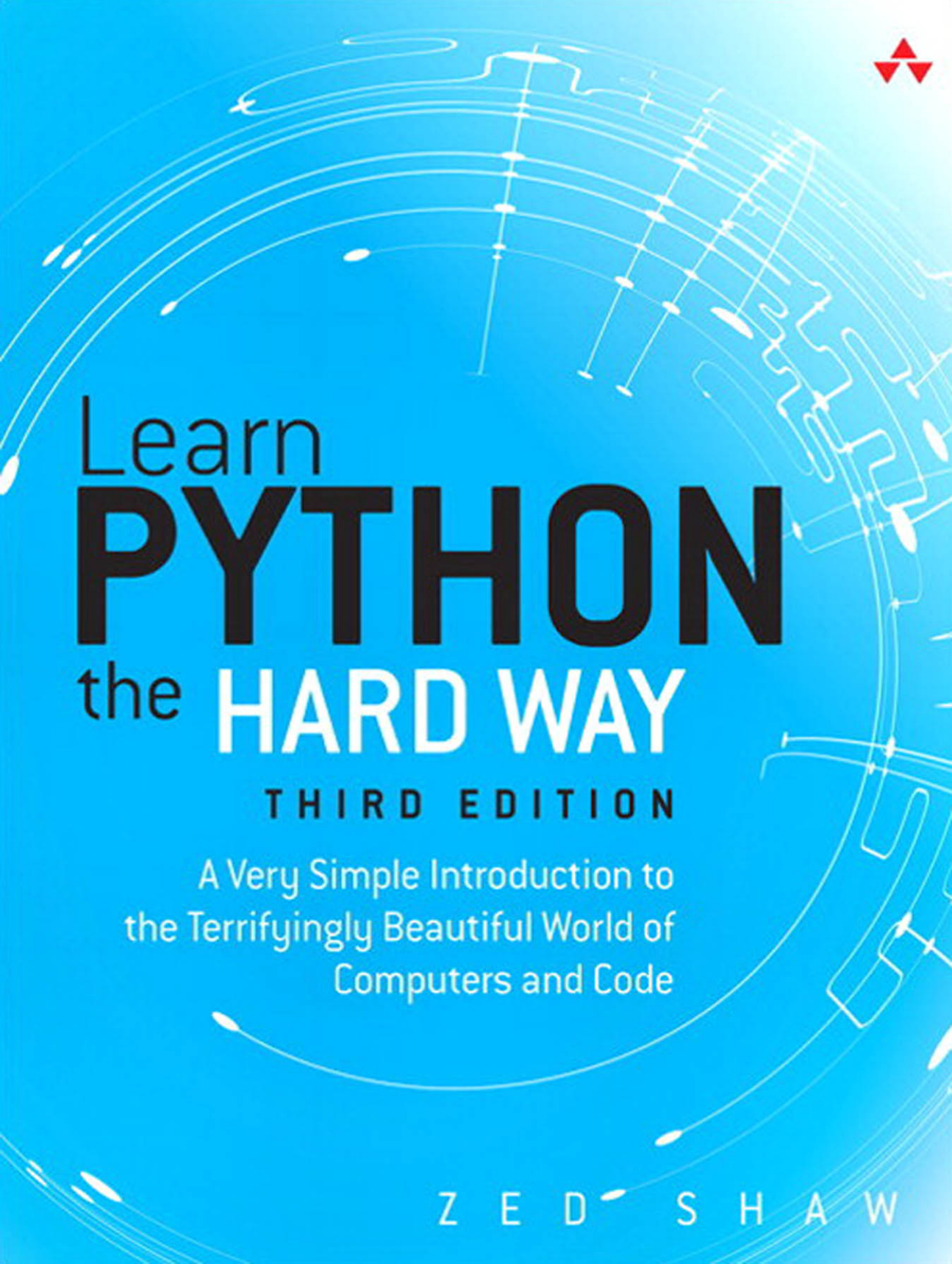
Learn
PYTHON
the HARD WAY
THIRD EDITION
A Very Simple Introduction to
the Terrifyingly Beautiful World of
Computers and Code
ZED SHAW

LEARN PYTHON THE HARD WAY

Third Edition

`f.readline()`, you're reading a line from the file and moving the read head to right after the `\n` that ends that file. This will be explained more as you go on.

Why are there empty lines between the lines in the file?
The `readline()` function returns the `\n` that's in the file at the end of that line. This means that print's `\n` is being added to the one already returned by `readline()`. To change this behavior simply add a `,` (comma) at the end of print so that it doesn't print its own `\n`.

Why does `seek(0)` not set the `current_line` to 0?
First, the `seek()` function is dealing in *bytes*, not lines. So that's going to the 0 byte (first byte) in the file. Second, `current_line` is just a variable and has no real connection to the file at all. We are manually incrementing it.

What is +=?
You know how in English I can rewrite "it is" to be "it's"? Or I can rewrite "you are" to "you're"? That's called a contraction, and this is kind of like a contraction for the two operations = and +. That means `x = x + y` is the same as `x += y`.

How does `readline()` know where each line is?
Inside `readline()` is code that scans each byte of the file until it finds a `\n` character, then stops reading the file to return what it found so far. The file `f` is responsible for maintaining the current position in the file after each `readline()` call, so that it will keep reading each line.

This page intentionally left blank

EXERCISE 21

Functions Can Return Something

You have been using the = character to name variables and set them to numbers or strings. We're now going to blow your mind again by showing you how to use = and a new Python word `return` to set variables to be a *value from a function*. There will be one thing to pay close attention to, but first type this in:

ex21.py

```
1   def add(a, b):
2       print "ADDING %d + %d" % (a, b)
3       return a + b
4
5   def subtract(a, b):
6       print "SUBTRACTING %d - %d" % (a, b)
7       return a - b
8
9   def multiply(a, b):
10      print "MULTIPLYING %d * %d" % (a, b)
11      return a * b
12
13  def divide(a, b):
14      print "DIVIDING %d / %d" % (a, b)
15      return a / b
16
17
18  print "Let's do some math with just functions!"
19
20  age = add(30, 5)
21  height = subtract(78, 4)
22  weight = multiply(90, 2)
23  iq = divide(100, 2)
24
25  print "Age: %d, Height: %d, Weight: %d, IQ: %d" % (age, height, weight, iq)
26
27
28  # A puzzle for the extra credit, type it in anyway.
29  print "Here is a puzzle."
30
31  what = add(age, subtract(height, multiply(weight, divide(iq, 2))))
32
33  print "That becomes: ", what, "Can you do it by hand?"
```

We are now doing our own math functions for `add`, `subtract`, `multiply`, and `divide`. The important thing to notice is the last line where we say `return a + b` (in add). What this does is the following:

1. Our function is called with two arguments: a and b.
2. We print out what our function is doing, in this case ADDING.
3. Then we tell Python to do something kind of backward: we return the addition of a + b. You might say this as, "I add a and b, then return them."
4. Python adds the two numbers. Then when the function ends, any line that runs it will be able to assign this a + b result to a variable.

As with many other things in this book, you should take this real slow, break it down, and try to trace what's going on. To help there's extra credit to get you to solve a puzzle and learn something cool.

What You Should See

Exercise 21 Session

```
$ python ex21.py
Let's do some math with just functions!
ADDING 30 + 5
SUBTRACTING 78 - 4
MULTIPLYING 90 * 2
DIVIDING 100 / 2
Age: 35, Height: 74, Weight: 180, IQ: 50
Here is a puzzle.
DIVIDING 50 / 2
MULTIPLYING 180 * 25
SUBTRACTING 74 - 4500
ADDING 35 + -4426
That becomes:  -4391 Can you do it by hand?
```

Study Drills

1. If you aren't really sure what `return` does, try writing a few of your own functions and have them return some values. You can return anything that you can put to the right of an =.
2. At the end of the script is a puzzle. I'm taking the return value of one function and *using* it as the argument of another function. I'm doing this in a chain so that I'm kind of creating a formula using the functions. It looks really weird, but if you run the script, you can see the results. What you should do is try to figure out the normal formula that would recreate this same set of operations.

3. Once you have the formula worked out for the puzzle, get in there and see what happens when you modify the parts of the functions. Try to change it on purpose to make another value.
4. Finally, do the inverse. Write out a simple formula and use the functions in the same way to calculate it.

This exercise might really whack your brain out, but take it slow and easy and treat it like a little game. Figuring out puzzles like this is what makes programming fun, so I'll be giving you more little problems like this as we go.

Common Student Questions

Why does Python print the formula or the functions "backward"?
It's not really backward; it's "inside out." When you start breaking down the function into separate formulas and function calls, you'll see how it works. Try to understand what I mean by "inside out" rather than "backward."

How can I use `raw_input()` to enter my own values?
Remember `int(raw_input())`? The problem with that is then you can't enter floating point, so also try using `float(raw_input())` instead.

What do you mean by "write out a formula"?
Try `24 + 34 / 100 - 1023` as a start. Convert that to use the functions. Now come up with your own similar math equation and use variables so it's more like a formula.

What Do You Know So Far?

There won't be any code in this exercise or the next one, so there's no WYSS or Study Drills either. In fact, this exercise is like one giant Study Drills section. I'm going to have you do a form of review of what you have learned so far.

First, go back through every exercise you have done so far and write down every word and symbol (another name for "character") that you have used. Make sure your list of symbols is complete.

Next to each word or symbol, write its name and what it does. If you can't find a name for a symbol in this book, then look for it online. If you do not know what a word or symbol does, then go read about it again and try using it in some code.

You may run into a few things you just can't find out or know, so just keep those on the list and be ready to look them up when you find them.

Once you have your list, spend a few days rewriting the list and double-checking that it's correct. This may get boring, but push through and really nail it down.

Once you have memorized the list and what they do, then you should step it up by writing out tables of symbols, their names, and what they do *from memory*. When you hit some you can't recall from memory, go back and memorize them again.

WARNING! The most important thing when doing this exercise is: "There is no failure, only trying."

What You Are Learning

It's important when you are doing a boring, mindless memorization exercise like this to know why. It helps you focus on a goal and know the purpose of all your efforts.

In this exercise, you are learning the names of symbols so that you can read source code more easily. It's similar to learning the alphabet and basic words of English, except this Python alphabet has extra symbols you might not know.

Just take it slow and do not hurt your brain. Hopefully by now these symbols are natural for you, so this isn't a big effort. It's best to take 15 minutes at a time with your list and then take a break. Giving your brain a rest will help you learn faster with less frustration.