

PYTHON FOR KIDS

SOLUTIONS TO PROGRAMMING PUZZLES

by Jason R. Briggs



**no starch
press**

San Francisco

PYTHON FOR KIDS, 2ND EDITION. SOLUTIONS TO PROGRAMMING PUZZLES.

Copyright © 2023 by Jason R. Briggs.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Publisher: William Pollock

Managing Editor: Jill Franklin

Production Editor: Paula Williamson

Developmental Editors: William Pollock, Eva Morrow, Jill Franklin

Cover and Interior Design: Octopod Studios

Illustrator: Miran Lipovača

For information on distribution, bulk sales, corporate sales, or translations, please contact No Starch Press, Inc. directly at info@nostarch.com or:

No Starch Press, Inc.

245 8th Street, San Francisco, CA 94103

phone: 415.863.9900

www.nostarch.com

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The information in this book is distributed on an “As Is” basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

CONTENTS IN DETAIL

INTRODUCTION IX

3 1 STRINGS, LISTS, TUPLES, AND DICTIONARIES

#1: Favorites.....	1
#2: Counting Combatants.....	2
#3: Greetings!	2
#4: Multiline Letter.....	3

4 5 DRAWING WITH TURTLES

#1: A Rectangle.....	5
#2: A Triangle	6
#3: A Box Without Corners	9
#4: A Tilted Box Without Corners	10

5 13 ASKING QUESTIONS WITH IF AND ELSE

#1: Are You Rich?	13
#2: Twinkies!	14
#3: Just the Right Number	14
#4: I Can Fight Those Ninjas.....	15

6 17 GOING LOOPY

#1: The Hello Loop.....	17
#2: Even Numbers	18
#3: My Five Favorite Ingredients	18
#4: Your Weight on the Moon	19

7 21 RECYCLING YOUR CODE WITH FUNCTIONS AND MODULES

#1: Basic Moon Weight Function	21
#2: Moon Weight Function and Years.....	22

#3: Moon Weight Program	23
#4: Mars Weight Program	24

8
HOW TO USE CLASSES AND OBJECTS **25**

#1: The Giraffe Shuffle	25
#2: Turtle Pitchfork	26
#3: Two Small Spirals	27
#4: Four Small Spirals	29

9
MORE TURTLE GRAPHICS **33**

#1: Drawing an Octagon	33
#2: Drawing a Filled Octagon	34
#3: Another Star-Drawing Function	35
#4: Four Spirals Revisited	36

10
USING TKINTER FOR BETTER GRAPHICS **39**

#1: Fill the Screen with Triangles	39
#2: The Moving Triangle	41
#3: The Moving Photo	42
#4: Fill the Screen with Photos	43

11
BEGINNING YOUR FIRST GAME: BOUNCE! **45**

#1: Changing colors	45
#2: Flashing colors	47
#3: Take your positions!	47
#4: Adding the Paddle . . . ?	48

12
FINISHING YOUR FIRST GAME: BOUNCE! **49**

#1: Delay the Game Start	49
#2: A Proper “Game Over”	50
#3: Accelerate the Ball	51
#4: Record the Player’s Score	52

14		
DEVELOPING THE MR. STICK MAN GAME		57
#1: Checkerboard		57
#2: Two-Image Checkerboard		58
#3: Bookshelf and Lamp		59
#4: Random Background		60

16		
COMPLETING THE MR. STICK MAN GAME		63
#1: "You Win!"		63
#2: Animating the Door		64
#3: Moving Platforms.....		66
#4: Lamp as a sprite		68



Here you'll find the solutions to the programming puzzles found at the end of each chapter in *Python for Kids*. Oftentimes, these puzzles will have multiple solutions; as long as your solution works, there's no need to match what is shown in this supplement. These examples will give you an idea of possible approaches, particularly if you find yourself stuck.



3

STRINGS, LISTS, TUPLES, AND DICTIONARIES

#1: FAVORITES

Make a list of your favorite hobbies and give the list the variable name `games`. Now make a list of your favorite foods and name the variable `foods`. Join the two lists and name the result `favorites`. Lastly, print the variable `favorites`.

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

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

```
['Pokemon', 'LEGO MINDSTORMS', 'Mountain Biking', 'Pancakes',
'Chocolate', 'Apples']
```

#2: COUNTING COMBATANTS

If there are three buildings with 25 ninjas hiding on each roof and two tunnels with 40 samurai hiding inside each tunnel, how many ninjas and samurai are about to do battle?

We can do the calculation for this puzzle in a number of ways. Since we know that there are three buildings with 25 ninjas on each roof, and two tunnels with 40 samurai in each, we could calculate the total number of ninjas and the total number of samurai, then add the two totals like this:

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

We could choose to combine these three equations by using parentheses to determine the order of operations, make our equation easier to read, and simplify things overall, like so:

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

Or we could name our variables, which tells us, and anyone who reads our code, 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
```

This is one of the best ways to calculate this equation, as our variables are clearly labeled.

#3: GREETINGS!

Create two variables: one that points to your first name and one that points to your last name. Now create a string and use placeholders to print your name with a message using those two variables, such as "Hi there, Brando Ickett!"

We'll give the variables meaningful names, then use format placeholders (`{}` `{}`) in our string to embed the variable values:

```
>>> first_name = 'Brando'
>>> last_name = 'Ickett'
```

```
>>> print(f'Hi there, {first_name} {last_name}!')
```

```
Hi there, Brando Ickett!
```

#4: MULTILINE LETTER

Take the letter we created earlier in the chapter and try to print the exact same text by using a single print call (and a multiline string).

To print the letter as a multiline string, we can either format the letter manually, line by line, or use the spaces variable.

Here's what the result would look like if we formatted by hand:

```
print(''                12 Butts Wynd
      ''                Twinklebottom Heath
      ''                West Snoring
```

```
Dear Sir
```

```
I wish to report that tiles are missing from the outside toilet roof.
I think it was bad wind the other night that blew them away.
```

```
Regards
Malcolm Dithering''')
```

Using the spaces variable, our result looks like the following:

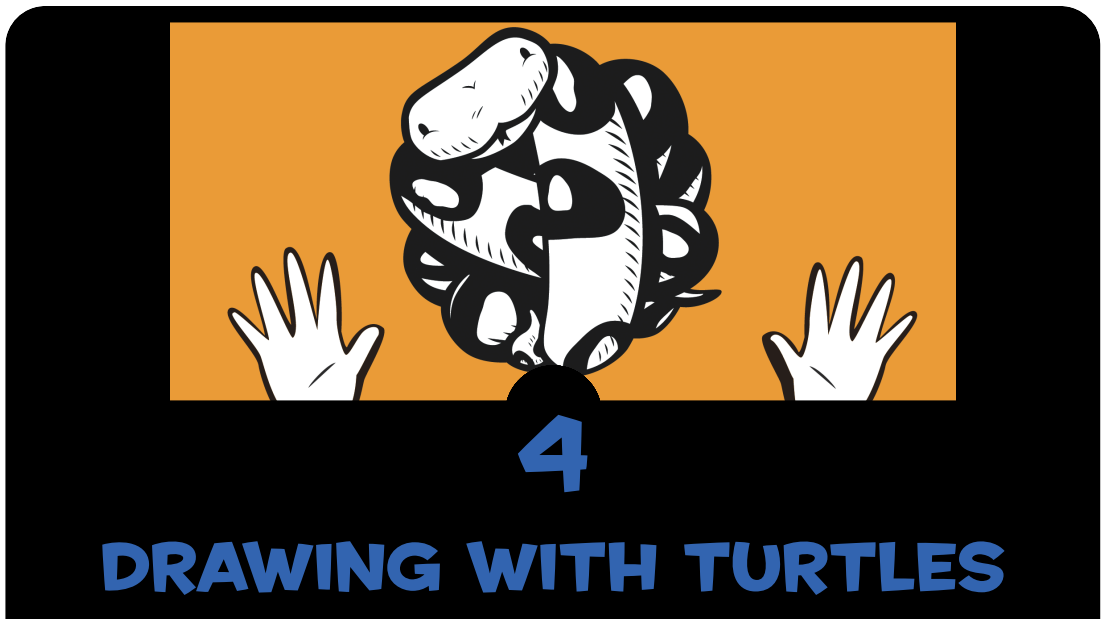
```
spaces = ' ' * 25
print(f''{spaces}12 Butts Wynd
      {spaces}Twinklebottom Heath
      {spaces}West Snoring
```

```
Dear Sir
```

```
I wish to report that tiles are missing from the outside toilet roof.
I think it was bad wind the other night that blew them away.
```

```
Regards
Malcolm Dithering''')
```

On the first line we create the spaces variable with 25 spaces by multiplying the space character (' ') by 25. Next, we print the multiline string using the spaces variable ({spaces}) at the start of each line of the letterhead so that the words are correctly indented.



#1: A RECTANGLE

Create a new canvas using the turtle module's Turtle function and then draw a rectangle.

Drawing a rectangle is similar to drawing a square, except that two parallel sides should be 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

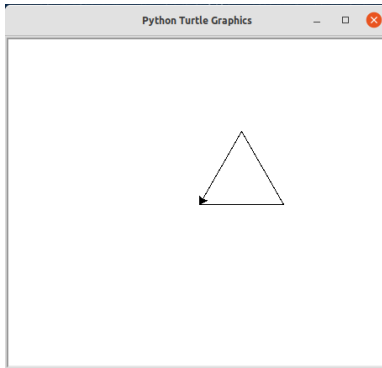
Create another canvas and draw a triangle.

The puzzle didn't specify what kind of triangle to draw. There are three types of triangles: equilateral, isosceles, and scalene. For this example, we'll 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)
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 ❶. We turn left 120 degrees (which creates an interior angle of 60 degrees) ❷, and move forward 100 pixels again ❸. The next turn is also 120 degrees ❹, and the turtle moves back to the starting position by moving forward another 100 pixels ❺. Here's the result of running the code:



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

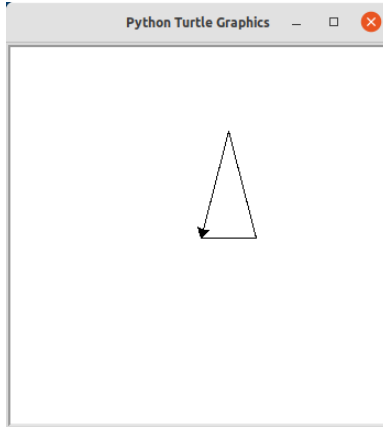
```
>>> import turtle
>>> t = turtle.Pen()
>>> t.forward(50)
>>> t.left(104.47751218592992)
>>> t.forward(100)
>>> t.left(151.04497562814015)
>>> t.forward(100)
```

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)
```

Here's the result of running this code:

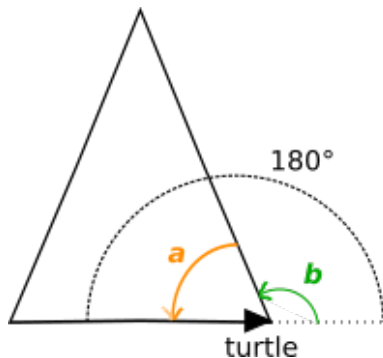


NOTE

The remainder of this solution uses more complicated mathematics equations and Python code that we'll learn about later in the book. You might like to come back to this once you've finished reading Python for Kids.

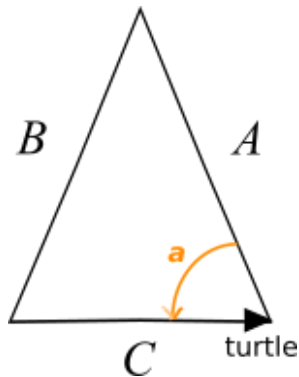
How do we come up with angles like 104.47751218592992 degrees and 151.04497562814015 degrees? After all, those are rather obscure numbers! Once we've decided on the lengths of each of the sides of the triangle, we can calculate the interior angles using Python and a bit of trigonometry.

The following diagram shows 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* should add up to 180 degrees.



Once we know the right equation to use, we can calculate the inside angle. 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.



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

$$\alpha = \arccos\left(\frac{A^2 + C^2 - B^2}{2AC}\right) \quad (4.1)$$

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

```

❶ >>> import math
    >>> A = 100
    >>> B = 100
    >>> C = 50
❷ >>> 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 ❶ and create variables for each of the sides (A, B, and C). We then use the `math` function `acos` (*arc cosine*) to calculate the angle ❷. This calculation returns the radians value 1.31811607165. *Radians* is another unit used to measure angles, like degrees.

NOTE

The backslash (`\`) at the end of the line at ❷ isn't part of the equation—backslashes are used to separate long lines of code. They're not always necessary, but in this case we're splitting a long line because it won't fit on the page otherwise.

The radians value can be converted into degrees using the `math` function `degrees`. We can calculate the outside angle (the 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:

$$\alpha = \arccos\left(\frac{A^2 + B^2 - C^2}{2AB}\right) \quad (4.2)$$

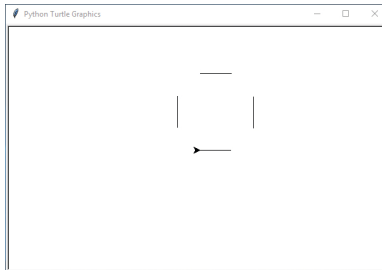
The code for this equation also looks similar to what we wrote before:

```
>>> 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
```

You don't need to use equations to work out the angles, especially if you haven't learned these topics in school yet. You can also just turn the turtle various degrees until you get something that looks about right.

#3: A BOX WITHOUT CORNERS

Write a program to draw the four lines shown below (the size isn't important, just the shape).



To create the solution—an octagon missing four sides—we'll tell the turtle 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:

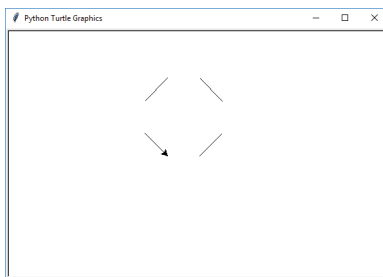
```
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
```

The final set of commands should be similar to the following code. Create a new window in the Shell, and then save the file as *nocorners.py*:

```
import turtle
t = turtle.Pen()
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)
t.forward(50)
t.left(45)
t.up()
t.forward(50)
t.down()
t.left(45)
```

#4: A TILTED BOX WITHOUT CORNERS

Write a program to draw the four lines shown below (similar to the previous puzzle, but the box is tilted on its side). Again, the size of the box isn't important—just the shape.



The code to create a titled box without corners is very similar to the previous puzzle. The path that the turtle takes is almost exactly the same, with the difference being when you call `t.up()` and `t.down()`:

```
import turtle
t = turtle.Pen()
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)
t.forward(50)
t.left(45)
t.up()
```



5

ASKING QUESTIONS WITH IF AND ELSE

#1: ARE YOU RICH?

What do you think the following code will do? Try to figure out the answer without typing it into the Python Shell and then check your work.

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

With this code, you'll get an indentation error once you reach the last line of the if statement:

```
>>> money = 2000
>>> if money > 1000:
    ❶ print("I'm rich!!")
```

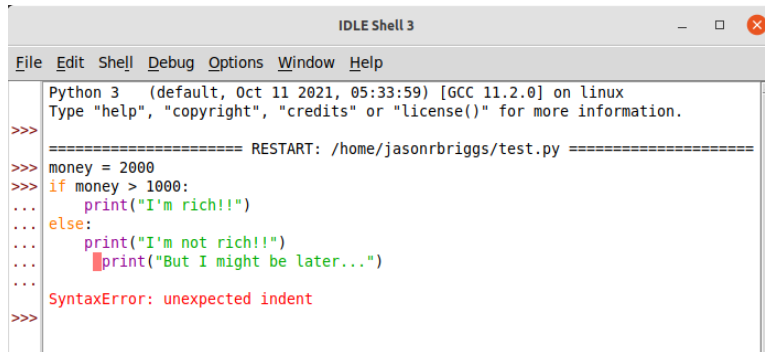
```

else:
    ❷ print("I'm not rich!!")
    ❸ print("But I might be later...")

```

SyntaxError: unexpected indent

This error occurs because the blocks of code at ❶ and ❷ start with four spaces, so Python doesn't expect to see two extra spaces on the final line ❸. IDLE highlights where it sees a problem with a red rectangular block so you can see where you went wrong:



```

IDLE Shell 3
File Edit Shell! Debug Options Window Help
Python 3 (default, Oct 11 2021, 05:33:59) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /home/jasonbriggs/test.py =====
>>> money = 2000
>>> if money > 1000:
...     print("I'm rich!!")
... else:
...     print("I'm not rich!!")
...     print("But I might be later...")
...
SyntaxError: unexpected indent
>>>

```

#2: TWINKIES!

Create an if statement that checks whether a number of Twinkies (in the variable `twinkies`) is less than 100 or greater than 500. Your program should print the message “Too few or too many” if the condition is True.

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

Create an if statement that checks whether the amount of money contained in the `money` variable is between 100 and 500 or between 1,000 and 5,000.

You could write more than one if statement to check whether an amount of money falls between 100 and 500 or 1,000 and 5,000, 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 1,000 and 5,000.

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
```

#4: I CAN FIGHT THOSE NINJAS

Create an if statement that prints “That’s too many” if the variable ninjas contains a number less than 50; prints “It’ll be a struggle, but I can take ‘em” if it’s less than 30; and prints “I can fight those ninjas!” if it’s less than 10.

This was a bit of a trick puzzle. If you create the if statement in the same order as the instructions, you won’t get the expected result. For 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 condition (`ninjas < 50`) is evaluated first, and the variable is less than 50, so the program prints the message you didn’t expect to see.

For this code 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!



#1: THE HELLO LOOP

What do you think the following code will do?

```
for x in range(0, 20):  
    print(f'hello {x}')  
    if x < 9:  
        break
```

The print statement in this for loop is run only once. When Python hits the if statement, *x* is less than 9 because the first value in the loop is 0, so it immediately breaks out of the loop.

```
>>> for x in range(0, 20):  
    print(f'hello {x}')  
    if x < 9:  
        break
```

```
hello 0
```

#2: EVEN NUMBERS

Create a loop that prints even numbers until it reaches your age (if your age is an odd number, create a loop that prints out odd numbers until it reaches your age).

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

Create a list containing five different sandwich ingredients, such as the following:

```
>>> ingredients = ['snails', 'leeches', 'gorilla belly-button lint',
                  'caterpillar eyebrows', 'centipede toes']
```

Now create a loop that prints out the list (including the numbers):

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

There are a couple of ways to print numbers alongside the items in your list. For example:

```
>>> ingredients = ['snails', 'leeches', 'gorilla belly-button lint',
                  'caterpillar eyebrows', 'centipede toes']

❶ >>> x = 1
❷ >>> for i in ingredients:
    ❸ print(f'{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 ❶. Next, we create a for loop to loop through the items in the list ❷, assigning each to the variable `i`, and we print the value of the `x` and `i` variables ❸ using the `{}` placeholder. We add 1 to the `x` variable ❹ so that each time we loop, the number we print increases.

#4: YOUR WEIGHT ON THE MOON

If you were standing on the moon right now, your weight would be 16.5 percent of what it is on Earth. You can calculate that by multiplying your Earth weight by 0.165. If you gained two pounds every year for the next 15 years, what would your weight be when you visited the moon each year and at the end of the 15 years? Write a program using a for loop that prints your moon weight for each year.

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):

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



7

RECYCLING YOUR CODE WITH FUNCTIONS AND MODULES

#1: BASIC MOON WEIGHT FUNCTION

We created a for loop to determine your weight on the moon over a period of 15 years. That for loop could easily be turned into a function. Try creating a function that takes a starting weight and increases its amount each year.

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 Project #4: Your Weight on the Moon in Chapter 6.

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

Take the function you've just created and change it to work out the weight over different periods, such as 5 years or 20 years. Be sure to change the function so it takes three arguments: the initial weight, the weight gained each year, and the number of years.

We'll change the previous function slightly 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(f'Year {year} is {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
```

On the second line of the function, 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

Instead of using a simple function and passing in the values as parameters, you can use `sys.stdin.readline()` or `input()` to make a mini-program that prompts for the values. In this case, you call the function without any parameters at all:

```
>>> moon_weight()
```

The function will display a message asking for the starting weight, a second message asking for the amount the weight will increase each year, and a final message asking for the number of years.

We can use the `stdin` object in 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.

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

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

If we call the function now, we should 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

```

#4: MARS WEIGHT PROGRAM

Let's change our moon weight program to calculate weights on Mars—only this time, for your entire family. The function should ask for each family member's weight, calculate how much they would weigh on Mars (by multiplying the number by 0.3782), and then add up and display the total weight at the end.

We could write this program a few different ways. For example, we could have a function that takes the number of family members as a parameter, loops to get the weight of that number of people, and calculates the total weight on Mars:

```

import sys
def mars_weight(number_in_family):
    ❶ total_mars_weight = 0
    ❷ for x in range(0, number_in_family):
        print("Please enter a family member's Earth weight")
        ❸ weight = float(input())
        ❹ total_mars_weight = total_mars_weight + (weight * 0.3782)
    ❺ print(f'The total weight of your family on Mars is {total_mars_weight}')

mars_weight(3)

```

We create a variable `total_mars_weight` and set the initial value as 0 ❶ (we'll add each weight we calculate to this). We loop using the range function ❷ and then read the weight using `input()` ❸. We calculate the Mars weight and add it to the total ❹. Finally, we print out our total weight value ❺.

If your code doesn't look exactly like this, that's fine—what's important is that it works the way you want it to!



8

HOW TO USE CLASSES AND OBJECTS

#1: THE GIRAFFE SHUFFLE

Add functions to the Giraffe class to move the giraffe's left and right feet forward and backward. Then create a function called dance to teach our giraffes to dance (the function will call the four foot functions you've just created).

Before adding the functions to make Harriet dance a jig, let's take another look at the Giraffes class:

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

We can add the functions for moving each foot like so:

```
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')
```

The dance function 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()
```

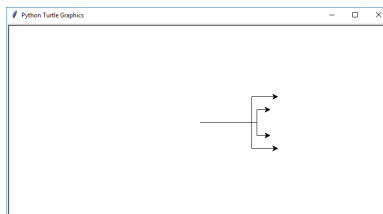
To make Harriet dance, we create an object and call the function:

```
>>> harriet = Giraffes()
>>> harriet.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

Create the following picture of a sideways pitchfork using four Turtle objects (the exact length of the lines isn't important).



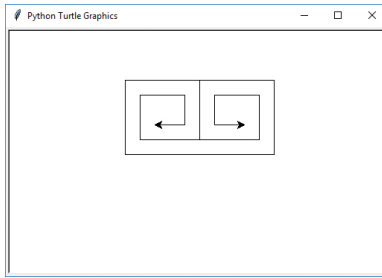
Pen is a class defined in the turtle module, so we can create more than one object of the Pen class for each of the four turtles. If we assign each object to a different variable, we can control them separately, which makes it easier to reproduce the arrowed lines in this puzzle. Remember that even though the class is the same, when you create a new *instance* of the class (in this case, using `turtle.Pen()`), you get a new object each time.

```
import turtle
t1 = turtle.Pen()
t2 = turtle.Pen()
t3 = turtle.Pen()
t4 = turtle.Pen()
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 this, so your code may not look exactly like mine.

#3: TWO SMALL SPIRALS

Create the following picture of two small spirals using Turtle objects (again, the exact size of the spirals isn't important).



Similar to the previous puzzle, we create two Pen objects and then move them independently. Each spiral is a move forward, then a turn, then move forward again—but we gradually reduce the amount we move forward to produce the pattern:

```
import turtle

t1 = turtle.Pen()
t2 = turtle.Pen()

t1.forward(100)
t1.left(90)
t1.forward(100)
t1.left(90)
t1.forward(100)
t1.left(90)
t1.forward(80)
t1.left(90)
t1.forward(80)
t1.left(90)
t1.forward(60)
t1.left(90)
t1.forward(60)
t1.left(90)
t1.forward(40)
t1.left(90)
t1.forward(40)

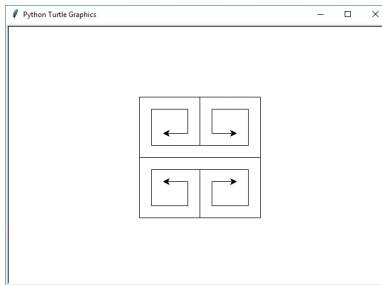
❶ t2.right(180)
t2.forward(100)
t2.right(90)
t2.forward(100)
t2.right(90)
t2.forward(100)
t2.right(90)
t2.forward(80)
t2.right(90)
t2.forward(80)
```

```
t2.right(90)
t2.forward(60)
t2.right(90)
t2.forward(60)
t2.right(90)
t2.forward(40)
t2.right(90)
t2.forward(40)
```

We make one spiral mirror the other by turning the turtle the opposite direction using `t2.right(180)` ❶.

#4: FOUR SMALL SPIRALS

Let's take the two spirals we created in the previous code and make a mirror image to create four spirals, which should look like the following image.



The code for four spirals is similar to the previous puzzle, except we create four Pen objects and turn each turtle different directions:

```
import time
import turtle

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

t1.forward(100)
t1.left(90)
t1.forward(100)
t1.left(90)
t1.forward(100)
t1.left(90)
t1.forward(80)
t1.left(90)
t1.forward(80)
```

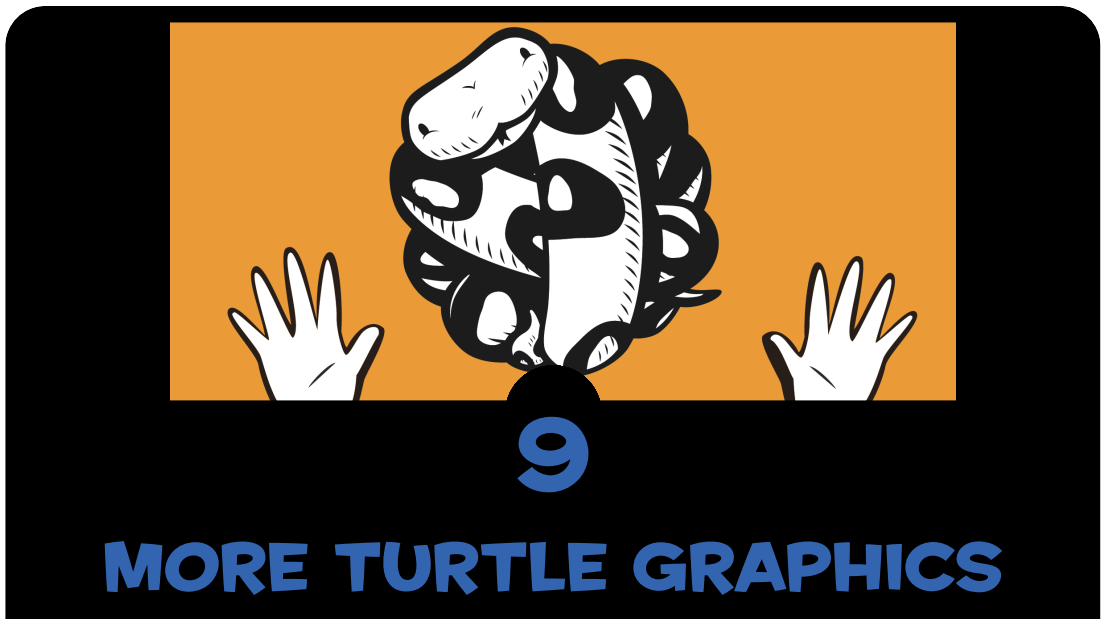
```
t1.left(90)
t1.forward(60)
t1.left(90)
t1.forward(60)
t1.left(90)
t1.forward(40)
t1.left(90)
t1.forward(40)
```

```
t2.right(180)
t2.forward(100)
t2.right(90)
t2.forward(100)
t2.right(90)
t2.forward(100)
t2.right(90)
t2.forward(80)
t2.right(90)
t2.forward(80)
t2.right(90)
t2.forward(60)
t2.right(90)
t2.forward(60)
t2.right(90)
t2.forward(40)
t2.right(90)
t2.forward(40)
```

```
t3.forward(100)
t3.right(90)
t3.forward(100)
t3.right(90)
t3.forward(100)
t3.right(90)
t3.forward(80)
t3.right(90)
t3.forward(80)
t3.right(90)
t3.forward(60)
t3.right(90)
t3.forward(60)
t3.right(90)
t3.forward(40)
t3.right(90)
t3.forward(40)
```

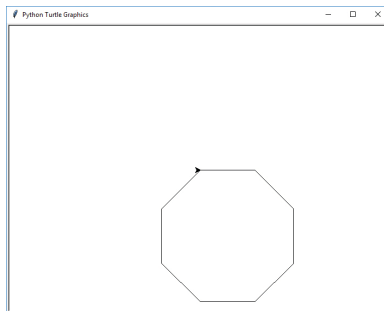


```
t4.left(180)
t4.forward(100)
t4.left(90)
t4.forward(100)
t4.left(90)
t4.forward(100)
t4.left(90)
t4.forward(80)
t4.left(90)
t4.forward(80)
t4.left(90)
t4.forward(60)
t4.left(90)
t4.forward(60)
t4.left(90)
t4.forward(40)
t4.left(90)
t4.forward(40)
```



#1: DRAWING AN OCTAGON

We've drawn stars, squares, and rectangles in this chapter. How about creating a function to draw an eight-sided shape like an octagon? Your shape should look similar to the image below.



An octagon has eight sides, so we'll need a for loop to complete this drawing. To draw an octagon, the arrow of 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 an 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).

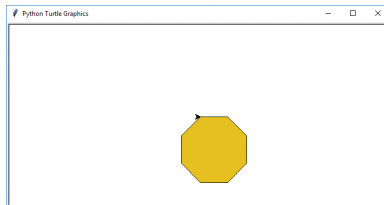
```
>>> import turtle
>>> t = turtle.Pen()
>>> def octagon(size):
    for x in range(1,9):
        t.forward(size)
        t.right(45)
```

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

Now that you have a function to draw an octagon, modify it to draw a filled octagon. Try drawing an octagon with an outline, as we did with the star. It should look similar to the image below.



If we change the previous function so that it draws a filled octagon, we won't be able to draw the outline (unless we create a copy of the function—one for the outline and one for filling it in—which isn't good programming practice). Instead, we should pass in a parameter to control whether the octagon should be filled.

```
>>> 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 ❶. If it is, we tell the turtle to start filling using the `begin_fill` function ❷. We then draw the octagon on the next two lines, in the same way as Puzzle #1: Drawing an Octagon, and then check to see if the filled parameter is True ❸. If it is, we call the `end_fill` function ❹, which 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 the shape). We can then set the color back to black, and call the function again with the parameter set to False to create the outline.

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

#3: ANOTHER STAR-DRAWING FUNCTION

Create a function to draw a star that will take two parameters: the size and number of points.

To create this star function, we'll divide 360 degrees by the number of points, which provides the interior angle for each point of the star (see 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 left ❷.

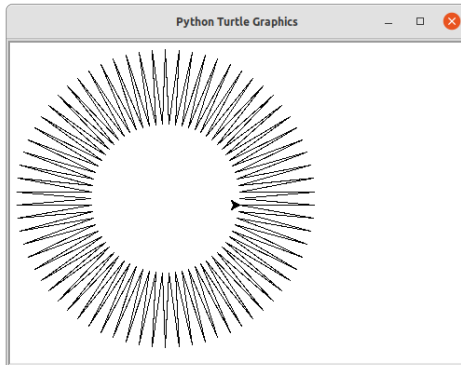
```
import turtle
t = turtle.Pen()
def draw_star(size, points):
    ❶ angle = 360 / points
    ❷ for x in range(0, points):
        ❸ t.forward(size)
        ❹ t.left(180 - angle)
        ❺ t.forward(size)
        ❻ t.right(180 - (angle * 2))
```

We loop from 0 up to the number of points ❷, and then move the turtle forward the number of pixels specified in the size parameter ❸. We turn the turtle the number of degrees we've previously calculated ❹, and then move forward again 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 ❻.

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

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

This should result in the following:



#4: FOUR SPIRALS REVISITED

Take the code you created to create four spirals and draw the same spirals again—only this time, try using for loops and if statements to simplify your code.

The trick to “simplifying” the code in Chapter 8, Project #4: Four Small Spirals, is recognizing that each spiral is roughly the same. Turn, move forward, turn again, move forward, then repeat; but, with each repetition we reduce the amount we move forward. Here’s one attempt at simplification—although you might look at this code and say to yourself, “This doesn’t look simpler to me!”:

```
import time
import turtle

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

t1.forward(100)
t2.right(180)
t2.forward(100)
t3.forward(100)
t4.left(180)
t4.forward(100)

f = 100
for x in range(0, 8):
    t1.left(90)
    t1.forward(f)
```

```

        if x == 1 or x == 3 or x == 5 or x == 7:
            f = f - 20

f = 100
for x in range(0, 8):
    t2.right(90)
    t2.forward(f)
    if x == 1 or x == 3 or x == 5 or x == 7:
        f = f - 20

f = 100
for x in range(0, 8):
    t3.right(90)
    t3.forward(f)
    if x == 1 or x == 3 or x == 5 or x == 7:
        f = f - 20

f = 100
for x in range(0, 8):
    t4.left(90)
    t4.forward(f)
    if x == 1 or x == 3 or x == 5 or x == 7:
        f = f - 20

```

We can rework this code even more by using a function to get something simpler. The first part of the code is the same as the previous example, creating the four turtle objects and then getting them into the correct starting position (and direction):

```

import time
import turtle

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

t1.forward(100)
t2.right(180)
t2.forward(100)
t3.forward(100)
t4.left(180)
t4.forward(100)

```

Then we create our function for drawing the spiral:

```

❶ def spiral(t, left):
    ❷ f = 100
    ❸ for x in range(0, 8):

```

```

④ if left == True:
    ⑤ t.left(90)
    else:
        ⑥ t.right(90)
    ⑦ t.forward(f)
    ⑧ if x % 2 != 0:
        ⑨ f = f - 20

```

```

spiral(t1, True)
spiral(t2, False)
spiral(t3, False)
spiral(t4, True)

```

The definition of our function ❶ takes a turtle parameter called `t` and a second parameter `left`. We set the variable we use for how far the turtle should move (`f`) to 100 to start with ❷. Then we create a for loop to repeat eight times (using the range function) ❸, and we check if value of the `left` variable is `True` ❹. If it is, we turn left ❺, otherwise we turn right ❻. We move the turtle forward the distance in the variable `f` ❼.

Line ❸ needs a little more explanation. In the previous version of the code, we checked if the loop variable `x` is equal to 1, 3, 5, or 7:

```
if x == 1 or x == 3 or x == 5 or x == 7:
```

If the variable contains an odd number, we want to reduce the amount we move forward. In the new version of the code, we check if the number can be divided into 2 with no remainder using the modulo function:

```
if x % 2 != 0:
```

If a number can't be divided by 2 evenly, then it's an odd number. Every time this is true, we subtract 20 from the `f` variable ❹.

The final bit of the code calls the `spiral` function for each of our turtle objects, passing either `True` or `False` for the `left` parameter depending on which way we want the turtle to turn.



10

USING TKINTER FOR BETTER GRAPHICS

#1: FILL THE SCREEN WITH TRIANGLES

Create a program using tkinter to fill the screen with triangles. Then change the code to fill the screen with different-colored (filled) triangles instead.

To fill the screen with triangles, we'll first 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()
```

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 10), 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.

```
>>> 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']
```

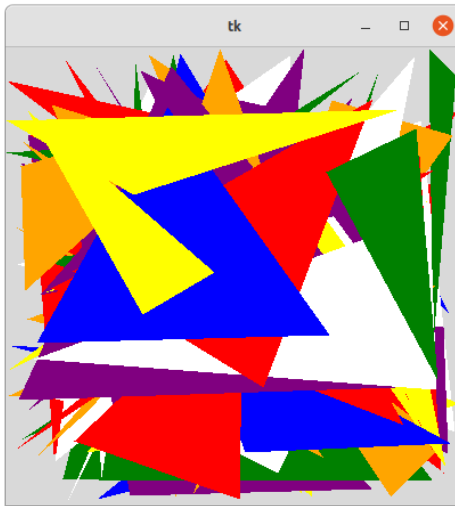
We can then use the choice function of the random module to randomly pick an item from this list of colors, and use it in the call to create_polygon:

```
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='')
```

If we loop 100 times again:

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

The result will be something like these triangles:



#2: THE MOVING TRIANGLE

Modify the code for the moving triangle in the “Creating Basic Animation” section of Chapter 10 to make it move across the screen to the right, then down, then back to the left, and then back to its starting position.

To create a moving triangle, we'll set up the canvas and then draw the triangle using the `create_polygon` function:

```
import time
from tkinter import *
tk = Tk()
canvas = Canvas(tk, width=400, height=200)
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 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 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)
```

#3: THE MOVING PHOTO

Try displaying a photo of yourself on the canvas. Make sure it's a GIF image! Can you make it move across the screen?

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

The following code will move the image diagonally down the screen:

```
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)
```

If you're using Ubuntu, Raspberry Pi, or macOS, the filename of the image will be different. If the file is in your home directory on Ubuntu or Raspberry Pi, 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')
```

#4: FILL THE SCREEN WITH PHOTOS

Take the photo you used in the previous puzzle, and shrink it down small, then fill the screen with lots of copies of your photo.

This is very similar code to filling the screen with random triangles, but rather than creating a random triangle in a random position on the screen, we want the image to stay the same size and then “stamp” it in random positions around the screen.

First, create the canvas as in the other examples:

```
from tkinter import *
import time
import random

tk = Tk()
canvas = Canvas(tk, width=400, height=400)
canvas.pack()
```

Then we'll load our image, loop 100 times, and generate a random number between 0 and 400 for both *x* and *y* coordinates to place the image:

```
img = PhotoImage(file='small-image.gif')

for x in range(0,100):
    p1 = random.randrange(400)
    p2 = random.randrange(400)
    canvas.create_image(p1, p2, anchor=NW, image=img)
    tk.update()
    time.sleep(0.5)
```

Every time we place the image, we call `tk.update()` to redraw the screen. For every loop, we sleep for half a second with `time.sleep(0.5)` so it takes some time to fill the screen.



11

BEGINNING YOUR FIRST GAME: BOUNCE!

#1: CHANGING COLORS

Try changing the starting color of the ball and the background color of the canvas—try a few different combinations of colors and see which ones you like.

Our Ball class already takes a color parameter:

```
class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
```

So we can just create our ball object passing a different color; let's try green:

```
ball = Ball(canvas, 'green')
```

Changing the background of the canvas is a little more difficult, but with a bit of guesswork, we can figure it out. The `create_oval` function has a `fill` parameter, so let's try using the same parameter when creating the canvas:

```
from tkinter import *

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, fill='blue')
canvas.pack()
tk.update()
```

```
Traceback (most recent call last):
  File "/usr/lib/python3.10/idlelib/run.py", line 573,
    in runcode
    exec(code, self.locals)
  File "/home/jasonrbriggs/test.py", line 7, in <module>
    canvas = Canvas(tk, width=500, height=400, bd=0,
              highlightthickness=0, fill='blue')
  File "/usr/lib/python3.10/tkinter/_init_.py",
    line 2717, in __init__
    Widget.__init__(self, master, 'canvas', cnf, kw)
  File "/usr/lib/python3.10/tkinter/_init_.py",
    line 2601, in __init__
    self.tk.call(
_tkinter.TclError: unknown option "-fill"
```

Well, that didn't work. Let's try something else. We want to set the background color of the canvas, so let's try using the background parameter instead:

```
from tkinter import *

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, background='blue')
canvas.pack()
tk.update()
```

That works fine! Sometimes experimenting with a small amount of code is the easiest way to figure out how to do something.

#2: FLASHING COLORS

Because there's a loop at the bottom of our code, it should be quite easy to change the color of the ball as it moves across the screen. We can add some code to the loop that picks different colors (think about the choice function we used earlier in the chapter), and then updates the color of the ball (perhaps by calling a new function on our Ball class).

To get a random color, we'll use the random module and the choice function. We can test this with a list of colors:

```
>>> random.choice(['yellow', 'red', 'blue', 'green', 'orange', 'black', 'white'])
'green'
>>> random.choice(['yellow', 'red', 'blue', 'green', 'orange', 'black', 'white'])
'red'
```

We can then use the itemconfig function in the draw function of our Ball class to change the color. We use the id of the oval we created in the `__init__` function, and use random.choice to pick a random color for the fill parameter:

```
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.y = -3
    if pos[0] <= 0:
        self.x = 3
    if pos[2] >= self.canvas_width:
        self.x = -3
    self.canvas.itemconfig(ball.id, fill=random.choice(['yellow', 'red',
        'blue', 'green', 'orange', 'black', 'white']))
```

#3: TAKE YOUR POSITIONS!

Try to change the code so the ball starts in a different position on the screen. You could make the position random by using the random module. But you'll have to ensure the ball doesn't start too close, or below, the paddle, which will make the game impossible to play.

One way to create a random starting position for your ball would be to only consider a random position in the top half of the screen. Currently, our ball starts at position (245, 100):

```
self.canvas.move(self.id, 245, 100)
```

That's 245 pixels across the screen horizontally, and 100 pixels down vertically. In Chapter 12, we're going to position our paddle 200 pixels across and 300 down, so we don't want the ball to appear below 300 pixels. Perhaps down to 200 pixels is a good area to start with? Our random range will include anything horizontally from 0 (the leftmost side of the canvas) and 485 pixels (the canvas is 500 pixels wide, our ball is 15 pixels wide; $500 - 15$ is 485 pixels). The random range for the vertical position will include anything from 0 pixels (the topmost side of the canvas) and 200. We don't need to subtract anything from the height because we're not worried about the ball appearing off the bottom of the screen in this case.

This code will provide random start positions as the variables `random_x` and `random_y`:

```
random_x = random.randrange(0, 485)
random_y = random.randrange(0, 200)
```

The change to move is as follows:

```
self.canvas.move(self.id, random_x, random_y)
```

The `__init__` function of our Ball class now looks like this:

```
def __init__(self, canvas, color):
    self.canvas = canvas
    self.id = canvas.create_oval(10, 10, 25, 25,
                                 fill=color)
    random_x = random.randrange(0, 485)
    random_y = random.randrange(0, 200)
    self.canvas.move(self.id, random_x, random_y)
    starts = [-3, -2, -1, 1, 2, 3]
    self.x = random.choice(starts)
    self.y = -3
    self.canvas_height = self.canvas.wininfo_height()
    self.canvas_width = self.canvas.wininfo_width()
    self.counter = 0
```

#4: ADDING THE PADDLE . . . ?

Based on the code we've created so far, can you figure out how to add the paddle before reaching the next chapter?

You can find the answer to this puzzle by reading Chapter 12 of *Python for Kids*.



12

FINISHING YOUR FIRST GAME: BOUNCE!

#1: DELAY THE GAME START

Our game starts quickly, and you need to click the canvas before it will recognize pressing the left and right arrow keys on your keyboard. Can you add a delay to the start of the game in order to give the player enough time to click the canvas? Or even better, can you add an event binding for a mouse click, which starts the game only then?

To start the game when the player clicks the canvas, we need to make a couple of small changes to the program. First, we'll add a new function to the Paddle class:

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

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

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 that binds 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` ❶, and the binding for the mouse button 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, as shown 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"

Everything freezes when the game ends, which isn't very player-friendly. Try adding the text "Game Over" when the ball hits the bottom of the screen. As an additional challenge, add a delay so that the text doesn't appear right away.

We'll use the `create_text` function to create the "Game Over" text. Add the following directly after the code that creates 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')
```

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:
        time.sleep(1)
        ❷ 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` ❶. If it is, we sleep for 1 second (to give a short delay before displaying the text), and then change the state parameter of the text to 'normal' rather than 'hidden' ❷ by 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

In tennis, when a ball hits your racket, it sometimes flies away faster than the speed at which it arrived, depending on how hard you swing. The ball in our game goes at the same speed, whether or not the paddle is moving. Try changing the program so that the paddle's speed is passed on to the speed of the ball.

It's a bit difficult to figure out where in the code to make this change. We want the ball to speed up if it's traveling in the same horizontal direction when it hits the paddle, and slow down if it's going in the opposite horizontal direction of the paddle. In order to do this, the left-right (horizontal) speed of the paddle should be added to the horizontal speed of the ball.

The best place to make this change is in the `hit_paddle` function of the `Ball` class, because that's where we can change the `x` variable that represents the horizontal speed of our ball:

```
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
```

Once we've determined that the ball has hit the paddle ❶, we add the value of the `x` variable of the paddle object to the `x` variable of the

ball 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 the new speed when it hits the paddle.

#4: RECORD THE PLAYER'S SCORE

How about recording the score? Every time the ball hits the paddle, the score should increase. Try displaying the score at the top-right corner of the canvas.

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 ❶. We also store the `canvas` parameter to use later as the object variable `canvas` 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 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 display 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)

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

The `hit` function takes no parameters (just `self`), and increases the score by 1 ❶ before using the `itemconfig` function of the `canvas` object to change the text displayed to the new score value ❷.

We can create an object of the `Score` class right 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
```

Once all four puzzles are completed, the full game code should look like this:

```
from tkinter import *
import random
import time

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
        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')

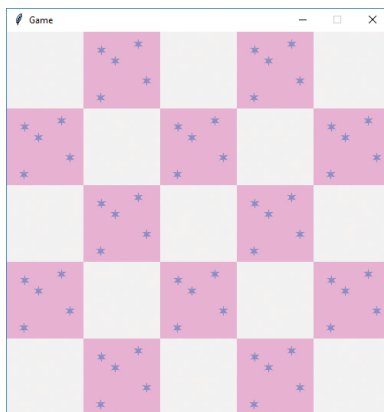
while 1:
    if ball.hit_bottom == False and paddle.started == True:
        ball.draw()
        paddle.draw()
    if ball.hit_bottom == True:
        time.sleep(1)
        canvas.itemconfig(game_over_text, state='normal')
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)

```



#1: CHECKERBOARD

Try changing the Game class so that the background image is drawn like a checkerboard, as in the image below.



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

```

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

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, it 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.

#2: TWO-IMAGE CHECKERBOARD

Once you've figured out how to create a checkerboard effect, try using two alternating images. Come up with another wallpaper image (using your graphics program), and then change the Game class so it displays a checkerboard with two alternating images instead of one image and the blank background.

Once you've figured out the checkerboard, you'll be able to draw two alternating images instead of an image and a blank. 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 for use later, 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
    ❶ 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 ❶, we use the `create_image` function to draw the new image on the screen.

#3: BOOKSHELF AND LAMP

You can create different wallpaper images to make the game's background more interesting. Create a copy of the background image; then draw a simple bookshelf, a table with a lamp, or a window. Dot these images around the screen by changing the `Game` class so that it displays a few different wallpaper images.

To draw different backgrounds, we'll change our alternating checkerboard code 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
draw_background = 0

```

We load the new images, saving them as variables `bg_shelf` ❶ and `bg_lamp` ❷, respectively, and then create a new variable called `count` ❸. In the previous solution, the if statement 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'll increment the value in the variable `count` by adding 1 (using `count = count + 1`). The value in `count` decides which image to draw:

```

for x in range(0, 5):
    for y in range(0, 5):
        ❶ if draw_background == 1:
            ❷ self.canvas.create_image(x * w, y * h, image=self.bg,
                                        anchor='nw')

```

```

        draw_background = 0
    else:
        ❸ count = count + 1
        ❹ if count == 5:
            ❺ self.canvas.create_image(x * w, y * h,
                image=self.bg_shelf, anchor='nw')
        ❻ elif count == 9:
            ❼ 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

```

If the `draw_background` variable is 1 ❶ we draw the first background image as normal ❷. Otherwise we increment the count variable ❸. If the value has reached 5 ❹, we draw the shelf image ❺. If the value has reached 9 ❻, we draw the lamp image ❼. Otherwise we draw the alternate background as we did previously.

#4: RANDOM BACKGROUND

As an alternative to the two-image checkerboard, try creating five different background images. You can either draw them as a repeating pattern of background images (1, 2, 3, 4, 5, 1, 2, 3, 4, 5, and so on), or you can draw them randomly.

Creating a random background is very similar code to Puzzle #2: Two-Image Checkerboard; with that solution, we created two image variables:

```

self.bg = PhotoImage(file='background.gif')
self.bg2 = PhotoImage(file='background2.gif')

```

For our random background with five different backgrounds, we can instead create a list of background images:

```

self.bg = [
    PhotoImage(file='background.gif'),
    PhotoImage(file='background2.gif'),
    PhotoImage(file='background3.gif'),
    PhotoImage(file='background4.gif'),
    PhotoImage(file='background5.gif')
]

```

Picking a random image to display is then as simple as using the choice function in the random module:

```

random.choice(self.bg)

```

The final code then looks like this:

```
❶ self.bg = [  
    PhotoImage(file='background.gif'),  
    PhotoImage(file='background2.gif'),  
    PhotoImage(file='background3.gif'),  
    PhotoImage(file='background4.gif'),  
    PhotoImage(file='background5.gif')  
]  
❷ w = self.bg[0].width()  
❸ h = self.bg[0].height()  
    for x in range(0, 5):  
        for y in range(0, 5):  
            ❹ self.canvas.create_image(x * w, y * h,  
                                       image=random.choice(self.bg),  
                                       anchor='nw')
```

We add our variable with the list of background images ❶. Then we set the `w` and `h` variables to the width and height of the first image in the list ❷ ❸ (remember that the first element in a list is always 0, so `self.bg[0]` is the first image 'background.gif'). Finally, we pick a random image to display, from the list, using the choice function ❹.



#1: "YOU WIN!"

Like the "Game Over" text in the Bounce! game, add "You Win!" text when the stick figure reaches the door.

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')

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 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)
```

We add an else clause to the if statement ❶, and Python runs this block of code if the running variable is no longer set to `True`. We sleep for a second so that the “You Win!” text doesn’t immediately appear ❷, and then change the state of the text to `'normal'` ❸ so that it appears on the canvas.

#2: ANIMATING THE DOOR

We created two images for the door: one open and one closed. When Mr. Stick Man reaches the door, the door image should change to the open door, Mr. Stick Man should vanish, and the door image should revert to the closed door. This will give the illusion that Mr. Stick Man is exiting and closing the door as he leaves. You can do this by changing the `DoorSprite` class and the `StickFigureSprite` class.

To animate the door so that it opens and closes when the stick figure reaches it, we’ll first change the `DoorSprite` class. 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
```

The two images are loaded into object variables ❶ 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 ❶. We call the `update_idletasks` function of the tk object to force the new image to be displayed 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 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` ❶, and then call the `opendoor` function of the `sprite` parameter ❷. This is actually a `DoorSprite` object, which we'll see in the next section of code. We sleep for 1 second ❸ before hiding the stick figure ❹ and then calling the `closedoor` function ❺. This makes it look as though the stick figure has gone through the door and closed it 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 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 or has collided with a sprite to the left, we check if the `endgame` variable is `True` ❶. If it is, we know that this is a `DoorSprite` object, and we call the `end` function using the `sprite` variable as the parameter ❷. 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 ❸ ❹.

#3: MOVING PLATFORMS

Try adding a new class called `MovingPlatformSprite`. This platform should move from side to side, making it more difficult for Mr. Stick Man to reach the door at the top. You can pick some platforms to be moving, and leave some platforms to be static, depending on how hard you want your game to be.

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 ❶, and then call the `__init__` function of the parent class with those same parameters ❷.

This means that any object of the `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) ❸, followed by a counter variable 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, we'll record the time in the `last_time` variable ❺; this 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 ❻ 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 stored in the `__init__` function. (You can see the difference ❶ 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

```

The `move` function checks to see if the time is greater than three-tenths of a second ❶. If it is, we set the `last_time` variable to the current time ❷. Then we move the platform image ❸ and increment the counter variable ❹. If the counter is greater than 20 (the if statement 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) ❻, and reset the counter to 0 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)
```

#4: LAMP AS A SPRITE

Instead of the bookshelf and lamp we added as background images previously, try adding a lamp that the stick man has to jump over. Rather than being a part of the game's background, it will be a sprite similar to the platforms or the door.

For this puzzle, we'll need the artwork for our lamp. The most difficult part will be figuring out where to put it on the screen.

We'll start by renaming the `PlatformSprite` class to something more sensible like `NonMovingSprite`—then we can use it for both the platforms and the lamp (as well as anything else we want to add to our game). In IDLE, navigate to the Edit menu (next to File in the menu bar), and select **Replace**. From here, enter `PlatformSprite` in the Find box, enter `NonMovingSprite` in the Replace With box, and then hit the **Replace All** button. All of the `PlatformSprite` instances have now been replaced with `NonMovingSprite`, as shown below:

```
g = Game()
platform1 = NonMovingSprite(g, PhotoImage(file='platform1.gif'),
    0, 480, 100, 10)
platform2 = NonMovingSprite(g, PhotoImage(file='platform1.gif'),
    150, 440, 100, 10)
platform3 = NonMovingSprite(g, PhotoImage(file='platform1.gif'),
    300, 400, 100, 10)
platform4 = NonMovingSprite(g, PhotoImage(file='platform1.gif'),
    300, 160, 100, 10)
platform5 = MovingPlatformSprite(g, PhotoImage(file='platform2.gif'),
    175, 350, 66, 10)
platform6 = NonMovingSprite(g, PhotoImage(file='platform2.gif'),
    50, 300, 66, 10)
platform7 = NonMovingSprite(g, PhotoImage(file='platform2.gif'),
    170, 120, 66, 10)
platform8 = NonMovingSprite(g, PhotoImage(file='platform2.gif'),
    45, 60, 66, 10)
platform9 = MovingPlatformSprite(g, PhotoImage(file='platform3.gif'),
    170, 250, 32, 10)
```

```
platform10 = NonMovingSprite(g, PhotoImage(file='platform3.gif'),
                               230, 200, 32, 10)
```

We'll add our lamp sprite to the bottom:

```
lamp = NonMovingSprite(g, PhotoImage(file='lamp.gif'), 240, 175, 21, 35)
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)
g.sprites.append(lamp)
```

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')
        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
    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 NonMovingSprite(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)

class MovingPlatformSprite(NonMovingSprite):
    def __init__(self, game, photo_image, x, y, width, height):
        NonMovingSprite.__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()

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
    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
    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()

g = Game()
platform1 = NonMovingSprite(g, PhotoImage(file='platform1.gif'),
    0, 480, 100, 10)
platform2 = NonMovingSprite(g, PhotoImage(file='platform1.gif'),
    150, 440, 100, 10)
platform3 = NonMovingSprite(g, PhotoImage(file='platform1.gif'),
    300, 400, 100, 10)
platform4 = NonMovingSprite(g, PhotoImage(file='platform1.gif'),
    300, 160, 100, 10)
platform5 = MovingPlatformSprite(g, PhotoImage(file='platform2.gif'),
    175, 350, 66, 10)
platform6 = NonMovingSprite(g, PhotoImage(file='platform2.gif'),
    50, 300, 66, 10)

```

```
platform7 = NonMovingSprite(g, PhotoImage(file='platform2.gif'),
    170, 120, 66, 10)
platform8 = NonMovingSprite(g, PhotoImage(file='platform2.gif'),
    45, 60, 66, 10)
platform9 = MovingPlatformSprite(g, PhotoImage(file='platform3.gif'),
    170, 250, 32, 10)
platform10 = NonMovingSprite(g, PhotoImage(file='platform3.gif'),
    230, 200, 32, 10)
lamp = NonMovingSprite(g, PhotoImage(file='lamp.gif'), 240, 175, 21, 35)
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)
g.sprites.append(lamp)
door = DoorSprite(g, 45, 30, 40, 35)
g.sprites.append(door)
sf = StickFigureSprite(g)
g.sprites.append(sf)
g.mainloop()
```

UPDATES

Visit <https://nostarch.com/python-kids-2nd-edition> for updates, errata, and other information.

COLOPHON

The fonts used in *Python for Kids, Second Edition* are New Baskerville, Futura, The Sans Mono Condensed and Dogma. The book was typeset with L^AT_EX 2_ε package nostarch by Boris Veytsman (2008/06/06 v1.3 *Typesetting books for No Starch Press*).

The book was produced as an example of the package nostarch.