1 Exercises

Use a web browser to go to the Python website `http://python.org`. This page contains information about Python and links to Python-related pages, and it gives you the ability to search the Python documentation.

For example, if you enter `print` in the search window, the first link that appears is the documentation of the `print` statement. At this point, not all of it will make sense to you, but it is good to know where it is.

Start the Python interpreter and type `help()` to start the online help utility. Or you can type `help('print')` to get information about the `print` statement.

If this example doesn't work, you may need to install additional Python documentation or set an environment variable; the details depend on your operating system and version of Python.

Start the Python interpreter and use it as a calculator. Python's syntax for math operations is almost the same as standard mathematical notation. For example, the symbols `+`, `-` and `/` denote addition, subtraction and division, as you would expect. The symbol for multiplication is `*`.

If you run a 10 kilometer race in 43 minutes 30 seconds, what is your average time per mile? What is your average speed in miles per hour? (Hint: there are 1.61 kilometers in a mile).

2 Exercises

Assume that we execute the following assignment statements:

```
width = 17
height = 12.0
delimiter = '.'
```

For each of the following expressions, write the value of the expression and the type (of the value of the expression).

1. `width/2`
2. `width/2.0`
3. `height/3`
4. `1 + 2 * 5`
5. `delimiter * 5`

Use the Python interpreter to check your answers. Practice using the Python interpreter as a calculator:

1. The volume of a sphere with radius r is $\frac{4}{3}\pi r^3$. What is the volume of a sphere with radius 5? Hint: 392.7 is wrong!

2. Suppose the cover price of a book is $24.95, but bookstores get a 40% discount. Shipping costs $3 for the first copy and 75 cents for each additional copy. What is the total wholesale cost for 60 copies?

3. If I leave my house at 6:52 am and run 1 mile at an easy pace (8:15 per mile), then 3 miles at tempo (7:12 per mile) and 1 mile at easy pace again, what time do I get home for breakfast?

3 Exercises

Python provides a built-in function called `len` that returns the length of a string, so the value of `len('allen')` is 5. Write a function named `right_justify` that takes a string named `s` as a parameter and prints the string with enough leading spaces so that the last letter of the string is in column 70 of the display.

```
>>> right_justify('allen')
                                                                allen
```

A function object is a value you can assign to a variable or pass as an argument. For example, `do_twice` is a function that takes a function object as an argument and calls it twice:

```
def do_twice(f):
    f()
    f()
```

Here's an example that uses `do_twice` to call a function named `print_spam` twice.

```
def print_spam():
    print 'spam'
do_twice(print_spam)
```

1. Type this example into a script and test it.

2. Modify `do_twice` so that it takes two arguments, a function object and a value, and calls the function twice, passing the value as an argument.

3. Write a more general version of `print_spam`, called `print_twice`, that takes a string as a parameter and prints it twice.

4. Use the modified version of `do_twice` to call `print_twice` twice, passing `'spam'` as an argument.

5. Define a new function called `do_four` that takes a function object and a value and calls the function four times, passing the value as a parameter. There should be only two statements in the body of this function, not four.

Solution: `http://thinkpython.com/code/do_four.py`. This exercise can be done using only the statements and other features we have learned so far.

1. Write a function that draws a grid like the following:

```
+ - - - - + - - - - +
|         |         |
|         |         |
|         |         |
|         |         |
|         |         |
+ - - - - + - - - - +
|         |         |
|         |         |
|         |         |
|         |         |
|         |         |
+ - - - - + - - - - +
```

   Hint: to print more than one value on a line, you can print a comma-separated sequence:

```
print '+', '-'
```

   If the sequence ends with a comma, Python leaves the line unfinished, so the value printed next appears on the same line.

```
print '+',
print '-'
```

   The output of these statements is `'+ -'`. A `print` statement all by itself ends the current line and goes to the next line.

2. Write a function that draws a similar grid with four rows and four columns.

Solution: `http://thinkpython.com/code/grid.py`. Credit: This exercise is based on an exercise in Oualline, *Practical C Programming, Third Edition*, O'Reilly Media, 1997.

4 Exercises

The following are the possible hands in poker, in increasing order of value (and decreasing order of probability):

pair: two cards with the same rank

two pair: two pairs of cards with the same rank

three of a kind: three cards with the same rank

straight: five cards with ranks in sequence (aces can be high or low, so `Ace-2-3-4-5` is a straight and so is `10-Jack-Queen-King-Ace`, but `Queen-King-Ace-2-3` is not.)

flush: five cards with the same suit

full house: three cards with one rank, two cards with another

four of a kind: four cards with the same rank

straight flush: five cards in sequence (as defined above) and with the same suit

The goal of these exercises is to estimate the probability of drawing these various hands.

1. Download the following files from `http://thinkpython.com/code`:

   `Card.py` : A complete version of the `Card`, `Deck` and `Hand` classes in this chapter.

   `PokerHand.py` : An incomplete implementation of a class that represents a poker hand, and some code that tests it.

2. If you run `PokerHand.py`, it deals seven 7-card poker hands and checks to see if any of them contains a flush. Read this code carefully before you go on.

3. Add methods to `PokerHand.py` named `has_pair`, `has_twopair`, etc. that return True or False according to whether or not the hand meets the relevant criteria. Your code should work correctly for "hands" that contain any number of cards (although 5 and 7 are the most common sizes).

4. Write a method named `classify` that figures out the highest-value classification for a hand and sets the `label` attribute accordingly. For example, a 7-card hand might contain a flush and a pair; it should be labeled "flush".

5. When you are convinced that your classification methods are working, the next step is to estimate the probabilities of the various hands. Write a function in `PokerHand.py` that shuffles a deck of cards, divides it into hands, classifies the hands, and counts the number of times various classifications appear.

6. Print a table of the classifications and their probabilities. Run your program with larger and larger numbers of hands until the output values converge to a reasonable degree of accuracy. Compare your results to the values at `http://en.wikipedia.org/wiki/Hand_rankings`.

Solution: `http://thinkpython.com/code/PokerHandSoln.py`.

This exercise uses TurtleWorld from Chapter **??**. You will write code that makes Turtles play tag. If you are not familiar with the rules of tag, see `http://en.wikipedia.org/wiki/Tag_(game)`.

1. Download `http://thinkpython.com/code/Wobbler.py` and run it. You should see a TurtleWorld with three Turtles. If you press the Run button, the Turtles wander at random.

2. Read the code and make sure you understand how it works. The `Wobbler` class inherits from `Turtle`, which means that the `Turtle` methods `lt`, `rt`, `fd` and `bk` work on Wobblers.

   The `step` method gets invoked by TurtleWorld. It invokes `steer`, which turns the Turtle in the desired direction, `wobble`, which makes a random turn in proportion to the Turtle's clumsiness, and `move`, which moves forward a few pixels, depending on the Turtle's speed.

3. Create a file named `Tagger.py`. Import everything from `Wobbler`, then define a class named `Tagger` that inherits from `Wobbler`. Call `make_world` passing the `Tagger` class object as an argument.

4. Add a `steer` method to `Tagger` to override the one in `Wobbler`. As a starting place, write a version that always points the Turtle toward the origin. Hint: use the math function `atan2` and the Turtle attributes `x`, `y` and `heading`.

5. Modify `steer` so that the Turtles stay in bounds. For debugging, you might want to use the Step button, which invokes `step` once on each Turtle.

6. Modify `steer` so that each Turtle points toward its nearest neighbor. Hint: Turtles have an attribute, `world`, that is a reference to the TurtleWorld they live in, and the TurtleWorld has an attribute, `animals`, that is a list of all Turtles in the world.

7. Modify `steer` so the Turtles play tag. You can add methods to `Tagger` and you can override `steer` and `__init__`, but you may not modify or override `step`, `wobble` or `move`. Also, `steer` is allowed to change the heading of the Turtle but not the position.

   Adjust the rules and your `steer` method for good quality play; for example, it should be possible for the slow Turtle to tag the faster Turtles eventually.

Solution: `http://thinkpython.com/code/Tagger.py`.

5 Exercises

The following are the possible hands in poker, in increasing order of value (and decreasing order of probability):

pair: two cards with the same rank

two pair: two pairs of cards with the same rank

three of a kind: three cards with the same rank

straight: five cards with ranks in sequence (aces can be high or low, so `Ace-2-3-4-5` is a straight and so is `10-Jack-Queen-King-Ace`, but `Queen-King-Ace-2-3` is not.)

flush: five cards with the same suit

full house: three cards with one rank, two cards with another

four of a kind: four cards with the same rank

straight flush: five cards in sequence (as defined above) and with the same suit

The goal of these exercises is to estimate the probability of drawing these various hands.

1. Download the following files from `http://thinkpython.com/code`:

   `Card.py` : A complete version of the `Card`, `Deck` and `Hand` classes in this chapter.

   `PokerHand.py` : An incomplete implementation of a class that represents a poker hand, and some code that tests it.

2. If you run `PokerHand.py`, it deals seven 7-card poker hands and checks to see if any of them contains a flush. Read this code carefully before you go on.

3. Add methods to `PokerHand.py` named `has_pair`, `has_twopair`, etc. that return True or False according to whether or not the hand meets the relevant criteria. Your code should work correctly for "hands" that contain any number of cards (although 5 and 7 are the most common sizes).

4. Write a method named `classify` that figures out the highest-value classification for a hand and sets the `label` attribute accordingly. For example, a 7-card hand might contain a flush and a pair; it should be labeled "flush".

5. When you are convinced that your classification methods are working, the next step is to estimate the probabilities of the various hands. Write a function in `PokerHand.py` that shuffles a deck of cards, divides it into hands, classifies the hands, and counts the number of times various classifications appear.

6. Print a table of the classifications and their probabilities. Run your program with larger and larger numbers of hands until the output values converge to a reasonable degree of accuracy. Compare your results to the values at `http://en.wikipedia.org/wiki/Hand_rankings`.

Solution: `http://thinkpython.com/code/PokerHandSoln.py`.

This exercise uses TurtleWorld from Chapter **??**. You will write code that makes Turtles play tag. If you are not familiar with the rules of tag, see `http://en.wikipedia.org/wiki/Tag_(game)`.

1. Download `http://thinkpython.com/code/Wobbler.py` and run it. You should see a TurtleWorld with three Turtles. If you press the Run button, the Turtles wander at random.

2. Read the code and make sure you understand how it works. The `Wobbler` class inherits from `Turtle`, which means that the `Turtle` methods `lt`, `rt`, `fd` and `bk` work on Wobblers.

   The `step` method gets invoked by TurtleWorld. It invokes `steer`, which turns the Turtle in the desired direction, `wobble`, which makes a random turn in proportion to the Turtle's clumsiness, and `move`, which moves forward a few pixels, depending on the Turtle's speed.

3. Create a file named `Tagger.py`. Import everything from `Wobbler`, then define a class named `Tagger` that inherits from `Wobbler`. Call `make_world` passing the `Tagger` class object as an argument.

4. Add a `steer` method to `Tagger` to override the one in `Wobbler`. As a starting place, write a version that always points the Turtle toward the origin. Hint: use the math function `atan2` and the Turtle attributes `x`, `y` and `heading`.

5. Modify `steer` so that the Turtles stay in bounds. For debugging, you might want to use the Step button, which invokes `step` once on each Turtle.

6. Modify `steer` so that each Turtle points toward its nearest neighbor. Hint: Turtles have an attribute, `world`, that is a reference to the TurtleWorld they live in, and the TurtleWorld has an attribute, `animals`, that is a list of all Turtles in the world.

7. Modify `steer` so the Turtles play tag. You can add methods to `Tagger` and you can override `steer` and `__init__`, but you may not modify or override `step`, `wobble` or `move`. Also, `steer` is allowed to change the heading of the Turtle but not the position.

   Adjust the rules and your `steer` method for good quality play; for example, it should be possible for the slow Turtle to tag the faster Turtles eventually.

Solution: `http://thinkpython.com/code/Tagger.py`.

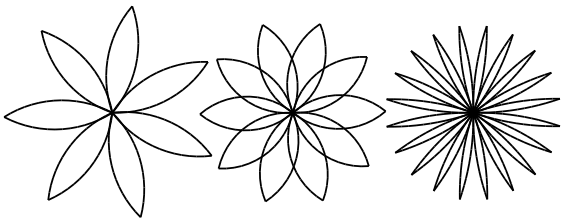

Figure 1: Turtle flowers.

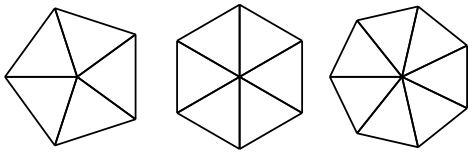

Figure 2: Turtle pies.

6 Exercises

Download the code in this chapter from `http://thinkpython.com/code/polygon.py`.

1. Write appropriate docstrings for `polygon`, `arc` and `circle`.

2. Draw a stack diagram that shows the state of the program while executing `circle(bob, radius)`. You can do the arithmetic by hand or add `print` statements to the code.

3. The version of `arc` in Section **??** is not very accurate because the linear approximation of the circle is always outside the true circle. As a result, the turtle ends up a few units away from the correct destination. My solution shows a way to reduce the effect of this error. Read the code and see if it makes sense to you. If you draw a diagram, you might see how it works.

Write an appropriately general set of functions that can draw flowers as in Figure 1. Solution: `http://thinkpython.com/code/flower.py`, also requires `http://thinkpython.com/code/polygon.py`. Write an appropriately general set of functions that can draw shapes as in Figure 2. Solution: `http://thinkpython.com/code/pie.py`. The letters of the alphabet can be constructed from a moderate number of basic elements, like vertical and horizontal lines and a few curves. Design a font that can be drawn with a minimal number of basic elements and then write functions that draw letters of the alphabet. You should write one function for each letter, with names `draw_a`, `draw_b`, etc., and put your functions in a file named `letters.py`. You can download a "turtle typewriter" from `http://thinkpython.com/code/typewriter.py` to help

you test your code. Solution: `http://thinkpython.com/code/letters.py`, also requires `http://thinkpython.com/code/polygon.py`. Read about spirals at `http://en.wikipedia.org/wiki/Spiral`; then write a program that draws an Archimedian spiral (or one of the other kinds). Solution: `http://thinkpython.com/code/spiral.py`.

7 Exercises

Fermat's Last Theorem says that there are no positive integers a, b, and c such that

$$a^n + b^n = c^n$$

for any values of n greater than 2.

1. Write a function named `check_fermat` that takes four parameters—`a`, `b`, `c` and `n`—and that checks to see if Fermat's theorem holds. If n is greater than 2 and it turns out to be true that

   $$a^n + b^n = c^n$$

   the program should print, "Holy smokes, Fermat was wrong!" Otherwise the program should print, "No, that doesn't work."

2. Write a function that prompts the user to input values for `a`, `b`, `c` and `n`, converts them to integers, and uses `check_fermat` to check whether they violate Fermat's theorem.

If you are given three sticks, you may or may not be able to arrange them in a triangle. For example, if one of the sticks is 12 inches long and the other two are one inch long, it is clear that you will not be able to get the short sticks to meet in the middle. For any three lengths, there is a simple test to see if it is possible to form a triangle:

> If any of the three lengths is greater than the sum of the other two, then you cannot form a triangle. Otherwise, you can. (If the sum of two lengths equals the third, they form what is called a "degenerate" triangle.)

1. Write a function named `is_triangle` that takes three integers as arguments, and that prints either "Yes" or "No," depending on whether you can or cannot form a triangle from sticks with the given lengths.

2. Write a function that prompts the user to input three stick lengths, converts them to integers, and uses `is_triangle` to check whether sticks with the given lengths can form a triangle.

The following exercises use TurtleWorld from Chapter **??**: Read the following function and see if you can figure out what it does. Then run it (see the examples in Chapter **??**).

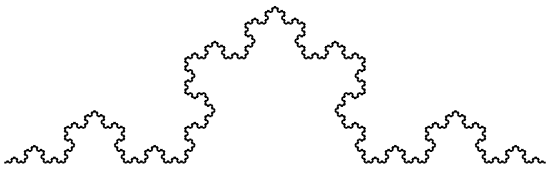

Figure 3: A Koch curve.

```
def draw(t, length, n):
    if n == 0:
        return
    angle = 50
    fd(t, length*n)
    lt(t, angle)
    draw(t, length, n-1)
    rt(t, 2*angle)
    draw(t, length, n-1)
    lt(t, angle)
    bk(t, length*n)
```

The Koch curve is a fractal that looks something like Figure 3. To draw a Koch curve with length x, all you have to do is

1. Draw a Koch curve with length $x/3$.
2. Turn left 60 degrees.
3. Draw a Koch curve with length $x/3$.
4. Turn right 120 degrees.
5. Draw a Koch curve with length $x/3$.
6. Turn left 60 degrees.
7. Draw a Koch curve with length $x/3$.

The exception is if x is less than 3: in that case, you can just draw a straight line with length x.

1. Write a function called `koch` that takes a turtle and a length as parameters, and that uses the turtle to draw a Koch curve with the given length.
2. Write a function called `snowflake` that draws three Koch curves to make the outline of a snowflake. Solution: `http://thinkpython.com/code/koch.py`.
3. The Koch curve can be generalized in several ways. See `http://en.wikipedia.org/wiki/Koch_snowflake` for examples and implement your favorite.

8 Exercises

Draw a stack diagram for the following program. What does the program print? Solution: `http://thinkpython.com/code/stack_diagram.py`.

```
def b(z):
    prod = a(z, z)
    print z, prod
    return prod
def a(x, y):
    x = x + 1
    return x * y
def c(x, y, z):
    total = x + y + z
    square = b(total)**2
    return square
x = 1
y = x + 1
print c(x, y+3, x+y)
```

The Ackermann function, $A(m, n)$, is defined:

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

See `http://en.wikipedia.org/wiki/Ackermann_function`. Write a function named `ack` that evaluates Ackermann's function. Use your function to evaluate `ack(3, 4)`, which should be 125. What happens for larger values of `m` and `n`? Solution: `http://thinkpython.com/code/ackermann.py`. A palindrome is a word that is spelled the same backward and forward, like "noon" and "redivider". Recursively, a word is a palindrome if the first and last letters are the same and the middle is a palindrome. The following are functions that take a string argument and return the first, last, and middle letters:

```
def first(word):
    return word[0]
def last(word):
    return word[-1]
def middle(word):
    return word[1:-1]
```

We'll see how they work in Chapter **??**.

1. Type these functions into a file named `palindrome.py` and test them out. What happens if you call `middle` with a string with two letters? One letter? What about the empty string, which is written `''` and contains no letters?

2. Write a function called `is_palindrome` that takes a string argument and returns `True` if it is a palindrome and `False` otherwise. Remember that you can use the built-in function `len` to check the length of a string.

Solution: `http://thinkpython.com/code/palindrome_soln.py`. A number, a, is a power of b if it is divisible by b and a/b is a power of b. Write a function called `is_power` that takes parameters `a` and `b` and returns `True` if `a` is a power of `b`. Note: you will have to think about the base case. The greatest common divisor (GCD) of a and b is the largest number that divides both of them with no remainder. One way to find the GCD of two numbers is based on the observation that if r is the remainder when a is divided by b, then $gcd(a, b) = gcd(b, r)$. As a base case, we can use $gcd(a, 0) = a$. Write a function called `gcd` that takes parameters `a` and `b` and returns their greatest common divisor. Credit: This exercise is based on an example from Abelson and Sussman's *Structure and Interpretation of Computer Programs*.

9 Exercises

To test the square root algorithm in this chapter, you could compare it with `math.sqrt`. Write a function named `test_square_root` that prints a table like this:

```
1.0 1.0           1.0           0.0
2.0 1.41421356237 1.41421356237 2.22044604925e-16
3.0 1.73205080757 1.73205080757 0.0
4.0 2.0           2.0           0.0
5.0 2.2360679775  2.2360679775  0.0
6.0 2.44948974278 2.44948974278 0.0
7.0 2.64575131106 2.64575131106 0.0
8.0 2.82842712475 2.82842712475 4.4408920985e-16
9.0 3.0           3.0           0.0
```

The first column is a number, a; the second column is the square root of a computed with the function from Section **??**; the third column is the square root computed by `math.sqrt`; the fourth column is the absolute value of the difference between the two estimates. The built-in function `eval` takes a string and evaluates it using the Python interpreter. For example:

```
>>> eval('1 + 2 * 3')
7
>>> import math
>>> eval('math.sqrt(5)')
2.2360679774997898
>>> eval('type(math.pi)')
<type 'float'>
```

Write a function called `eval_loop` that iteratively prompts the user, takes the resulting input and evaluates it using `eval`, and prints the result. It should continue until the user enters `'done'`, and then return the value of the last expression it evaluated. The mathematician Srinivasa Ramanujan found an infinite series that can be used to generate a numerical approximation of $1/\pi$:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Write a function called `estimate_pi` that uses this formula to compute and return an estimate of π. It should use a `while` loop to compute terms of the summation until the last term is smaller than `1e-15` (which is Python notation for 10^{-15}). You can check the result by comparing it to `math.pi`. Solution: `http://thinkpython.com/code/pi.py`.

10 Exercises

A string slice can take a third index that specifies the "step size;" that is, the number of spaces between successive characters. A step size of 2 means every other character; 3 means every third, etc.

```
>>> fruit = 'banana'
>>> fruit[0:5:2]
'bnn'
```

A step size of -1 goes through the word backwards, so the slice `[::-1]` generates a reversed string. Use this idiom to write a one-line version of `is_palindrome` from Exercise 8. The following functions are all *intended* to check whether a string contains any lowercase letters, but at least some of them are wrong. For each function, describe what the function actually does (assuming that the parameter is a string).

```
def any_lowercase1(s):
    for c in s:
        if c.islower():
            return True
        else:
            return False
def any_lowercase2(s):
    for c in s:
        if 'c'.islower():
            return 'True'
        else:
            return 'False'
def any_lowercase3(s):
    for c in s:
```

```
        flag = c.islower()
    return flag
def any_lowercase4(s):
    flag = False
    for c in s:
        flag = flag or c.islower()
    return flag
def any_lowercase5(s):
    for c in s:
        if not c.islower():
            return False
    return True
```

ROT13 is a weak form of encryption that involves "rotating" each letter in a word by 13 places. To rotate a letter means to shift it through the alphabet, wrapping around to the beginning if necessary, so 'A' shifted by 3 is 'D' and 'Z' shifted by 1 is 'A'. Write a function called `rotate_word` that takes a string and an integer as parameters, and that returns a new string that contains the letters from the original string "rotated" by the given amount. For example, "cheer" rotated by 7 is "jolly" and "melon" rotated by -10 is "cubed". You might want to use the built-in functions `ord`, which converts a character to a numeric code, and `chr`, which converts numeric codes to characters. Potentially offensive jokes on the Internet are sometimes encoded in ROT13. If you are not easily offended, find and decode some of them. Solution: `http://thinkpython.com/code/rotate.py`.

11 Exercises

There are solutions to these exercises in the next section. You should at least attempt each one before you read the solutions. In 1939 Ernest Vincent Wright published a 50,000 word novel called *Gadsby* that does not contain the letter "e." Since "e" is the most common letter in English, that's not easy to do. In fact, it is difficult to construct a solitary thought without using that most common symbol. It is slow going at first, but with caution and hours of training you can gradually gain facility. All right, I'll stop now. Write a function called `has_no_e` that returns `True` if the given word doesn't have the letter "e" in it. Modify your program from the previous section to print only the words that have no "e" and compute the percentage of the words in the list have no "e." Write a function named `avoids` that takes a word and a string of forbidden letters, and that returns `True` if the word doesn't use any of the forbidden letters. Modify your program to prompt the user to enter a string of forbidden letters and then print the number of words that don't contain any of them. Can you find a combination of 5 forbidden letters that excludes the smallest number of words? Write a function named `uses_only` that takes a word and a string of letters, and that returns `True` if the word contains only letters in the list. Can

you make a sentence using only the letters `acefhlo`? Other than "Hoe alfalfa?" Write a function named `uses_all` that takes a word and a string of required letters, and that returns `True` if the word uses all the required letters at least once. How many words are there that use all the vowels `aeiou`? How about `aeiouy`? Write a function called `is_abecedarian` that returns `True` if the letters in a word appear in alphabetical order (double letters are ok). How many abecedarian words are there?

12 Exercises

This question is based on a Puzzler that was broadcast on the radio program *Car Talk* (`http://www.cartalk.com/content/puzzlers`):

> Give me a word with three consecutive double letters. I'll give you a couple of words that almost qualify, but don't. For example, the word committee, c-o-m-m-i-t-t-e-e. It would be great except for the 'i' that sneaks in there. Or Mississippi: M-i-s-s-i-s-s-i-p-p-i. If you could take out those i's it would work. But there is a word that has three consecutive pairs of letters and to the best of my knowledge this may be the only word. Of course there are probably 500 more but I can only think of one. What is the word?

Write a program to find it. Solution: `http://thinkpython.com/code/cartalk1.py`. Here's another *Car Talk* Puzzler (`http://www.cartalk.com/content/puzzlers`):

> "I was driving on the highway the other day and I happened to notice my odometer. Like most odometers, it shows six digits, in whole miles only. So, if my car had 300,000 miles, for example, I'd see 3-0-0-0-0-0. "Now, what I saw that day was very interesting. I noticed that the last 4 digits were palindromic; that is, they read the same forward as backward. For example, 5-4-4-5 is a palindrome, so my odometer could have read 3-1-5-4-4-5. "One mile later, the last 5 numbers were palindromic. For example, it could have read 3-6-5-4-5-6. One mile after that, the middle 4 out of 6 numbers were palindromic. And you ready for this? One mile later, all 6 were palindromic! "The question is, what was on the odometer when I first looked?"

Write a Python program that tests all the six-digit numbers and prints any numbers that satisfy these requirements. Solution: `http://thinkpython.com/code/cartalk2.py`. Here's another *Car Talk* Puzzler you can solve with a search (`http://www.cartalk.com/content/puzzlers`):

> "Recently I had a visit with my mom and we realized that the two digits that make up my age when reversed resulted in her age. For

example, if she's 73, I'm 37. We wondered how often this has happened over the years but we got sidetracked with other topics and we never came up with an answer. "When I got home I figured out that the digits of our ages have been reversible six times so far. I also figured out that if we're lucky it would happen again in a few years, and if we're really lucky it would happen one more time after that. In other words, it would have happened 8 times over all. So the question is, how old am I now?"

Write a Python program that searches for solutions to this Puzzler. Hint: you might find the string method `zfill` useful. Solution: `http://thinkpython.com/code/cartalk3.py`.

13 Exercises

Write a function called `is_sorted` that takes a list as a parameter and returns `True` if the list is sorted in ascending order and `False` otherwise. You can assume (as a precondition) that the elements of the list can be compared with the relational operators <, >, etc. For example, `is_sorted([1,2,2])` should return `True` and `is_sorted(['b','a'])` should return `False`. Two words are anagrams if you can rearrange the letters from one to spell the other. Write a function called `is_anagram` that takes two strings and returns `True` if they are anagrams. The (so-called) Birthday Paradox:

1. Write a function called `has_duplicates` that takes a list and returns `True` if there is any element that appears more than once. It should not modify the original list.

2. If there are 23 students in your class, what are the chances that two of you have the same birthday? You can estimate this probability by generating random samples of 23 birthdays and checking for matches. Hint: you can generate random birthdays with the `randint` function in the `random` module.

You can read about this problem at `http://en.wikipedia.org/wiki/Birthday_paradox`, and you can download my solution from `http://thinkpython.com/code/birthday.py`. Write a function called `remove_duplicates` that takes a list and returns a new list with only the unique elements from the original. Hint: they don't have to be in the same order. Write a function that reads the file `words.txt` and builds a list with one element per word. Write two versions of this function, one using the `append` method and the other using the idiom `t = t + [x]`. Which one takes longer to run? Why? Hint: use the `time` module to measure elapsed time. Solution: `http://thinkpython.com/code/wordlist.py`. To check whether a word is in the word list, you could use the `in` operator, but it would be slow because it searches through the words in order. Because the words are in alphabetical order, we can speed things

up with a bisection search (also known as binary search), which is similar to what you do when you look a word up in the dictionary. You start in the middle and check to see whether the word you are looking for comes before the word in the middle of the list. If so, then you search the first half of the list the same way. Otherwise you search the second half. Either way, you cut the remaining search space in half. If the word list has 113,809 words, it will take about 17 steps to find the word or conclude that it's not there. Write a function called `bisect` that takes a sorted list and a target value and returns the index of the value in the list, if it's there, or `None` if it's not. Or you could read the documentation of the `bisect` module and use that! Solution: `http://thinkpython.com/code/inlist.py`. Two words are a "reverse pair" if each is the reverse of the other. Write a program that finds all the reverse pairs in the word list. Solution: `http://thinkpython.com/code/reverse_pair.py`. Two words "interlock" if taking alternating letters from each forms a new word. For example, "shoe" and "cold" interlock to form "schooled." Solution: `http://thinkpython.com/code/interlock.py`. Credit: This exercise is inspired by an example at `http://puzzlers.org`.

1. Write a program that finds all pairs of words that interlock. Hint: don't enumerate all pairs!

2. Can you find any words that are three-way interlocked; that is, every third letter forms a word, starting from the first, second or third?

14 Exercises

If you did Exercise 13, you already have a function named `has_duplicates` that takes a list as a parameter and returns `True` if there is any object that appears more than once in the list. Use a dictionary to write a faster, simpler version of `has_duplicates`. Solution: `http://thinkpython.com/code/has_duplicates.py`. Two words are "rotate pairs" if you can rotate one of them and get the other (see `rotate_word` in Exercise 10). Write a program that reads a wordlist and finds all the rotate pairs. Solution: `http://thinkpython.com/code/rotate_pairs.py`. Here's another Puzzler from *Car Talk* (`http://www.cartalk.com/content/puzzlers`):

> This was sent in by a fellow named Dan O'Leary. He came upon a common one-syllable, five-letter word recently that has the following unique property. When you remove the first letter, the remaining letters form a homophone of the original word, that is a word that sounds exactly the same. Replace the first letter, that is, put it back and remove the second letter and the result is yet another homophone of the original word. And the question is, what's the word? Now I'm going to give you an example that doesn't work. Let's look at the five-letter word, 'wrack.' W-R-A-C-K, you know like to 'wrack with pain.' If I remove the first letter, I am left with a

four-letter word, 'R-A-C-K.' As in, 'Holy cow, did you see the rack on that buck! It must have been a nine-pointer!' It's a perfect homophone. If you put the 'w' back, and remove the 'r,' instead, you're left with the word, 'wack,' which is a real word, it's just not a homophone of the other two words. But there is, however, at least one word that Dan and we know of, which will yield two homophones if you remove either of the first two letters to make two, new four-letter words. The question is, what's the word?

You can use the dictionary from Exercise **??** to check whether a string is in the word list. To check whether two words are homophones, you can use the CMU Pronouncing Dictionary. You can download it from `http://www.speech.cs.cmu.edu/cgi-bin/cmudict` or from `http://thinkpython.com/code/c06d` and you can also download `http://thinkpython.com/code/pronounce.py`, which provides a function named `read_dictionary` that reads the pronouncing dictionary and returns a Python dictionary that maps from each word to a string that describes its primary pronunciation. Write a program that lists all the words that solve the Puzzler. Solution: `http://thinkpython.com/code/homophone.py`.

15 Exercises

Write a function called `most_frequent` that takes a string and prints the letters in decreasing order of frequency. Find text samples from several different languages and see how letter frequency varies between languages. Compare your results with the tables at `http://en.wikipedia.org/wiki/Letter_frequencies`. Solution: `http://thinkpython.com/code/most_frequent.py`.
More anagrams!

1. Write a program that reads a word list from a file (see Section **??**) and prints all the sets of words that are anagrams. Here is an example of what the output might look like:

   ```
   ['deltas', 'desalt', 'lasted', 'salted', 'slated', 'staled']
   ['retainers', 'ternaries']
   ['generating', 'greatening']
   ['resmelts', 'smelters', 'termless']
   ```

   Hint: you might want to build a dictionary that maps from a set of letters to a list of words that can be spelled with those letters. The question is, how can you represent the set of letters in a way that can be used as a key?

2. Modify the previous program so that it prints the largest set of anagrams first, followed by the second largest set, and so on.

3. In Scrabble a "bingo" is when you play all seven tiles in your rack, along with a letter on the board, to form an eight-letter word. What set of 8 letters forms the most possible bingos? Hint: there are seven. Solution: `http://thinkpython.com/code/anagram_sets.py`.

Two words form a "metathesis pair" if you can transform one into the other by swapping two letters; for example, "converse" and "conserve." Write a program that finds all of the metathesis pairs in the dictionary. Hint: don't test all pairs of words, and don't test all possible swaps. Solution: `http://thinkpython.com/code/metathesis.py`. Credit: This exercise is inspired by an example at `http://puzzlers.org`. Here's another Car Talk Puzzler (`http://www.cartalk.com/content/puzzlers`):

> What is the longest English word, that remains a valid English word, as you remove its letters one at a time? Now, letters can be removed from either end, or the middle, but you can't rearrange any of the letters. Every time you drop a letter, you wind up with another English word. If you do that, you're eventually going to wind up with one letter and that too is going to be an English word—one that's found in the dictionary. I want to know what's the longest word and how many letters does it have? I'm going to give you a little modest example: Sprite. Ok? You start off with sprite, you take a letter off, one from the interior of the word, take the r away, and we're left with the word spite, then we take the e off the end, we're left with spit, we take the s off, we're left with pit, it, and I.

Write a program to find all words that can be reduced in this way, and then find the longest one. This exercise is a little more challenging than most, so here are some suggestions:

1. You might want to write a function that takes a word and computes a list of all the words that can be formed by removing one letter. These are the "children" of the word.

2. Recursively, a word is reducible if any of its children are reducible. As a base case, you can consider the empty string reducible.

3. The wordlist I provided, `words.txt`, doesn't contain single letter words. So you might want to add "I", "a", and the empty string.

4. To improve the performance of your program, you might want to memoize the words that are known to be reducible.

Solution: `http://thinkpython.com/code/reducible.py`.

16 Exercises

The "rank" of a word is its position in a list of words sorted by frequency: the most common word has rank 1, the second most common has rank 2, etc.

Zipf's law describes a relationship between the ranks and frequencies of words in natural languages (`http://en.wikipedia.org/wiki/Zipf's_law`). Specifically, it predicts that the frequency, f, of the word with rank r is:

$$f = cr^{-s}$$

where s and c are parameters that depend on the language and the text. If you take the logarithm of both sides of this equation, you get:

$$\log f = \log c - s \log r$$

So if you plot $\log f$ versus $\log r$, you should get a straight line with slope $-s$ and intercept $\log c$.

Write a program that reads a text from a file, counts word frequencies, and prints one line for each word, in descending order of frequency, with $\log f$ and $\log r$. Use the graphing program of your choice to plot the results and check whether they form a straight line. Can you estimate the value of s?

Solution: `http://thinkpython.com/code/zipf.py`. To make the plots, you might have to install matplotlib (see `http://matplotlib.sourceforge.net/`).

sectionExercises

The `urllib` module provides methods for manipulating URLs and downloading information from the web. The following example downloads and prints a secret message from `thinkpython.com`:

```
import urllib

conn = urllib.urlopen('http://thinkpython.com/secret.html')
for line in conn:
    print line.strip()
```

Run this code and follow the instructions you see there. Solution: `http://thinkpython.com/code/zip_code.py`.

17 Exercises

Swampy (see Chapter **??**) provides a module named `World`, which defines a user-defined type also called `World`. You can import it like this:

```
from swampy.World import World
```

Or, depending on how you installed Swampy, like this:

```
from World import World
```

The following code creates a World object and calls the `mainloop` method, which waits for the user.

```
world = World()
world.mainloop()
```

A window should appear with a title bar and an empty square. We will use this window to draw Points, Rectangles and other shapes. Add the following lines before calling `mainloop` and run the program again.

```
canvas = world.ca(width=500, height=500, background='white')
bbox = [[-150,-100], [150, 100]]
canvas.rectangle(bbox, outline='black', width=2, fill='green4')
```

You should see a green rectangle with a black outline. The first line creates a Canvas, which appears in the window as a white square. The Canvas object provides methods like `rectangle` for drawing various shapes.

`bbox` is a list of lists that represents the "bounding box" of the rectangle. The first pair of coordinates is the lower-left corner of the rectangle; the second pair is the upper-right corner.

You can draw a circle like this:

```
canvas.circle([-25,0], 70, outline=None, fill='red')
```

The first parameter is the coordinate pair for the center of the circle; the second parameter is the radius.

If you add this line to the program, the result should resemble the national flag of Bangladesh (see `http://en.wikipedia.org/wiki/Gallery_of_sovereign-state_flags`).

1. Write a function called `draw_rectangle` that takes a Canvas and a Rectangle as arguments and draws a representation of the Rectangle on the Canvas.

2. Add an attribute named `color` to your Rectangle objects and modify `draw_rectangle` so that it uses the color attribute as the fill color.

3. Write a function called `draw_point` that takes a Canvas and a Point as arguments and draws a representation of the Point on the Canvas.

4. Define a new class called Circle with appropriate attributes and instantiate a few Circle objects. Write a function called `draw_circle` that draws circles on the canvas.

5. Write a program that draws the national flag of the Czech Republic. Hint: you can draw a polygon like this:

   ```
   points = [[-150,-100], [150, 100], [150, -100]]
   canvas.polygon(points, fill='blue')
   ```

I have written a small program that lists the available colors; you can download it from `http://thinkpython.com/code/color_list.py`.

18 Exercises

Code examples from this chapter are available from `http://thinkpython.com/code/Time1.py`; solutions to these exercises are available from `http://thinkpython.com/code/Time1_soln.py`.

Write a function called `mul_time` that takes a Time object and a number and returns a new Time object that contains the product of the original Time and the number.

Then use `mul_time` to write a function that takes a Time object that represents the finishing time in a race, and a number that represents the distance, and returns a Time object that represents the average pace (time per mile).

The `datetime` module provides `date` and `time` objects that are similar to the Date and Time objects in this chapter, but they provide a rich set of methods and operators. Read the documentation at `http://docs.python.org/2/library/datetime.html`.

1. Use the `datetime` module to write a program that gets the current date and prints the day of the week.

2. Write a program that takes a birthday as input and prints the user's age and the number of days, hours, minutes and seconds until their next birthday.

3. For two people born on different days, there is a day when one is twice as old as the other. That's their Double Day. Write a program that takes two birthdays and computes their Double Day.

4. For a little more challenge, write the more general version that computes the day when one person is n times older than the other.

19 Exercises

This exercise is a cautionary tale about one of the most common, and difficult to find, errors in Python. Write a definition for a class named `Kangaroo` with the following methods:

1. An `__init__` method that initializes an attribute named `pouch_contents` to an empty list.

2. A method named `put_in_pouch` that takes an object of any type and adds it to `pouch_contents`.

3. A `__str__` method that returns a string representation of the Kangaroo object and the contents of the pouch.

Test your code by creating two `Kangaroo` objects, assigning them to variables named `kanga` and `roo`, and then adding `roo` to the contents of `kanga`'s pouch.

Download http://thinkpython.com/code/BadKangaroo.py. It contains a solution to the previous problem with one big, nasty bug. Find and fix the bug.

If you get stuck, you can download http://thinkpython.com/code/GoodKangaroo.py, which explains the problem and demonstrates a solution.

Visual is a Python module that provides 3-D graphics. It is not always included in a Python installation, so you might have to install it from your software repository or, if it's not there, from http://vpython.org.

The following example creates a 3-D space that is 256 units wide, long and high, and sets the "center" to be the point $(128, 128, 128)$. Then it draws a blue sphere.

```
from visual import *

scene.range = (256, 256, 256)
scene.center = (128, 128, 128)

color = (0.1, 0.1, 0.9)          # mostly blue
sphere(pos=scene.center, radius=128, color=color)
```

color is an RGB tuple; that is, the elements are Red-Green-Blue levels between 0.0 and 1.0 (see http://en.wikipedia.org/wiki/RGB_color_model).

If you run this code, you should see a window with a black background and a blue sphere. If you drag the middle button up and down, you can zoom in and out. You can also rotate the scene by dragging the right button, but with only one sphere in the world, it is hard to tell the difference.

The following loop creates a cube of spheres:

```
t = range(0, 256, 51)
for x in t:
    for y in t:
        for z in t:
            pos = x, y, z
            sphere(pos=pos, radius=10, color=color)
```

1. Put this code in a script and make sure it works for you.

2. Modify the program so that each sphere in the cube has the color that corresponds to its position in RGB space. Notice that the coordinates are in the range 0–255, but the RGB tuples are in the range 0.0–1.0.

3. Download http://thinkpython.com/code/color_list.py and use the function read_colors to generate a list of the available colors on your system, their names and RGB values. For each named color draw a sphere in the position that corresponds to its RGB values.

You can see my solution at http://thinkpython.com/code/color_space.py.