8 * (random() - .5)
        ast_2 = Asteroid(self.x, self.y, self.r / 2, new_dx_2,
                         new_dy_2, self.color)

        return (ast_1, ast_2)



class Ship(object):

    """
    Our heroic vessel

    Attributes:
        x, y: position of center of the bottom of the ship triangle
        dx, dy: velocity
        theta: angle of rotation for the ship

    Methods:
        rotate: helper method to perform point rotation
        draw: calculate the orientation of the ship and draw it
        move: move the ship using its current velocity
        change_velocity: adjust the ship's velcity based on its current heading
        create_bullet: create and return a Bullet object
        get_rect: helper method to return bounding Rect
    """

    def __init__(self):

        self.x = 320
        self.y = 240
        self.dx = 0
        self.dy = 0
        self.theta = 0.0


    def rotate(self, x, y, theta):

        """
        Rotate a point

        Args:
            x: the x coordinate of the point
            y: the y coordinate of the point
            theta: the counter clockwise angle of rotation
```

```
        Returns:
            the rotated (x, y) pair
        """

        cos_theta = cos(radians(theta))
        sin_theta = sin(radians(theta))
        new_x = cos_theta * x - sin_theta * y
        new_y = sin_theta * x + cos_theta * y

        return (new_x, new_y)


    def draw(self):

        """
        Calculate the vertices of the ship triangle based on the
        (x,y) position the center of the base and theta

        Draw the triangle to the screen as a polygon
        """

        # Find the positions of the points by calculating a displacement
        # for each triangle point, then adding the displacement to the
        # ship's (x, y) position

        # Rotate the displacements counter-clockwise by an angle of theta
        # relative to the center of the bottom line of the ship
        (ll_x, ll_y) = self.rotate(-5, 0, self.theta)
        (lr_x, lr_y) = self.rotate(5, 0, self.theta)
        (tip_x, tip_y) = self.rotate(0, -10, self.theta)

        # Create the final list of points that will be drawn
        # by adding each displacement to the ship's position
        point_list = [[tip_x + self.x, tip_y + self.y],
                      [ll_x + self.x, ll_y + self.y],
                      [lr_x + self.x, lr_y + self.y]]

        # Draw the ship triangle
        pygame.draw.polygon(screen, black, point_list)


    def move(self):

        """ Move the ship forwards """

        self.x += self.dx
        self.y += self.dy

        # Leaving the left side
        if self.x < -10:
```

```
            self.x = screen.get_width() + 10

        # Leaving the right side
        if self.x > screen.get_width() + 10:
            self.x = -10

        # Leaving the top
        if self.y < -10:
            self.y = screen.get_height() + 10

        # Leaving the bottom
        if self.y > screen.get_height() + 10:
            self.y = -10


    def change_velocity(self):

        """
        Calculate a new velocity based on the present velocity
        and the ship's current heading
        """

        # Calculate the displacement of the tip of the ship
        # relative to the center of the bottom
        #
        # This give's the ship's current heading
        (heading_x, heading_y) = self.rotate(0, -10, self.theta)

        # Add the heading to the ship's current velocity
        self.dx += .1 * heading_x
        self.dy += .1 * heading_y


    def create_bullet(self):

        """ Create a new bullet object based on the ship's heading """

        (heading_x, heading_y) = self.rotate(0, -10, self.theta)

        bullet_x = heading_x + self.x
        bullet_y = heading_y + self.y
        bullet_dx = .25 * heading_x
        bullet_dy = .25 * heading_y

        new_bullet = Bullet(bullet_x, bullet_y, bullet_dx, bullet_dy)

        return new_bullet


    def get_rect(self):
```

```
        """
        Return a Rect object bounding the ship

        The Rect is made a little bit smaller than the
        true bounding rectangle so the player won't
        perceive unfairness in the hit calculation
        """

        (ll_x, ll_y) = self.rotate(-5, 0, self.theta)
        (lr_x, lr_y) = self.rotate(5, 0, self.theta)
        (tip_x, tip_y) = self.rotate(0, -10, self.theta)

        leftmost_x = min([ll_x, lr_x, tip_x, 0])
        topmost_y = min([ll_y, lr_y, tip_y, 0])

        rightmost_x = max([ll_x, lr_x, tip_x, 0])
        bottommost_y = max([ll_y, lr_y, tip_y, 0])

        rect_x = leftmost_x + self.x + 2
        rect_y = topmost_y + self.y + 2
        width = rightmost_x - leftmost_x - 2
        height = bottommost_y - topmost_y - 2

        return Rect(rect_x, rect_y, width, height)



class Bullet(object):

    """
    Bullets fired by the ship

    Attributes:
        x, y: position
        dx, dy: velocity

    Methods:
        draw: draw the bullet as a small circle
        move: move the bullet
        get_rect: helper method to return bounding rectangle
    """

    def __init__(self, arg_x, arg_y, arg_dx, arg_dy):
        self.x = arg_x
        self.y = arg_y
        self.dx = arg_dx
        self.dy = arg_dy
        self.r = 1


    def draw(self):
```

```
        """ Draw the bullet as a small circle """

        x_int = int(round(self.x))
        y_int = int(round(self.y))
        pygame.draw.circle(screen, black, (x_int, y_int), self.r)


    def move(self):

        """
        Move the bullet

        If the asteroid leaves one side of the screen it will float
        back in on the other side
        """

        self.x += self.dx
        self.y += self.dy

        # Leaving the left side
        if self.x + self.r < 0:
            self.x = screen.get_width() + self.r

        # Leaving the right side
        if self.x - self.r > screen.get_width():
            self.x = 0 - self.r

        # Leaving the top
        if self.y + self.r < 0:
            self.y = screen.get_height() + self.r

        # Leaving the bottom
        if self.y - self.r > screen.get_height():
            self.y = 0 - self.r


    def get_rect(self):

        """ Return a Rect containing this bullet """
        return Rect(self.x - self.r, self.y - self.r, 2 * self.r, 2 * self.r)


def random_asteroid():

    """
    Initialize and return an asteroid with random attributes

    Args:
        None
```

```
    Returns:
        the new Asteroid object
    """

    # Generate a random location that isn't in the middle of the screen
    #
    # This ensures that the player's ship has some space at the start
    finding_location = True
    while finding_location:
        new_x = randint(1, screen.get_width())
        new_y = randint(1, screen.get_height())

        if (new_x < 200 or new_x > 400) and (new_y < 200 or new_y > 400):
            finding_location = False

    new_r = randint(10, 90)

    # random() returns a value that is uniformly distributed
    # in [0, 1]; map this value to [-2, 2]
    new_dx = 6 * (random() - .5)
    new_dy = 6 * (random() - .5)

    # Create a random color
    new_color = pygame.Color(randint(0, 255), randint(0, 255),
                             randint(0, 255))

    # Create and return the new asteroid
    new_asteroid = Asteroid(new_x, new_y, new_r, new_dx, new_dy, new_color)
    return new_asteroid


def check_bullet_asteroid_collisions(asteroid_list, bullet_list):

    """
    Check for collisions between an asteroid and any bullet

    Args:
        asteroid_list: the list of Asteroid object
        bullet_list: the list of Bullets

    Returns:
        Nothing, but asteroid_list and bullet_list may change
    """

    # Loop over the contents of the asteroid list, checking for
    # intersections with each bullet
    #
    # If a collision occurs, split the asteroid if it's large
    # enough to split, then delete the bullet and the asteroid from
    # their respective lists
```

```
    ast_ix = 0

    while ast_ix < len(asteroid_list):
        delete_this_asteroid = False
        ast_rect = asteroid_list[ast_ix].get_rect()
        bullet_ix = 0

        while bullet_ix < len(bullet_list):
            bullet_rect = bullet_list[bullet_ix].get_rect()

            if pygame.Rect.colliderect(bullet_rect, ast_rect):

                if asteroid_list[ast_ix].is_large_enough_to_split():
                    (new_ast_1, new_ast_2) = asteroid_list[ast_ix].split()
                    asteroid_list.append(new_ast_1)
                    asteroid_list.append(new_ast_2)

                del bullet_list[bullet_ix]
                delete_this_asteroid = True

            else:
                bullet_ix += 1

        if delete_this_asteroid:
            del asteroid_list[ast_ix]
        else:
            ast_ix += 1


def check_ship_asteroid_collisions(asteroid_list, ship):

    """
    Test for ship/asteroid collisions

    Args:
        asteroid_list: the list of Asteroids
        ship: the Ship object

    Returns:
        True if a collision occurs, False otherwise
    """

    ship_rect = ship.get_rect()

    for asteroid in asteroid_list:
        ast_rect = asteroid.get_rect()

        if pygame.Rect.colliderect(ship_rect, ast_rect):
            return True

    return False
```

```
#--- Initialization

# Initialize pygame
pygame.init()

# Initialize a clock object
fps_clock = pygame.time.Clock()

# Color definitions
black = pygame.Color('black')
white = pygame.Color('white')

# Window size
size = 640, 480

# Initialize the screen
screen = pygame.display.set_mode(size)

# Initialize key repeat
pygame.key.set_repeat(10, 10)

# Create the ship
ship = Ship()

# Keep the asteroids in a list
asteroid_list = []

# Initialize some random asteroids
for i in range(15):
    asteroid_list.append(random_asteroid())

# Keep the bullets in a list
bullet_list = []


#--- Main loop
running = True
while running:

    # Clear the screen
    screen.fill(white)

    # Move and draw each asteroid
    for ast in asteroid_list:
        ast.move()
        ast.draw()

    # Move and draw bullets
    for bullet in bullet_list:
```

```
            bullet.move()
            bullet.draw()

        # Move and draw the ship
        ship.move()
        ship.draw()

        # Check for collisions
        check_bullet_asteroid_collisions(asteroid_list, bullet_list)

        if check_ship_asteroid_collisions(asteroid_list, ship):
            running = False


        # Check for events
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

            if event.type == KEYDOWN:
                if event.key == K_UP:
                    ship.change_velocity()
                if event.key == K_LEFT:
                    ship.theta -= 10
                if event.key == K_RIGHT:
                    ship.theta += 10
                if event.key == K_SPACE:
                    if len(bullet_list) < 20:
                        new_bullet = ship.create_bullet()
                        bullet_list.append(new_bullet)


        # Update the display
        pygame.display.update()
        fps_clock.tick(20)

    pygame.quit()
```