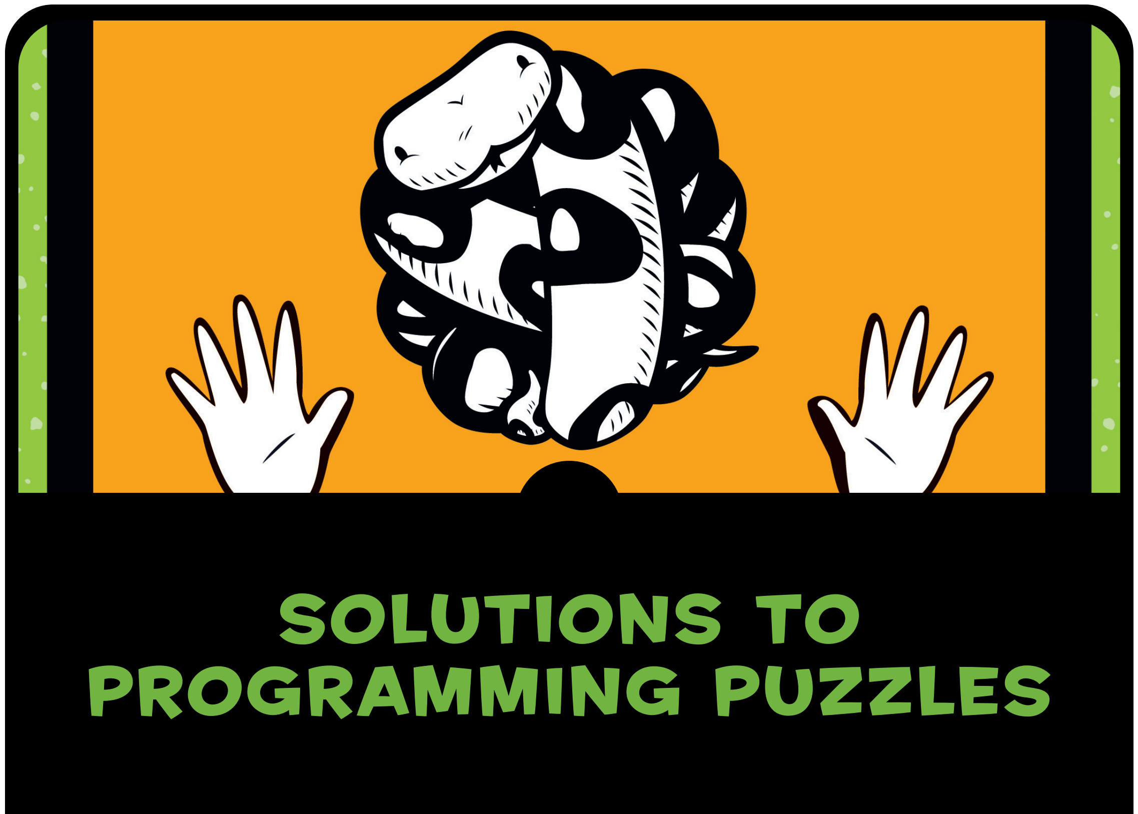


Here are the solutions to the programming puzzles at the ends of the chapters. There's not always a single solution to a puzzle, so the one you've come up with may not match what you'll find here, but the examples will give you an idea of possible approaches.

www.itbook.store/books/9781593274078

CHAPTER 3

#1: FAVORITES

Here's one solution with three favorite hobbies and three favorite foods:

```
>>> hobbies = ['Pokemon', 'LEGO Mindstorms', 'Mountain Biking']
>>> foods = ['Pancakes', 'Chocolate', 'Apples']
>>> favorites = hobbies + foods
>>> print(favorites)
['Pokemon', 'LEGO Mindstorms', 'Mountain Biking', 'Pancakes',
'Chocolate', 'Apples']
```

#2: COUNTING COMBATANTS

We can do the calculation in a number of different ways. We have three buildings with 25 ninjas hiding on each roof, and two tunnels with 40 samurai hiding in each. We could work out the total ninjas and then the total samurai, and just add those two numbers together:

```
>>> 3*25
75
>>> 2*40
80
>>> 75+80
155
```

Much shorter (and better) is to combine those three equations, using parentheses (the parentheses aren't necessary, due to the order of the mathematical operations, but they make the equation easier to read):

```
>>> (3*25)+(2*40)
```

But perhaps a nicer Python program would be something like the following, which tells us what we're calculating:

```
>>> roofs = 3
>>> ninjas_per_roof = 25
>>> tunnels = 2
>>> samurai_per_tunnel = 40
>>> print((roofs * ninjas_per_roof) + (tunnels * samurai_per_tunnel))
155
```


www.itbook.store/books/9781593274078

#3: Greetings!

In this solution, we give the variables meaningful names, and then use format placeholders (%s %s) in our string to embed the values of those variables:

```
>>> first_name = 'Brando'
>>> last_name = 'Ickett'
>>> print('Hi there, %s %s!' % (first_name, last_name))
Hi there, Brando Ickett!
```

Chapter 4

#1: A Rectangle

Drawing a rectangle is almost exactly the same as drawing a square, except that the turtle needs to draw two sides that are longer than the other two:

```
>>> import turtle
>>> t = turtle.Pen()
>>> t.forward(100)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(100)
>>> t.left(90)
>>> t.forward(50)
```

#2: A Triangle

The puzzle didn't specify what sort of triangle to draw. There are three types of triangles: *equilateral*, *isosceles*, and *scalene*. Depending on how much you know about geometry, you may have drawn any sort of triangle, perhaps by fiddling with the angles until you ended up with something that looked right.

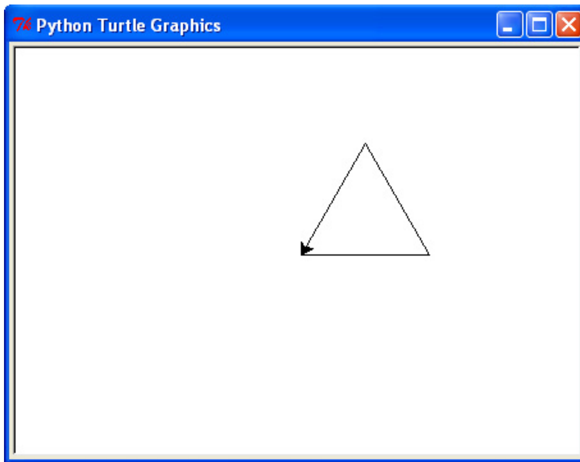
For this example, let's concentrate on the first two types, because they're the most straightforward to draw. An equilateral triangle has three equal sides and three equal angles:

```
  >>> import turtle
  >>> t = turtle.Pen()
❶ >>> t.forward(100)
```


www.itbook.store/books/9781593274078

```
❷ >>> t.left(120)
❸ >>> t.forward(100)
❹ >>> t.left(120)
❺ >>> t.forward(100)
```

We draw the base of the triangle by moving forward 100 pixels at ❶. We turn left 120 degrees (this creates an interior angle of 60 degrees) at ❷, and again move forward 100 pixels at ❸. The next turn is also 120 degrees at ❹, and the turtle moves back to the starting position by moving forward another 100 pixels at ❺. Here's the result of running the code:



An isosceles triangle has two equal sides and two equal angles:

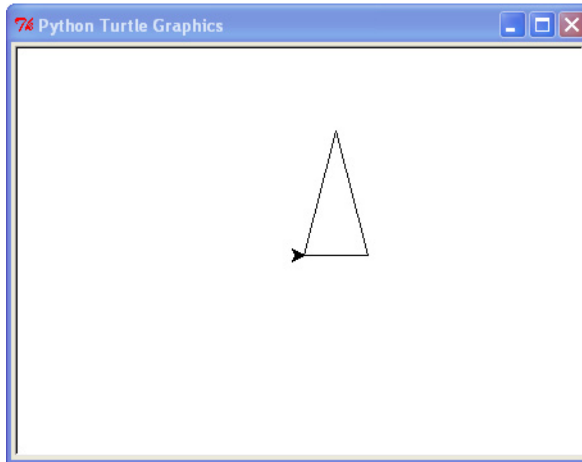
```
>>> import turtle
>>> t = turtle.Pen()
>>> t.forward(50)
>>> t.left(104.47751218592992)
>>> t.forward(100)
>>> t.left(151.04497562814015)
>>> t.forward(100)
```

In this solution, the turtle moves forward 50 pixels, and then turns 104.47751218592992 degrees. It moves forward 100 pixels, followed by a turn of 151.04497562714015 degrees, and then forward 100 pixels again. To turn the turtle back to face its starting position, we can call the following line again:

```
>>> t.left(104.47751218592992)
```

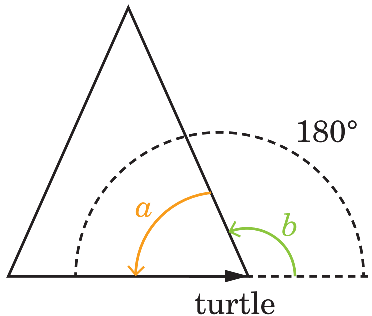

www.itbook.store/books/9781593274078

Here's the result of running this code:



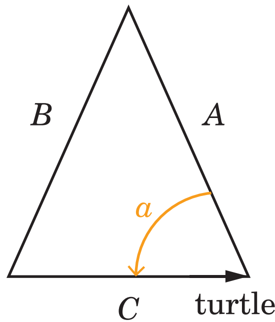
How do we come up with angles like 104.47751218592992 degrees and 151.04497562814015 degrees? After all, those are rather obscure numbers!

Once we've decided on the lengths of each of the sides of the triangle, we can calculate the interior angles using Python and a bit of trigonometry. In the following diagram, you can see that if we know the degree of angle a, we can work out the degrees of the (outside) angle b that the turtle needs to turn. The two angles a and b will add up to 180 degrees.



It's not difficult to calculate the inside angle if you know the right equation. For example, say we want to create a triangle with a bottom length of 50 pixels (let's call that side C), and two sides A and B, both 100 pixels long.

www.itbook.store/books/9781593274078



The equation to calculate the inside angle a using sides A, B, and C would be:

$$a = \arccos\left(\frac{A^2 + C^2 - B^2}{2AC}\right)$$

We can create a little Python program to calculate the value using Python's `math` module:

```
>>> import math
>>> A = 100
>>> B = 100
>>> C = 50
❶ >>> a = math.acos((math.pow(A,2) + math.pow(C,2) - \
        math.pow(B,2)) / (2*A*C))
>>> print(a)
1.31811607165
```

We first `import` the `math` module, and then create variables for each of the sides (`A`, `B`, and `C`). At ❶, we the use the `math` function `acos` (arc cosine) to calculate the angle. This calculation returns the radians value 1.31811607165. Radians are another unit used to measure angles, like degrees.

NOTE *The backslash (\) in the line at ❶ isn't part of the equation—backslashes, as explained in Chapter 16, are used to separate long lines of code. They're not necessary, but in this case we're splitting a long line because it won't fit on a page otherwise.*

The radians value can be converted into degrees using the math function `degrees`, and we can calculate the outside angle (the

www.itbook.store/books/9781593274078

amount we need to tell the turtle to turn), by subtracting this value from 180 degrees:

```
>>> print(180 - math.degrees(a))
104.477512186
```

The equation for the turtle's next turn is similar:

$$b = \arccos\left(\frac{A^2 + B^2 - C^2}{2AB}\right)$$

The code for this equation also looks similar:

```
>>> b = math.acos((math.pow(A,2) + math.pow(B,2) - \
        math.pow(C,2)) / (2*A*B))
>>> print(180 - math.degrees(b))
151.04497562814015
```

Of course, you don't need to use equations to work out the angles. You can also just try playing with the degrees the turtle turns until you get something that looks about right.

#3: A BOX WITHOUT CORNERS

The solution to this puzzle (an octagon missing four sides) is to do the same thing four times in a row. Move forward, turn left 45 degrees, lift the pen, move forward, put the pen down, and turn left 45 degrees again:

```
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
```

So the final set of commands would be like the following code (the best way to run this is to create a new window in the shell, and then save the file as *nocorners.py*):

```
import turtle
t = turtle.Pen()
t.forward(50)
```


www.itbook.store/books/9781593274078

```
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
```

Chapter 5

#1: Are You Rich?

You will actually get an *indentation error* once you reach the last line of the `if` statement:

```
>>> money = 2000
>>> if money > 1000:
❶       print("I'm rich!!")
else:
        print("I'm not rich!!")
❷         print("But I might be later...")
SyntaxError: unexpected indent
```

You get this error because the first block at ❶ starts with four spaces, so Python doesn't expect to see two extra spaces on the final line at ❷. It highlights the place where it sees a problem with a vertical rectangular block, so you can tell where you went wrong.

www.itbook.store/books/9781593274078

#2: Twinkies!

The code to check for a number of Twinkies less than 100 or more than 500 should look like this:

```
>>> twinkies = 600
>>> if twinkies < 100 or twinkies > 500:
>>>     print('Too few or too many')

Too few or too many
```

#3: Just the Right Number

You could write more than one `if` statement to check whether an amount of money falls between 100 and 500 or 1000 and 5000, but by using the `and` and `or` keywords, we can do it with a single statement:

```
if (amount >= 100 and amount <= 500) or (amount >= 1000 \
        and amount <= 5000):
    print('amount is between 100 & 500 or between 1000 & 5000')
```

(Be sure to put brackets around the first and last two conditions so that Python will check whether the amount is between 100 and 500 *or* between 1000 and 5000.)

We can test the code by setting the `amount` variable to different values:

```
>>> amount = 800
>>> if (amount >= 100 and amount <= 500) or (amount >= 1000 \
        and amount <= 5000):
    print('amount is between 100 & 500 or between 1000 & 5000')

>>> amount = 400
>>> if (amount >= 100 and amount <= 500) or (amount >= 1000 \
        and amount <= 5000):
    print('amount is between 100 & 500 or between 1000 & 5000')

amount is between 100 & 500 or between 1000 & 5000
>>> amount = 3000
>>> if (amount >= 100 and amount <= 500) or (amount >= 1000 \
        and amount <= 5000):
    print('amount is between 100 & 500 or between 1000 & 5000')

amount is between 100 & 500 or between 1000 & 5000
```


www.itbook.store/books/9781593274078

#4: I CAN FIGHT THOSE NINJAS

This was a bit of an evil, trick puzzle. If you create the `if` statement in the same order as the instructions, you won't get the result you might be expecting. Here's an example:

```
>>> ninjas = 5
>>> if ninjas < 50:
        print("That's too many")
elif ninjas < 30:
        print("It'll be a struggle, but I can take 'em")
elif ninjas < 10:
        print("I can fight those ninjas!")

That's too many
```

Even though the number of ninjas is less than 10, you get the message "That's too many." This is because the first condition (`< 50`) is *evaluated* first (in other words, Python checks it first), and because the variable really *is* less than 50, the program prints the message you didn't expect to see.

To get it to work properly, reverse the order in which you check the number, so that you see whether the number is less than 10 first:

```
>>> ninjas = 5
>>> if ninjas < 10:
        print("I can fight those ninjas!")
elif ninjas < 30:
        print("It'll be a struggle, but I can take 'em")
elif ninjas < 50:
        print("That's too many")

I can fight those ninjas!
```

CHAPTER 6

#1: THE HELLO LOOP

The `print` statement in this `for` loop is run only once. This is because when Python hits the `if` statement, `x` is less than 9, so it immediately breaks out of the loop.

```
>>> for x in range(0, 20):
        print('hello %s' % x)
```


www.itbook.store/books/9781593274078

```
        if x < 9:
            break

hello 0
```

#2: EVEN NUMBERS

We can use the step parameter with the `range` function to produce the list of even numbers. If you are 14 years old, the start parameter will be 2, and the end parameter will be 16 (because the `for` loop will run until the value just before the end parameter).

```
>>> for x in range(2, 16, 2):
        print(x)
2
4
6
8
10
12
14
```

#3: MY FIVE FAVORITE INGREDIENTS

There are a couple of different ways to print numbers with the items in the list. Here's one way:

```
>>> ingredients = ['snails', 'leeches', 'gorilla belly-button lint',
              'caterpillar eyebrows', 'centipede toes']
❶ >>> x = 1
❷ >>> for i in ingredients:
❸         print('%s %s' % (x, i))
❹         x = x + 1

1 snails
2 leeches
3 gorilla belly-button lint
4 caterpillar eyebrows
5 centipede toes
```

We create a variable `x` to store the number we want to print at ❶. Next, we create a `for` loop to loop through the items in the list at ❷, assigning each to the variable `i`, and we print the value of the `x` and `i` variables at ❸, using the `%s` placeholder. We add 1 to the `x` variable at ❹, so that each time we loop, the number we print increases.

www.itbook.store/books/9781593274078

#4: YOUR WEIGHT ON THE MOON

To calculate your weight in kilograms on the moon over 15 years, first create a variable to store your starting weight:

```
>>> weight = 30
```

For each year, you can calculate the new weight by adding a kilogram, and then multiplying by 16.5 percent (0.165) to get the weight on the moon:

```
>>> weight = 30
>>> for year in range(1, 16):
        weight = weight + 1
        moon_weight = weight * 0.165
        print('Year %s is %s' % (year, moon_weight))

Year 1 is 5.115
Year 2 is 5.28
Year 3 is 5.445
Year 4 is 5.61
Year 5 is 5.775
Year 6 is 5.94
Year 7 is 6.105
Year 8 is 6.2700000000000005
Year 9 is 6.4350000000000005
Year 10 is 6.6000000000000005
Year 11 is 6.765000000000001
Year 12 is 6.930000000000001
Year 13 is 7.095000000000001
Year 14 is 7.260000000000001
Year 15 is 7.425000000000001
```

CHAPTER 7

#1: BASIC MOON WEIGHT FUNCTION

The function should take two parameters: `weight` and `increase` (the amount the weight will increase each year). The rest of the code is very similar to the solution for Puzzle #4 in Chapter 6.

```
>>> def moon_weight(weight, increase):
        for year in range(1, 16):
            weight = weight + increase
            moon_weight = weight * 0.165
            print('Year %s is %s' % (year, moon_weight))
```


www.itbook.store/books/9781593274078

```
>>> moon_weight(40, 0.5)
Year 1 is 6.6825
Year 2 is 6.765
Year 3 is 6.8475
Year 4 is 6.93
Year 5 is 7.0125
Year 6 is 7.095
Year 7 is 7.1775
Year 8 is 7.26
Year 9 is 7.3425
Year 10 is 7.425
Year 11 is 7.5075
Year 12 is 7.59
Year 13 is 7.6725
Year 14 is 7.755
Year 15 is 7.8375
```

#2: Moon Weight Function and Years

We need only a minor change to the function so that the number of years can be passed in as a parameter.

```
>>> def moon_weight(weight, increase, years):
        years = years + 1
        for year in range(1, years):
            weight = weight + increase
            moon_weight = weight * 0.165
            print('Year %s is %s' % (year, moon_weight))

>>> moon_weight(35, 0.3, 5)
Year 1 is 5.8245
Year 2 is 5.874
Year 3 is 5.9235
Year 4 is 5.973
Year 5 is 6.0225
```

Notice on the second line of the function that we add 1 to the years parameter, so that the `for` loop will end on the correct year (rather than the year before).

#3: Moon Weight Program

We can use the `stdin` object of the `sys` module to allow someone to enter values (using the `readline` function). Because `sys.stdin.readline` returns a string, we need to convert these strings into numbers so that we can perform the calculations.

www.itbook.store/books/9781593274078

```
import sys
def moon_weight():
    print('Please enter your current Earth weight')
❶    weight = float(sys.stdin.readline())
    print('Please enter the amount your weight might increase each year')
❷    increase = float(sys.stdin.readline())
    print('Please enter the number of years')
❸    years = int(sys.stdin.readline())
    years = years + 1
    for year in range(1, years):
        weight = weight + increase
        moon_weight = weight * 0.165
        print('Year %s is %s' % (year, moon_weight))
```

At ❶, we read the input using `sys.stdin.readline`, and then convert the string into a float, using the `float` function. This value is stored as the `weight` variable. We do the same process at ❷ for the variable `increase`, but use the `int` function at ❸, because we enter only whole numbers for a number of years (not fractional numbers). The rest of the code after that line is exactly the same as in the previous solution.

If we call the function now, we'll see something like the following:

```
>>> moon_weight()
Please enter your current Earth weight
45
Please enter the amount your weight might increase each year
0.4
Please enter the number of years
12
Year 1 is 7.491
Year 2 is 7.557
Year 3 is 7.623
Year 4 is 7.689
Year 5 is 7.755
Year 6 is 7.821
Year 7 is 7.887
Year 8 is 7.953
Year 9 is 8.019
Year 10 is 8.085
Year 11 is 8.151
Year 12 is 8.217
```


www.itbook.store/books/9781593274078

CHAPTER 8

#1: THE GIRAFFE SHUFFLE

Before adding the functions to make Reginald dance a jig, let's take another look at the `Animals`, `Mammals`, and `Giraffes` classes. Here's the `Animals` class (this used to be a subclass of the `Animate` class, which was removed to make this example a little simpler):

```
class Animals:
    def breathe(self):
        print('breathing')
    def move(self):
        print('moving')
    def eat_food(self):
        print('eating food')
```

The `Mammals` class is a subclass of `Animals`:

```
class Mammals(Animals):
    def feed_young_with_milk(self):
        print('feeding young')
```

And the `Giraffes` class is a subclass of `Mammals`:

```
class Giraffes(Mammals):
    def eat_leaves_from_trees(self):
        print('eating leaves')
```

The functions for moving each foot are pretty easy to add:

```
class Giraffes(Mammals):
    def eat_leaves_from_trees(self):
        print('eating leaves')
    def left_foot_forward(self):
        print('left foot forward')
    def right_foot_forward(self):
        print('right foot forward')
    def left_foot_backward(self):
        print('left foot back')
    def right_foot_backward(self):
        print('right foot back')
```


www.itbook.store/books/9781593274078

The dance function just needs to call each of the foot functions in the right order:

```
    def dance(self):
        self.left_foot_forward()
        self.left_foot_backward()
        self.right_foot_forward()
        self.right_foot_backward()
        self.left_foot_backward()
        self.right_foot_backward()
        self.right_foot_forward()
        self.left_foot_forward()
```

To make Reginald dance, we create an object and call the function:

```
>>> reginald = Giraffes()
>>> reginald.dance()

left foot forward
left foot back
right foot forward
right foot back
left foot back
right foot back
right foot forward
left foot forward
```

#2: TURTLE PITCHFORK

Pen is a class defined in the turtle module, so we can create more than one object of the Pen class for each of the four turtles. If we assign each object to a different variable, we can control them separately, which makes it simple to reproduce the arrowed lines in this puzzle. The concept that each *object* is an independent thing is an important one in programming, particularly when we're talking about classes and objects.

```
import turtle
t1 = turtle.Pen()
t2 = turtle.Pen()
t3 = turtle.Pen()
t4 = turtle.Pen()
```


www.itbook.store/books/9781593274078

```
t1.forward(100)
t1.left(90)
t1.forward(50)
t1.right(90)
t1.forward(50)
t2.forward(110)
t2.left(90)
t2.forward(25)
t2.right(90)
t2.forward(25)
t3.forward(110)
t3.right(90)
t3.forward(25)
t3.left(90)
t3.forward(25)
t4.forward(100)
t4.right(90)
t4.forward(50)
t4.left(90)
t4.forward(50)
```

There are a number of ways to draw the same thing, so your code may not look exactly like this.

CHAPTER 9

#1: MYSTERY CODE

The abs function returns the absolute value of a number, which basically means that a negative number becomes positive. So in this mystery code code, the first print statement displays 20, and the second displays 0.

```
❶ >>> a = abs(10) + abs(-10)
>>> print(a)
20

❷ >>> b = abs(-10) + -10
>>> print(b)
0
```

The calculation at ❶ ends up being 10 + 10. The calculation at ❷ turns into 10 + –10.

www.itbook.store/books/9781593274078

#2: A HIDDEN MESSAGE

The trick here is to first create a string containing the message, and then use the dir function to find out which functions are available on the string:

```
>>> s = 'this if is you not are a reading very this good then way you
to have hide done a it message wrong'
>>> print(dir(s))
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__',
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__',
'__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '_formatter_field_name_split',
'_formatter_parser', 'capitalize', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find', 'format', 'index',
'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Looking at the list, a function called split looks useful. We can use the help function to find out what it does:

```
>>> help(s.split)
Help on built-in function split:

split(...)
    S.split([sep[, maxsplit]]) -> list of strings

    Return a list of the words in S, using sep as the
    delimiter string. If maxsplit is given, at most maxsplit
    splits are done. If sep is not specified or is None, any
    whitespace string is a separator and empty strings are
    removed from the result.
```

According to this description, split returns the string separated into words, according to whatever characters are provided in the sep parameter. If there's no sep parameter, the function uses whitespace. So this function will break up our string.

Now that we know which function to use, we can loop through the words in the string. There are a number of different ways to

www.itbook.store/books/9781593274078

print out every other word (starting with the first). Here's one possibility:

```
❶ >>> message = 'this if is you not are a reading very this good then
way you to have hide done a it message wrong'
❷ >>> words = message.split()
❸ >>> for x in range(0, len(words), 2):
❹         print(words[x])
```

We create the string at ❶, and at ❷, we use the `split` function to separate the string into a list of individual words. We then create a `for` loop using the `range` function at ❸. The first parameter to this function is 0 (the beginning of the list); the next parameter uses the `len` function to find the length of the list (this will be the end of the range); and the final parameter is a step value of 2 (so the range of numbers will look like 0, 2, 4, 6, 8, and so on). We use the `x` variable from our `for` loop to print out values from the list at ❹.

#3: Copying a File

To copy a file, we open it, and then read the contents into a variable. We open the destination file for writing (using the `'w'` parameter for writing), and then write out the contents of the variable. The final code is as follows:

```
f = open('test.txt')
s = f.read()
f.close()
f = open('output.txt', 'w')
f.write(s)
f.close()
```

That example works, but actually a better way to copy a file is using a Python module called `shutil`:

```
import shutil
shutil.copy('test.txt', 'output.txt')
```

Chapter 10

#1: Copied Cars

There are two `print` statements in this code, and we need to figure out what's printed for each one.

www.itbook.store/books/9781593274078

Here's the first:

```
>>> car1 = Car()
❶ >>> car1.wheels = 4
❷ >>> car2 = car1
>>> car2.wheels = 3
>>> print(car1.wheels)

3
```

Why is the result of the `print` statement 3, when we clearly set 4 wheels for `car1` at ❶? Because at ❷, both variables `car1` and `car2` are pointing at the same object.

Now, what about the second `print` statement?

```
>>> car3 = copy.copy(car1)
>>> car3.wheels = 6
>>> print(car1.wheels)

3
```

In this case, `car3` is a copy of the object; it's not labeling the same object as `car1` and `car2`. So when we set the number of wheels to 6, it has no effect on `car1`'s wheels.

#2: PICKLED FAVORITES

We use the `pickle` module to save the contents of a variable (or variables) to a file:

```
❶ >>> import pickle
❷ >>> favorites = ['PlayStation', 'Fudge', 'Movies', 'Python for Kids']
❸ >>> f = open('favorites.dat', 'wb')
❹ >>> pickle.dump(favorites, f)
>>> f.close()
```

We import the `pickle` module at ❶, and create our list of favorite things at ❷. We then `open` a file called *favorites.dat* by passing the string `'wb'` as the second parameter at ❸ (this means write-binary). We then use the `pickle` module's `dump` function to save the contents of the `favorites` variable into the file at ❹.

The second part of this solution is to read the file back in. Assuming you closed and reopened the shell, we'll need to import the `pickle` module again.

www.itbook.store/books/9781593274078

```
>>> import pickle
>>> f = open('favorites.dat', 'rb')
>>> favorites = pickle.load(f)
>>> print(favorites)

['PlayStation', 'Fudge', 'Movies', 'Python for Kids']
```

This is similar to the other code, except that we open the file with the parameter `'rb'` (which means read-binary), and use the `pickle` module's `load` function.

CHAPTER 11

#1: DRAWING AN OCTAGON

An octagon has eight sides, so we'll need at least a `for` loop for this drawing. If you think about the direction of the turtle for a moment, and what it needs to do when drawing the octagon, you may realize that the arrow for the turtle will turn completely around, like the hand of a clock, by the time it finishes drawing. This means it has turned a full 360 degrees. If we divide 360 by the number of sides of the octagon, we get the number of degrees for the angle that the turtle needs to turn after each step of the loop (45 degrees, as mentioned in the hint).

```
>>> import turtle
>>> t = turtle.Pen()
>>> def octagon(size):
        for x in range(1,9):
            t.forward(size)
            t.right(45)
```

We can call the function to test it using 100 as the size of one of the sides:

```
>>> octagon(100)
```

#2: DRAWING A FILLED OCTAGON

If we change the function so that it draws a filled octagon, we'll make it more difficult to draw the outline. A better approach is to pass in a parameter to control whether the octagon should be filled.

www.itbook.store/books/9781593274078

```
>>> import turtle
>>> t = turtle.Pen()
>>> def octagon(size, filled):
❶        if filled == True:
❷            t.begin_fill()
         for x in range(1,9):
             t.forward(size)
             t.right(45)
❸        if filled == True:
❹            t.end_fill()
```

First, we check if the `filled` parameter is set to `True` at ❶. If it is, we tell the turtle to start filling using the `begin_fill` function at ❷. We then draw the octagon on the next two lines, in the same way as Puzzle #1, and then check to see if the `filled` parameter is `True` at ❸. If it is, we call the `end_fill` function at ❹, which actually fills our shape.

We can test this function by setting the color to yellow and calling the function with the parameter set to `True` (so it will fill). We can then set the color back to black, and call the function again with the parameter set to `False` for our outline.

```
>>> t.color(1, 0.85, 0)
>>> octagon(40, True)
>>> t.color(0, 0, 0)
>>> octagon(40, False)
```

#3: A STAR FUNCTION

The trick to this star function is to divide 360 degrees into the number of points, which gives the interior angle for each point of the star (see line ❶ in the following code). To determine the exterior angle, we subtract that number from 180 to get the number of degrees the turtle must turn at ❹.

```
import turtle
t = turtle.Pen()
def draw_star(size, points):
❶    angle = 360 / points
❷    for x in range(0, points):
❸        t.forward(size)
❹        t.left(180 - angle)
❺        t.forward(size)
❻        t.right(180-(angle * 2))
```

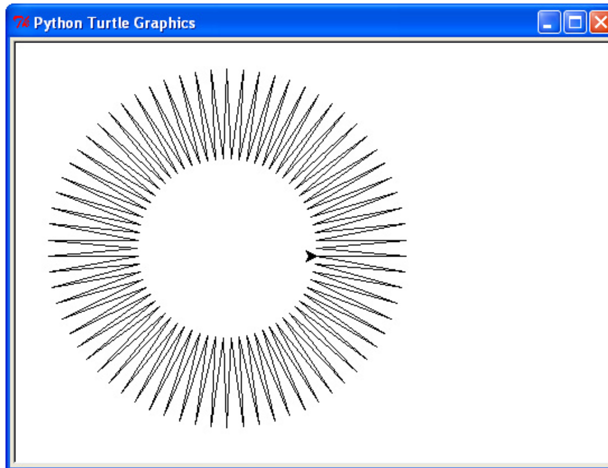

www.itbook.store/books/9781593274078

We loop from 0 up to the number of points at ❷, and then move the turtle forward the number of pixels specified in the size parameter at ❸. We turn the turtle the number of degrees we've previously calculated at ❹, and then move forward again at ❺, which draws the first "spine" of the star. In order to move around in a circular pattern, drawing the spines, we need to increase the angle, so we multiply the calculated angle by two and turn the turtle right at ❻.

For example, you can call this function with 80 pixels and 70 points:

```
>>> draw_star(80, 70)
```

This gives the following results:



CHAPTER 12

#1: FILL THE SCREEN WITH TRIANGLES

To fill the screen with triangles, the first step is to set up the canvas. Let's give it a width and height of 400 pixels.

```
>>> from tkinter import *
>>> import random
>>> w = 400
>>> h = 400
>>> tk = Tk()
>>> canvas = Canvas(tk, width=w, height=h)
>>> canvas.pack()
```


www.itbook.store/books/9781593274078

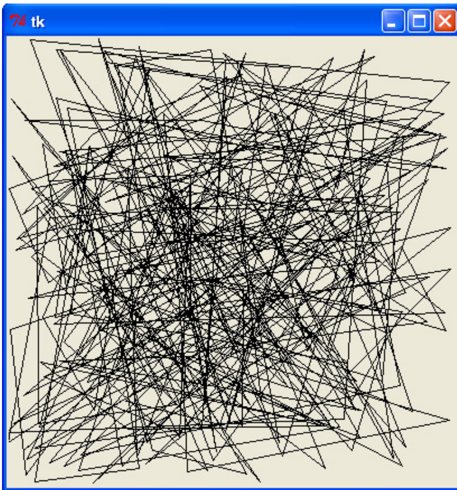
A triangle has three points, which means three sets of x and y coordinates. We can use the `randrange` function in the `random` module (as in the random rectangle example in Chapter 12), to randomly generate the coordinates for the three points (six numbers in total). We can then use the `random_triangle` function to draw the triangle.

```
>>> def random_triangle():
        p1 = random.randrange(w)
        p2 = random.randrange(h)
        p3 = random.randrange(w)
        p4 = random.randrange(h)
        p5 = random.randrange(w)
        p6 = random.randrange(h)
        canvas.create_polygon(p1, p2, p3, p4, p5, p6, \
                fill="", outline="black")
```

Finally, we create a loop to draw a whole bunch of random triangles.

```
>>> for x in range(0, 100):
        random_triangle()
```

This results in something like the following:



To fill the window with random colored triangles, first create a list of colors. We can add this to the setup code at the beginning of the program.

www.itbook.store/books/9781593274078

```
>>> from tkinter import *
>>> import random
>>> w = 400
>>> h = 400
>>> tk = Tk()
>>> canvas = Canvas(tk, width=w, height=h)
>>> canvas.pack()
>>> colors = ['red','green','blue','yellow','orange','white','purple']
```

We can then use the `choice` function of the `random` module to randomly pick an item from this list of colors, and use it in the call to `create_polygon`:

```
def random_triangle():
    p1 = random.randrange(w)
    p2 = random.randrange(h)
    p3 = random.randrange(w)
    p4 = random.randrange(h)
    p5 = random.randrange(w)
    p6 = random.randrange(h)
    color = random.choice(colors)
    canvas.create_polygon(p1, p2, p3, p4, p5, p6, \
            fill=color, outline="")
```

Suppose we loop 100 times again:

```
>>> for x in range(0, 100):
        random_triangle()
```

The result will be something like these triangles:

www.itbook.store/books/9781593274078

#2: THE MOVING TRIANGLE

For the moving triangle, first we set up the canvas again, and then draw the triangle using the create_polygon function:

```
import time
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=200, height=400)
canvas.pack()
canvas.create_polygon(10, 10, 10, 60, 50, 35)
```

To move the triangle horizontally across the screen, the x value should be a positive number and the y value should be 0. We can create a for loop for this, using 1 as the ID of the triangle, 10 as the x parameter, and 0 as the y parameter:

```
for x in range(0, 35):
    canvas.move(1, 10, 0)
    tk.update()
    time.sleep(0.05)
```

Moving down the screen is very similar, with a 0 value for the x parameter and a positive value for the y parameter:

```
for x in range(0, 14):
    canvas.move(1, 0, 10)
    tk.update()
    time.sleep(0.05)
```

To move back across the screen, we need a negative value for the x parameter (and, once again, 0 for the y parameter). To move up, we need a negative value for the y parameter.

```
for x in range(0, 35):
    canvas.move(1, -10, 0)
    tk.update()
    time.sleep(0.05)

for x in range(0, 14):
    canvas.move(1, 0, -10)
    tk.update()
    time.sleep(0.05)
```


www.itbook.store/books/9781593274078

#3: THE MOVING PHOTO

The code for the moving photo solution depends on the size of your image, but assuming your image is called *face.gif*, and you've saved it to your *C:* drive, you can display it, and then move it just like any other drawn shape.

```
import time
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
myimage = PhotoImage(file='c:\\face.gif')
canvas.create_image(0, 0, anchor=NW, image=myimage)
for x in range(0, 35):
    canvas.move(1, 10, 10)
    tk.update()
    time.sleep(0.05)
```

This code will move the image diagonally down the screen.

If you're using Ubuntu or Mac OS X, the filename of the image will be different. If the file is in your home directory, on Ubuntu, loading the image might look like this:

```
myimage = PhotoImage(file='/home/malcolm/face.gif')
```

On a Mac, loading the image might look like this:

```
myimage = PhotoImage(file='/Users/samantha/face.gif')
```

CHAPTER 14

#1: DELAY THE GAME START

To make the game start when the player clicks the canvas, we need to make a couple of small changes to the program. The first is to add a new function to the `Paddle` class:

```
    def turn_left(self, evt):
        self.x = -2

    def turn_right(self, evt):
        self.x = 2
```


www.itbook.store/books/9781593274078

```
    def start_game(self, evt):
        self.started = True
```

This function will set the object variable started to True when it's called. We also need to include this object variable in the __init__ function of Paddle (and set it to False), and then add an event binding for the start_game function (binding it to the mouse button).

```
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
❶       self.started = False
        self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
❷       self.canvas.bind_all('<Button-1>', self.start_game)
```

You can see the addition of the new object variable started at ❶, and the binding for the mouse button at ❷.

The final change is to the last loop in the code. We need to check that the object variable started is True before drawing the ball and paddle, which you can see in this if statement.

```
while 1:
    if ball.hit_bottom == False and paddle.started == True:
        ball.draw()
        paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

#2: A PROPER "GAME OVER"

We can use the create_text function to create the "Game Over" text. We'll add this just after the code to create the ball and paddle.

```
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, paddle, 'red')
game_over_text = canvas.create_text(250, 200, text='GAME OVER', \
        state='hidden')
```


www.itbook.store/books/9781593274078

The `create_text` function has a named parameter called `state`, which we set to the string `'hidden'`. This means that Python draws the text, but makes it invisible. To display the text once the game is over, we add a new `if` statement to the loop at the bottom of the code:

```
while 1:
    if ball.hit_bottom == False and paddle.started == True:
        ball.draw()
        paddle.draw()
❶   if ball.hit_bottom == True:
❷       time.sleep(1)
❸       canvas.itemconfig(game_over_text, state='normal')
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

We see if the `hit_bottom` object variable is set to `True` at ❶. If it is, we sleep for 1 second at ❷ (to give a short delay before displaying the text), and then change the `state` parameter of the text to `'normal'` rather than `'hidden'` at ❸, using the `itemconfig` function of the canvas. We pass two parameters to this function: the identifier of the text drawn on the canvas (stored in the variable `game_over_text`) and the named parameter `state`.

#3: Accelerate the Ball

This change is simple, but you may have found it difficult to figure out where in the code to make the change. We want the ball to speed up if it's travelling in the same horizontal direction when it hits the paddle, and slow down if it's going in the opposite horizontal direction. In order to do this, the left-right (horizontal) speed of the paddle should be added to the horizontal speed of the ball.

The simplest place to make this change is in the `hit_paddle` function of the `Ball` class:

```
    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
❶           if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
❷               self.x += self.paddle.x
                return True
        return False
```


www.itbook.store/books/9781593274078

Once we've determined that the ball has hit the paddle at ❶, we add the value of the x variable of the paddle object to the x variable of the ball at ❷. If the paddle is moving across the screen to the right (its x variable might be set to 2, for example), and the ball strikes it, traveling to the right with an x value of 3, the ball will bounce off the paddle with a new (horizontal) speed of 5. Adding both x variables together means the ball gets a new speed when it hits the paddle.

#4: Record the Player's Score

To add the score to our game, we can create a new class called Score:

```
class Score:
    def __init__(self, canvas, color):
❶       self.score = 0
❷       self.canvas = canvas
❸       self.id = canvas.create_text(450, 10, text=self.score, \
                fill=color)
```

The __init__ function of the Score class takes three parameters: self, canvas, and color. The first line of this function sets up an object variable score, with a value of 0, at ❶. We also store the canvas parameter to use later as the object variable canvas at ❷.

We use the canvas parameter to create our score text, displaying it at position (450, 10), and setting the fill to the value of the color parameter at ❸. The text to display is the current value of the score variable (in other words, 0).

The Score class needs another function, which will be used to increase the score and redisplay the new value:

```
class Score:
    def __init__(self, canvas, color):
        self.score = 0
        self.canvas = canvas
        self.id = canvas.create_text(450, 10, text=self.score, \
               fill=color)

❶   def hit(self):
❷       self.score += 1
❸       self.canvas.itemconfig(self.id, text=self.score)
```


www.itbook.store/books/9781593274078

The hit function takes no parameters ❶, and simply increases the score by 1 at ❷, before using the itemconfig function of the canvas object to change the text displayed to the new score value at ❸.

We can create an object of the Score class just before we create the paddle and ball objects:

```
score = Score(canvas, 'green')
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, paddle, score, 'red')
game_over_text = canvas.create_text(250, 200, text='GAME OVER', \
        state='hidden')
```

The final change to this code is in the Ball class. We need to store the Score object (which we use when we create the Ball object), and then trigger the hit function in the hit_paddle function of the ball.

The beginning of the Ball's __init__ function now has a parameter score, which we use to create an object variable, also called score.

```
    def __init__(self, canvas, paddle, score, color):
        self.canvas = canvas
        self.paddle = paddle
        self.score = score
```

The hit_paddle function should now look like this:

```
    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                self.x += self.paddle.x
                self.score.hit()
                return True
        return False
```

The full game code once all four of these puzzles are completed now looks like this:

```
from tkinter import *
import random
import time
```


www.itbook.store/books/9781593274078

```
class Ball:
    def __init__(self, canvas, paddle, score, color):
        self.canvas = canvas
        self.paddle = paddle
        self.score = score
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
        self.canvas.move(self.id, 245, 100)
        starts = [-3, -2, -1, 1, 2, 3]
        random.shuffle(starts)
        self.x = starts[0]
        self.y = -3
        self.canvas_height = self.canvas.winfo_height()
        self.canvas_width = self.canvas.winfo_width()
        self.hit_bottom = False

    def hit_paddle(self, pos):
        paddle_pos = self.canvas.coords(self.paddle.id)
        if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
            if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
                self.x += self.paddle.x
                self.score.hit()
                return True
        return False

    def draw(self):
        self.canvas.move(self.id, self.x, self.y)
        pos = self.canvas.coords(self.id)
        if pos[1] <= 0:
            self.y = 3
        if pos[3] >= self.canvas_height:
            self.hit_bottom = True
        if self.hit_paddle(pos) == True:
            self.y = -3
        if pos[0] <= 0:
            self.x = 3
        if pos[2] >= self.canvas_width:
            self.x = -3

class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)
        self.x = 0
        self.canvas_width = self.canvas.winfo_width()
        self.started = False
```


www.itbook.store/books/9781593274078

```
        self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
        self.canvas.bind_all('<Button-1>', self.start_game)

    def draw(self):
        self.canvas.move(self.id, self.x, 0)
        pos = self.canvas.coords(self.id)
        if pos[0] <= 0:
            self.x = 0
        elif pos[2] >= self.canvas_width:
            self.x = 0

    def turn_left(self, evt):
        self.x = -2

    def turn_right(self, evt):
        self.x = 2

    def start_game(self, evt):
        self.started = True

class Score:
    def __init__(self, canvas, color):
        self.score = 0
        self.canvas = canvas
        self.id = canvas.create_text(450, 10, text=self.score, \
                fill=color)

    def hit(self):
        self.score += 1
        self.canvas.itemconfig(self.id, text=self.score)

tk = Tk()
tk.title("Game")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0, highlightthickness=0)
canvas.pack()
tk.update()

score = Score(canvas, 'green')
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas, paddle, score, 'red')
game_over_text = canvas.create_text(250, 200, text='GAME OVER', \
        state='hidden')
```


www.itbook.store/books/9781593274078

```
while 1:
    if ball.hit_bottom == False and paddle.started == True:
        ball.draw()
        paddle.draw()
    if ball.hit_bottom == True:
        time.sleep(1)
        canvas.itemconfig(game_over_text, state='normal')
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

CHAPTER 16

#1: CHECKERBOARD

In order to draw a checkerboard background, we need to change the loops in the `__init__` function of our game, as follows:

```
        self.bg = PhotoImage(file="background.gif")
        w = self.bg.width()
        h = self.bg.height()
❶       draw_background = 0
        for x in range(0, 5):
            for y in range(0, 5):
❷               if draw_background == 1:
❸                   self.canvas.create_image(x * w, y * h, \
                            image=self.bg, anchor='nw')
❹                   draw_background = 0
❺               else:
❻                   draw_background = 1
```

At ❶, we create a variable called `draw_background` and set its value to 0. At ❷, we check if the value of the variable is 1, and if it is, we draw the background image at ❸ and set the variable back to 0 at ❹. If the value isn't 1 (that's the `else` at ❺), we set its value to 1 at ❻.

What's this change to our code doing? Well, the first time we hit the `if` statement, it won't draw the background image, and `draw_background` will be set to 1. The next time we hit the `if` statement, we will draw the image and set the variable value back to 0. Each time we loop, we flip the value of the variable. One time we draw the image; the next time we don't.

www.itbook.store/books/9781593274078

#2: TWO-IMAGE CHECKERBOARD

Once you've figured out the checkerboard, drawing two alternating images instead of an image and a blank is quite easy. We need to load the new background image as well as the original. In the following example, we load our new image *background2.gif* (you'll need to draw it in GIMP first) and save it as the object variable `bg2`.

```
        self.bg = PhotoImage(file="background.gif")
        self.bg2 = PhotoImage(file="background2.gif")
        w = self.bg.width()
        h = self.bg.height()
        draw_background = 0
        for x in range(0, 5):
            for y in range(0, 5):
                if draw_background == 1:
                    self.canvas.create_image(x * w, y * h, \
                            image=self.bg, anchor='nw')
                    draw_background = 0
                else:
                    self.canvas.create_image(x * w, y * h, \
                            image=self.bg2, anchor='nw')
                    draw_background = 1
```

In the second part of the `if` statement we created in the Puzzle #1 solution, we use the `create_image` function to draw the new image on the screen.

#3: BOOKSHELF AND LAMP

To draw different backgrounds, we can start with our alternating checkerboard code, but we'll change it again to load a couple of new images, and then dot them around the canvas. For this example, I first copied the image *background2.gif* and drew a bookshelf on it, saving the new image as *shelf.gif*. I then made another copy of *background2.gif*, drew a lamp, and called the new image *lamp.gif*.

```
        self.bg = PhotoImage(file="background.gif")
        self.bg2 = PhotoImage(file="background2.gif")
❶       self.bg_shelf = PhotoImage(file="shelf.gif")
❷       self.bg_lamp = PhotoImage(file="lamp.gif")
        w = self.bg.width()
        h = self.bg.height()
❸       count = 0
```


www.itbook.store/books/9781593274078

```
        draw_background = 0
        for x in range(0, 5):
            for y in range(0, 5):
                if draw_background == 1:
                    self.canvas.create_image(x * w, y * h, \
                            image=self.bg, anchor='nw')
                    draw_background = 0
                else:
❹                   count = count + 1
❺                   if count == 5:
❻                       self.canvas.create_image(x * w, y * h, \
                                image=self.bg_shelf, anchor='nw')
❼                   elif count == 9:
❽                       self.canvas.create_image(x * w, y * h, \
                                image=self.bg_lamp, anchor='nw')
                    else:
                        self.canvas.create_image(x * w, y * h, \
                                image=self.bg2, anchor='nw')
                    draw_background = 1
```

We load the new images at ❶ and ❷, saving them as variables bg_shelf and bg_lamp, respectively, and then create a new variable called count at ❸. In the previous solution, we had an if statement where we drew one background image or another based on the value in the variable draw_background. We do the same thing here, except rather than just displaying the alternate background image, we increment the value in the variable count by adding 1 (using count = count + 1) at ❹. Based on the value in count, we then decide which image to draw. At ❺, if the value has reached 5, we draw the shelf image ❻. At ❼, if the value has reached 9, we draw the lamp image ❽. Otherwise we just draw the alternate background as we did previously.

CHAPTER 18

#1: "YOU WIN!"

We can add the "You Win!" text as a variable of the Game class in its __init__ function:

```
        for x in range(0, 5):
            for y in range(0, 5):
                self.canvas.create_image(x * w, y * h, \
                        image=self.bg, anchor='nw')
```


www.itbook.store/books/9781593274078

```
        self.sprites = []
        self.running = True
        self.game_over_text = self.canvas.create_text(250, 250, \
                text='YOU WIN!', state='hidden')
```

To display the text when the game ends, we just need to add an else statement to the mainloop function:

```
    def mainloop(self):
        while 1:
            if self.running == True:
                for sprite in self.sprites:
                    sprite.move()
❶           else:
❷               time.sleep(1)
❸               self.canvas.itemconfig(self.game_over_text, \
                        state='normal')
            self.tk.update_idletasks()
            self.tk.update()
            time.sleep(0.01)
```

You can see this change in lines ❶ through ❸. We add an else clause to the if statement at ❶, and Python runs this block of code if the running variable is no longer set to True. At ❷, we sleep for a second so that the "You Win!" text doesn't immediately appear, and then change the state of the text to 'normal' at ❸ so that it appears on the canvas.

#2: Animating the Door

To animate the door so that it opens and closes when the stick figure reaches it, we need to change the DoorSprite class first. Rather than passing the image as a parameter, the sprite will now load the two door images itself, in the __init__ function:

```
class DoorSprite(Sprite):
      def __init__(self, game, x, y, width, height):
          Sprite.__init__(self, game)
❶         self.closed_door = PhotoImage(file="door1.gif")
❷         self.open_door = PhotoImage(file="door2.gif")
          self.image = game.canvas.create_image(x, y, \
                image=self.closed_door, anchor='nw')
          self.coordinates = Coords(x, y, x + (width / 2), y + height)
          self.endgame = True
```


www.itbook.store/books/9781593274078

As you can see, the two images are loaded into object variables at ❶ and ❷. We'll now need to change the code at the bottom of the game where we create the `door` object so that it no longer tries to use an image parameter:

```
door = DoorSprite(g, 45, 30, 40, 35)
```

`DoorSprite` needs two new functions: one to display the open door image and one to display the closed door image.

```
    def opendoor(self):
❶        self.game.canvas.itemconfig(self.image, image=self.open_door)
❷        self.game.tk.update_idletasks()

    def closedoor(self):
❸        self.game.canvas.itemconfig(self.image,
                image=self.closed_door)
        self.game.tk.update_idletasks()
```

Using the `itemconfig` function of the canvas, we change the displayed image to the image stored in the `open_door` object variable at ❶. We call the `update_idletasks` function of the `tk` object to force the new image to be displayed at ❷. (If we don't do this, the image won't change immediately.) The `closedoor` function is similar, but displays the image stored in the `closed_door` variable at ❸.

The next new function is added to the `StickFigureSprite` class:

```
    def end(self, sprite):
❶        self.game.running = False
❷        sprite.opendoor()
❸        time.sleep(1)
❹        self.game.canvas.itemconfig(self.image, state='hidden')
❺        sprite.closedoor()
```

We set the `running` object variable of the game to `False` at ❶, and then call the `opendoor` function of the `sprite` parameter at ❷. This is actually a `DoorSprite` object, which we'll see in the next section of code. At ❸, we sleep for 1 second before hiding the stick figure at ❹ and then calling the `closedoor` function at ❺. This makes it look as though the stick figure has gone through the door and closed the door behind him.

The final change is to the `move` function of the `StickFigureSprite`. In the earlier version of the code, when the stick figure collided

www.itbook.store/books/9781593274078

with the door, we set the running variable to False, but since this has been moved to the end function, we need to call that function instead:

```
            if left and self.x < 0 and collided_left(co, sprite_co):
                self.x = 0
                left = False
❶               if sprite.endgame:
❷                   self.end(sprite)
            if right and self.x > 0 and collided_right(co, sprite_co):
                self.x = 0
                right = False
❸               if sprite.endgame:
❹                   self.end(sprite)
```

In the section of code where we check whether the stick figure is moving left, and whether he has collided with a sprite to the left, we check if the endgame variable is True at ❶. If it is, we know that this is a DoorSprite object, and at ❷ we call the end function using the sprite variable as the parameter. We make the same change in the section of code where we see if the stick figure is moving right and has collided with a sprite to the right (at ❸ and ❹).

#3: MOVING PLATFORMS

A moving platform class will be similar to the class for the stick figure. We'll need to recalculate the position of the platform, rather than having a fixed set of coordinates. We can create a subclass of the PlatformSprite class, so the __init__ function becomes as follows:

```
class MovingPlatformSprite(PlatformSprite):
❶   def __init__(self, game, photo_image, x, y, width, height):
❷       PlatformSprite.__init__(self, game, photo_image, x, y, \
                width, height)
❸       self.x = 2
❹       self.counter = 0
❺       self.last_time = time.time()
❻       self.width = width
❼       self.height = height
```

We pass in the same parameters as the PlatformSprite class at ❶, and then call the __init__ function of the parent class with those same parameters at ❷. This means that any object of the

www.itbook.store/books/9781593274078

MovingPlatformSprite class will be set up exactly the same as an object of the PlatformSprite class. We then create an x variable with the value of 2 (the platform will start moving right) at ❸, followed by a counter variable at ❹. We'll use this counter to signal when the platform should change direction. Because we don't want the platform to move back and forth as fast as possible, in the same way that our StickFigureSprite shouldn't move back and forth as fast as possible, we'll record the time in the last_time variable at ❺ (this last_time variable will be used to slow down the movement of the platform). The final additions to this function are to save the width and height at ❻ and ❼.

The next addition to our new class is the coords function:

```
        self.last_time = time.time()
        self.width = width
        self.height = height

    def coords(self):
        xy = self.game.canvas.coords(self.image)
        self.coordinates.x1 = xy[0]
        self.coordinates.y1 = xy[1]
❶       self.coordinates.x2 = xy[0] + self.width
❷       self.coordinates.y2 = xy[1] + self.height
        return self.coordinates
```

The coords function is almost exactly the same as the one we used for the stick figure, except that rather than using a fixed width and height, we use the values we stored in the __init__ function. (You can see the difference on lines ❶ and ❷.)

Since this is a moving sprite, we also need to add a move function:

```
        self.coordinates.x2 = xy[0] + self.width
        self.coordinates.y2 = xy[1] + self.height
        return self.coordinates

    def move(self):
❶       if time.time() - self.last_time > 0.03:
❷           self.last_time = time.time()
❸           self.game.canvas.move(self.image, self.x, 0)
❹           self.counter = self.counter + 1
❺           if self.counter > 20:
❻               self.x = self.x * -1
❼               self.counter = 0
```


www.itbook.store/books/9781593274078

The move function checks to see if the time is greater than three-tenths of a second at ❶. If it is, we set the last_time variable to the current time at ❷. At ❸, we move the platform image, and then increment the counter variable at ❹. If the counter is greater than 20 (the if statement at ❺), we reverse the direction of movement by multiplying the x variable by –1 (so if it's positive it becomes negative, and if it's negative it becomes positive) at ❻, and reset the counter to 0 at ❼. Now the platform will move in one direction for a count of 20, and then back the other way for a count of 20.

To test the moving platforms, we can change a couple of the existing platform objects from PlatformSprite to MovingPlatformSprite:

```
platform5 = MovingPlatformSprite(g, PhotoImage(file="platform2.gif"), \
        175, 350, 66, 10)
platform6 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        50, 300, 66, 10)
platform7 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        170, 120, 66, 10)
platform8 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        45, 60, 66, 10)
platform9 = MovingPlatformSprite(g, PhotoImage(file="platform3.gif"), \
        170, 250, 32, 10)
platform10 = PlatformSprite(g, PhotoImage(file="platform3.gif"), \
        230, 200, 32, 10)
```

The following shows the full code with all the changes.

```
from tkinter import *
import random
import time

class Game:
    def __init__(self):
        self.tk = Tk()
        self.tk.title("Mr Stick Man Races for the Exit")
        self.tk.resizable(0, 0)
        self.tk.wm_attributes("-topmost", 1)
        self.canvas = Canvas(self.tk, width=500, height=500, \
                highlightthickness=0)
        self.canvas.pack()
        self.tk.update()
        self.canvas_height = 500
        self.canvas_width = 500
        self.bg = PhotoImage(file="background.gif")
```


www.itbook.store/books/9781593274078

```
        w = self.bg.width()
        h = self.bg.height()
        for x in range(0, 5):
            for y in range(0, 5):
                self.canvas.create_image(x * w, y * h, \
                        image=self.bg, anchor='nw')
        self.sprites = []
        self.running = True
        self.game_over_text = self.canvas.create_text(250, 250, \
                text='YOU WIN!', state='hidden')

    def mainloop(self):
        while 1:
            if self.running:
                for sprite in self.sprites:
                    sprite.move()
            else:
                time.sleep(1)
                self.canvas.itemconfig(self.game_over_text, \
                        state='normal')
            self.tk.update_idletasks()
            self.tk.update()
            time.sleep(0.01)

class Coords:
    def __init__(self, x1=0, y1=0, x2=0, y2=0):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

def within_x(co1, co2):
    if (co1.x1 > co2.x1 and co1.x1 < co2.x2) \
            or (co1.x2 > co2.x1 and co1.x2 < co2.x2) \
            or (co2.x1 > co1.x1 and co2.x1 < co1.x2) \
            or (co2.x2 > co1.x1 and co2.x2 < co1.x1):
        return True
    else:
        return False

def within_y(co1, co2):
    if (co1.y1 > co2.y1 and co1.y1 < co2.y2) \
            or (co1.y2 > co2.y1 and co1.y2 < co2.y2) \
            or (co2.y1 > co1.y1 and co2.y1 < co1.y2) \
            or (co2.y2 > co1.y1 and co2.y2 < co1.y1):
        return True
```


www.itbook.store/books/9781593274078

```
    else:
        return False

def collided_left(co1, co2):
    if within_y(co1, co2):
        if co1.x1 <= co2.x2 and co1.x1 >= co2.x1:
            return True
    return False

def collided_right(co1, co2):
    if within_y(co1, co2):
        if co1.x2 >= co2.x1 and co1.x2 <= co2.x2:
            return True
    return False

def collided_top(co1, co2):
    if within_x(co1, co2):
        if co1.y1 <= co2.y2 and co1.y1 >= co2.y1:
            return True
    return False

def collided_bottom(y, co1, co2):
    if within_x(co1, co2):
        y_calc = co1.y2 + y
        if y_calc >= co2.y1 and y_calc <= co2.y2:
            return True
    return False

class Sprite:
    def __init__(self, game):
        self.game = game
        self.endgame = False
        self.coordinates = None
    def move(self):
        pass
    def coords(self):
        return self.coordinates

class PlatformSprite(Sprite):
    def __init__(self, game, photo_image, x, y, width, height):
        Sprite.__init__(self, game)
        self.photo_image = photo_image
        self.image = game.canvas.create_image(x, y, \
                image=self.photo_image, anchor='nw')
        self.coordinates = Coords(x, y, x + width, y + height)
```


www.itbook.store/books/9781593274078

```
class MovingPlatformSprite(PlatformSprite):
    def __init__(self, game, photo_image, x, y, width, height):
        PlatformSprite.__init__(self, game, photo_image, x, y, \
                width, height)
        self.x = 2
        self.counter = 0
        self.last_time = time.time()
        self.width = width
        self.height = height

    def coords(self):
        xy = self.game.canvas.coords(self.image)
        self.coordinates.x1 = xy[0]
        self.coordinates.y1 = xy[1]
        self.coordinates.x2 = xy[0] + self.width
        self.coordinates.y2 = xy[1] + self.height
        return self.coordinates

    def move(self):
        if time.time() - self.last_time > 0.03:
            self.last_time = time.time()
            self.game.canvas.move(self.image, self.x, 0)
            self.counter += 1
            if self.counter > 20:
                self.x *= -1
                self.counter = 0

class DoorSprite(Sprite):
    def __init__(self, game, x, y, width, height):
        Sprite.__init__(self, game)
        self.closed_door = PhotoImage(file="door1.gif")
        self.open_door = PhotoImage(file="door2.gif")
        self.image = game.canvas.create_image(x, y, \
                image=self.closed_door, anchor='nw')
        self.coordinates = Coords(x, y, x + (width / 2), y + height)
        self.endgame = True

    def opendoor(self):
        self.game.canvas.itemconfig(self.image, image=self.open_door)
        self.game.tk.update_idletasks()

    def closedoor(self):
        self.game.canvas.itemconfig(self.image, \
                image=self.closed_door)
        self.game.tk.update_idletasks()
```


www.itbook.store/books/9781593274078

```
class StickFigureSprite(Sprite):
    def __init__(self, game):
        Sprite.__init__(self, game)
        self.images_left = [
            PhotoImage(file="figure-L1.gif"),
            PhotoImage(file="figure-L2.gif"),
            PhotoImage(file="figure-L3.gif")
        ]
        self.images_right = [
            PhotoImage(file="figure-R1.gif"),
            PhotoImage(file="figure-R2.gif"),
            PhotoImage(file="figure-R3.gif")
        ]
        self.image = game.canvas.create_image(200, 470, \
                image=self.images_left[0], anchor='nw')
        self.x = -2
        self.y = 0
        self.current_image = 0
        self.current_image_add = 1
        self.jump_count = 0
        self.last_time = time.time()
        self.coordinates = Coords()
        game.canvas.bind_all('<KeyPress-Left>', self.turn_left)
        game.canvas.bind_all('<KeyPress-Right>', self.turn_right)
        game.canvas.bind_all('<space>', self.jump)

    def turn_left(self, evt):
        if self.y == 0:
            self.x = -2

    def turn_right(self, evt):
        if self.y == 0:
            self.x = 2

    def jump(self, evt):
        if self.y == 0:
            self.y = -4
            self.jump_count = 0

    def animate(self):
        if self.x != 0 and self.y == 0:
            if time.time() - self.last_time > 0.1:
                self.last_time = time.time()
                self.current_image += self.current_image_add
                if self.current_image >= 2:
                    self.current_image_add = -1
                if self.current_image <= 0:
                    self.current_image_add = 1
```


www.itbook.store/books/9781593274078

```
        if self.x < 0:
            if self.y != 0:
                self.game.canvas.itemconfig(self.image, \
                        image=self.images_left[2])
            else:
                self.game.canvas.itemconfig(self.image, \
                        image=self.images_left[self.current_image])
        elif self.x > 0:
            if self.y != 0:
                self.game.canvas.itemconfig(self.image, \
                        image=self.images_right[2])
            else:
                self.game.canvas.itemconfig(self.image, \
                        image=self.images_right[self.current_image])

    def coords(self):
        xy = self.game.canvas.coords(self.image)
        self.coordinates.x1 = xy[0]
        self.coordinates.y1 = xy[1]
        self.coordinates.x2 = xy[0] + 27
        self.coordinates.y2 = xy[1] + 30
        return self.coordinates

    def move(self):
        self.animate()
        if self.y < 0:
            self.jump_count += 1
            if self.jump_count > 20:
                self.y = 4
        if self.y > 0:
            self.jump_count -= 1
        co = self.coords()
        left = True
        right = True
        top = True
        bottom = True
        falling = True
        if self.y > 0 and co.y2 >= self.game.canvas_height:
            self.y = 0
            bottom = False
        elif self.y < 0 and co.y1 <= 0:
            self.y = 0
            top = False
        if self.x > 0 and co.x2 >= self.game.canvas_width:
            self.x = 0
            right = False
```


www.itbook.store/books/9781593274078

```
            elif self.x < 0 and co.x1 <= 0:
                self.x = 0
                left = False
            for sprite in self.game.sprites:
                if sprite == self:
                    continue
                sprite_co = sprite.coords()
                if top and self.y < 0 and collided_top(co, sprite_co):
                    self.y = -self.y
                    top = False
                if bottom and self.y > 0 and collided_bottom(self.y, \
                        co, sprite_co):
                    self.y = sprite_co.y1 - co.y2
                    if self.y < 0:
                        self.y = 0
                    bottom = False
                    top = False
                if bottom and falling and self.y == 0 \
                        and co.y2 < self.game.canvas_height \
                        and collided_bottom(1, co, sprite_co):
                    falling = False
                if left and self.x < 0 and collided_left(co, sprite_co):
                    self.x = 0
                    left = False
                    if sprite.endgame:
                        self.end(sprite)
                if right and self.x > 0 \
                        and collided_right(co, sprite_co):
                    self.x = 0
                    right = False
                    if sprite.endgame:
                        self.end(sprite)
            if falling and bottom and self.y == 0 \
                    and co.y2 < self.game.canvas_height:
                self.y = 4
            self.game.canvas.move(self.image, self.x, self.y)

        def end(self, sprite):
            self.game.running = False
            sprite.opendoor()
            time.sleep(1)
            self.game.canvas.itemconfig(self.image, state='hidden')
            sprite.closedoor()
```


www.itbook.store/books/9781593274078

```
g = Game()
platform1 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
        0, 480, 100, 10)
platform2 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
        150, 440, 100, 10)
platform3 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
        300, 400, 100, 10)
platform4 = PlatformSprite(g, PhotoImage(file="platform1.gif"), \
        300, 160, 100, 10)
platform5 = MovingPlatformSprite(g, PhotoImage(file="platform2.gif"),\
        175, 350, 66, 10)
platform6 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        50, 300, 66, 10)
platform7 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        170, 120, 66, 10)
platform8 = PlatformSprite(g, PhotoImage(file="platform2.gif"), \
        45, 60, 66, 10)
platform9 = MovingPlatformSprite(g, PhotoImage(file="platform3.gif"),\
        170, 250, 32, 10)
platform10 = PlatformSprite(g, PhotoImage(file="platform3.gif"), \
        230, 200, 32, 10)
g.sprites.append(platform1)
g.sprites.append(platform2)
g.sprites.append(platform3)
g.sprites.append(platform4)
g.sprites.append(platform5)
g.sprites.append(platform6)
g.sprites.append(platform7)
g.sprites.append(platform8)
g.sprites.append(platform9)
g.sprites.append(platform10)
door = DoorSprite(g, 45, 30, 40, 35)
g.sprites.append(door)
sf = StickFigureSprite(g)
g.sprites.append(sf)
g.mainloop()
```


www.itbook.store/books/9781593274078