
Learn Python The Hard Way

Release 0.1

Zed A. Shaw

April 26, 2010

CONTENTS

1	The Hard Way Is Easier	3
1.1	Reading And Writing	3
1.2	Attention To Detail	3
1.3	Spotting Differences	3
1.4	Do Not Copy-Paste	4
2	Exercise 0: The Setup	5
2.1	What You Should See	5
3	Exercise 1: A Good First Program	7
3.1	What You Should See	7
3.2	Extra Credit	8
4	Exercise 2: Comments And Pound Characters	9
4.1	What You Should See	9
4.2	Extra Credit	9
5	Exercise 3: Numbers And Math	11
5.1	What You Should See	12
5.2	Extra Credit	12
6	Exercise 4: Variables And Names	13
6.1	What You Should See	13
6.2	Extra Credit	14
7	Exercise 5: More Variables And Printing	15
7.1	What You Should See	15
7.2	Extra Credit	16
8	Exercise 6: Strings And Text	17
8.1	What You Should See	18
8.2	Extra Credit	18
9	Exercise 7: More Printing	19
9.1	What You Should See	19
9.2	Extra Credit	20
10	Exercise 8: Printing, Printing	21
10.1	What You Should See	21
10.2	Extra Credit	21

11 Exercise 9: Printing, Printing	23
11.1 What You Should See	23
11.2 Extra Credit	23
12 Exercise 10: What Was That?	25
12.1 What You Should See	25
12.2 Extra Credit	26
13 Exercise 27: Memorizing Logic	27
13.1 The Truth Terms	27
13.2 The Truth Tables	28
14 Advice From An Old Programmer	31
15 Indices and tables	33

Contents:

THE HARD WAY IS EASIER

This simple book is meant to give you a first start in programming. The title says it is the hard way to learn to write code but it's actually not. It's the "hard" way only in that it's the way people used to teach things. In this book you will do something incredibly simple that all programmers actually do to learn a language:

1. Go through each exercise.
2. Type in each sample *exactly*.
3. Make it run.

That's it. This will be *very* difficult at first, but stick with it. If you go through this book, and do each exercise for 1-2 hours a night, then you'll have a good foundation for moving on to another book. You might not really learn "programming" from this book, but you will learn the foundation skills you need to start learning the language.

This book's job is to teach you the three most basic essential skills that a beginning programmer needs to know: Reading And Writing, Attention To Detail, Spotting Differences.

1.1 Reading And Writing

It seems stupidly obvious, but if you have a problem typing you'll have a problem learning to code. Not only that, but if you have a problem typing the fairly odd characters in source code then you'll be unable to learn even the most basic things about how software works.

Typing the code samples and getting them to run will help you learn the names of the symbols, get familiar with typing them, and get you reading the language.

1.2 Attention To Detail

The one skill that separates bad programmers from good programmers is attention to detail. In fact, it's what separate the good from the bad in any profession. Without an attention to the tiniest details of your work you'll miss key important elements of what you create. In programming this is how you end up with bugs and difficult to use systems.

By going through this book and copying each example *exactly* you will be training your brain to focus on exacting details of what you are doing.

1.3 Spotting Differences

A very important skill that most programmers develop over time is the ability to visually notice differences between things. An experienced programmer can take two pieces of code that are slightly different and immediately start

pointing out the differences. In fact, programmers have invented tools to make this even easier. Of course, the invention of these tools also means that even with years of training finding little differences in programming is difficult.

While you do these exercises and type each one in, you'll be making mistakes. It's inevitable, and even seasoned programmers would make a few mistakes. Your job is to compare what you've written to what's required and fix all the differences, and by doing this you'll train yourself to notice your mistakes, bugs, and other problems caused by them.

1.4 Do Not Copy-Paste

The last thing I'll say before we begin is that you *must* type each of these exercises in manually. If you copy and paste you might as well just not even do them. The point of these exercises is to train your hands, your brain, and your mind in how to read, write, and see code. If you copy-paste you are cheating yourself out of the effectiveness of the lessons.

EXERCISE 0: THE SETUP

This exercise has no code. It is simply the exercise you complete in order to get your computer setup to run Python. Because this is a simple book we'll assume you're using a Mac OSX computer. Later versions of this book will have instructions for Windows and Linux computers.

To complete this exercise you have to finish the following tasks:

1. Find a text editor for programming. This text editor should be easy to use, not have too many features, and should be free. Do a search in your favorite search engine for one.
2. Put your editor in your Dock so you can reach it easily.
3. Find your "Terminal" program. Search for it. You'll find it.
4. Put your Terminal in your Dock as well.
5. Run your Terminal program. It won't look like much.
6. In your Terminal program, run `python`. You run things in Terminal by just typing their name and hitting RETURN.
7. Hit CTRL-D (^D) and get out of python.
8. You should be back at a prompt similar to what you had before you typed `python`. If not find out why.
9. Learn how to make a directory in the Terminal. Search online for help.
10. Learn how to change into a directory in the Terminal. Again search online.
11. Use your editor to create a file in this directory. Typically you will make the file and then "Save" or "Save As.." and pick this directory.
12. Go back to Terminal using just the keyboard to switch windows. Look it up if you can't figure it out.
13. Back in Terminal see if you can list the directory to see your newly created file. Search online for how to list a directory.

You're done with this exercise. This exercise could actually be hard for you depending on your familiarity with your computer. If it is difficult, then take the time to read and study and get through it, because until you can do these very basic things you'll find it difficult to get much programming done.

2.1 What You Should See

Here's me doing the above on my computer in Terminal. Your computer would be different, so see if you can figure out all the differences between what I did and what you should do. Notice I use a text editor called `vim`. You shouldn't use it, it's too hard to use for you right now.

```
Last login: Sat Apr 24 00:56:54 on ttys001
~ $ python
Python 2.5.1 (r251:54863, Feb  6 2009, 19:02:12)
[GCC 4.0.1 (Apple Inc. build 5465)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> ^D
~ $ mkdir mystuff
~ $ cd mystuff
mystuff $ ls
mystuff $ vim test.txt
mystuff $ ls
test.txt
mystuff $
```

If a programmer tells you to use `vim` or `emacs` tell them no. These editors are for when you are a better programmer. All you need right now is an editor that lets you put text into a file.

EXERCISE 1: A GOOD FIRST PROGRAM

Remember, you should have spent a good amount of time in Exercise 0 learning how to install a text editor, run the text editor, run the Terminal, and work with both of them. If you haven't done that then don't go on, you'll not have a good time. This is the only time I'll start an exercise with a warning that you should not skip or get ahead of yourself.

```
1 print "Hello World!"
2 print "Hello Again"
3 print "I like typing this."
4 print "This is fun."
5 print 'Yay! Printing.'
6 print "I'd much rather you 'not'."
7 print 'I "said" do not touch this.'
```

Take the above and type it into a single file named `ex1.py`. This is important as python works best with files ending in `.py`.

Warning: Do not type the numbers on the far left of these lines. Those are called “line numbers” and they’re used by programmers to talk about what part of a program is wrong. Python will tell you errors related to these line numbers, but *you* do not type them in.

Then in Terminal you *run* the file by typing:

```
python ex1.py
```

If you did it right then you should see the same output I have below. If not, then you’ve done something wrong. No, the computer is not wrong.

3.1 What You Should See

```
$ python ex1.py
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd much rather you 'not'.
```

```
I "said" do not touch this.  
$
```

You may see the name of your directory before the \$ which is fine, but if your output is not exactly the same, then find out why and fix it.

If you have an error it will look like this:

```
$ python ex1.py  
File "ex1.py", line 3  
    print "I like typing this."  
        ^  
SyntaxError: EOL while scanning single-quoted string  
$
```

It's important you be able to read these since you'll be making many of these mistakes. Even I make many of these mistakes. Let's look at this line-by-line.

1. Here we ran our command in the shell to run the `ex1.py` script.
2. Python then tells us that the file `ex1.py` has an error on line 3.
3. It then prints this line for us.
4. Then it puts a ^ (caret) character to point at where the problem is. Notice the missing " (double-quote) character?
5. Finally, it prints out a "SyntaxError" and tells us something about what might be the error. Usually these are very cryptic, but if you copy that text into a search engine you'll find someone else who's had that error and you can probably figure out how to fix it.

3.2 Extra Credit

You will also have extra credit you should do to make sure you understand each exercise. For this exercise, try these things:

1. Make your script print another line.
2. Make your script print only one of the lines.
3. Put a '#' (pound) character at the beginning of a line. What did it do? Try to find out what this character does.

From now on, I won't explain how each exercise works unless an exercise is different for some reason. Each time there is code you should put in a new file, the output you should see when you run the file in terminal, and extra credit you should do.

EXERCISE 2: COMMENTS AND POUND CHARACTERS

Comments are very important in your programs. They are used to tell you what something does in English, and they also are used to disable parts of your program if you need to remove them temporarily. Here's how you do comments.

```
1 # A comment, this is so you can read your program later.
2 # Anything after the # is ignored by python.
3
4 print "I could have code like this." # and the comment after is ignored
5
6 # You can also use a comment to "disable" or comment out a piece of code:
7 # print "This won't run."
8
9 print "This will run."
```

4.1 What You Should See

```
$ python ex2.py
I could have code like this.
This will run.
$
```

4.2 Extra Credit

1. Find out if you were right about what the # character does and make sure you know what it's called (pound character).
2. Take your `ex2.py` file and review each line going backwards. Start at the last line, and check each word in reverse against what you should have typed.
3. Did you find more mistakes? Fix them.
4. Read what you typed above out loud, including saying each character by its name. Did you find more mistakes? Fix them.

EXERCISE 3: NUMBERS AND MATH

Every programming language has some kind of way of doing numbers and math. Don't worry, programmers lie frequently about being math geniuses when they really aren't. If they were math geniuses, they would be doing math not writing ads and social network games to steal people's money.

This exercise has lots of math symbols so let's name them right away so you know what they're called. As you type this one in, say the names. When saying them feels boring you can stop saying them. Here's the names:

- + plus
- - minus
- / slash
- * asterisk
- ^ caret
- % percent
- < less-than
- > greater-than
- <= less-than-equal
- >= greater-than-equal

Notice how the operations are missing? After you type in the code for this exercise you are to go back and figure out what each of these does and complete the table. For example, + does addition.

```
1 print "I will now count my chickens:"
2
3 print "Hens", 25 + 30 / 6
4 print "Roosters", 100 - 25 * 3 % 4
5
6 print "Now I will count the eggs:"
7
8 print 3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6
9
10 print "Is it true that 3 + 2 < 5 - 7?"
11
12 print 3 + 2 < 5 - 7
13
14 print "What is 3 + 2?", 3 + 2
15 print "What is 5 - 7?", 5 - 7
16
17 print "Oh, that's why it's True."
```

```
18
19 print "How about some more."
20
21 print "Is it greater?", 5 > -2
22 print "Is it greater or equal?", 5 >= -2
23 print "Is it less or equal?", 5 <= -2
```

5.1 What You Should See

```
$ python ex3.py
I will now count my chickens:
Hens 30
Roosters 97
Now I will count the eggs:
7
Is it true that 3 + 2 < 5 - 7?
False
What is 3 + 2? 5
What is 5 - 7 -2
Oh, that's why it's True.
How about some more.
Is it greater? True
Is it greater or equal? True
Is it less or equal? False
$
```

5.2 Extra Credit

1. Above each line, use the # to write a comment to yourself explaining what the line does.
2. Remember in Exercise 0 when you started python? Start python this way again and using the above characters and what you know, use python as a calculator.
3. Find something you need to calculate and write a new .py file that does it.
4. Notice the math seems “wrong”? There are no fractions, only whole numbers. Find out why by researching what a “floating point” number means.
5. Rewrite ex3.py to use floating point numbers so it’s more accurate.

EXERCISE 4: VARIABLES AND NAMES

You can print things out with `print` and you can do math. The next step is to learn about variables. In programming a variable is nothing more than a name for something so you can use the name rather than the something as you code. Programmers use these variable names to make their code read more like English, and because programmers have a lousy ability to remember things. If they didn't use good names for things in their software they'd get lost when they came back and tried to read their code again.

If you get stuck with this exercise, remember the tricks you've been taught so far of finding differences and focusing on details:

1. Write a comment above each line explaining to yourself what it does in English.
2. Read your `.py` file backwards.
3. Read your `.py` file out loud saying even the characters.

```
1 cars = 100
2 space_in_a_car = 4.0
3 drivers = 30
4 passengers = 90
5 cars_not_driven = cars - drivers
6 cars_driven = drivers
7 carpool_capacity = cars_driven * 4.0
8 average_passengers_per_car = carpool_capacity / passengers
9
10
11 print "There are", cars, "cars available."
12 print "There are only", drivers, "drivers available."
13 print "There will be", cars_not_driven, "empty cars today."
14 print "We can transport", carpool_capacity, "people today."
15 print "We have", passengers, "to carpool today."
16 print "We need to put about", average_passengers_per_car, "in each car."
```

Note: The `_` in `space_in_a_car` is called an underscore character. Find out how to type it if you don't already know. We use this character a lot to put an imaginary space between words in variable names.

6.1 What You Should See

```
$ python ex4.py
There are 100 cars available.
There are only 30 drivers available.
There will be 70 empty cars today.
```

```
We can transport 120.0 people today.  
We have 90 to carpool today.  
We need to put about 1.33333333333 in each car.  
$
```

6.2 Extra Credit

When I wrote this program the first time I had a mistake, and python told me about it like this:

```
Traceback (most recent call last):  
  File "ex4.py", line 8, in <module>  
    average_passengers_per_car = car_pool_capacity / passenger  
NameError: name 'car_pool_capacity' is not defined
```

Explain this error in your own words, make sure you use line numbers and explain why.

Here's more extra credit:

1. Explain why the 4.0 is used instead of just 4.
2. Remember that 4.0 is a “floating point” number, make sure you find out what that means.
3. Write comments above each of the variable assignments.
4. Make sure you know what = is called (equals) and that it's making names for things.
5. You should also know that _ is an underscore character.
6. Try running `python` as a calculator like you did before and use variable names to do your calculations. Popular variable names are also `i`, `x`, and `j`.

EXERCISE 5: MORE VARIABLES AND PRINTING

We'll now do even more typing of variables and printing them out. This time though we'll use something called a "format string". You might not know it, but every time you put " (double-quotes) around a piece of text you've been making a string. A string is how you make something that your program might give to a human. You print them, save them to files, send them to web servers, all sorts of things.

Strings are really handy, so in this exercise you'll learn how to make strings that have variables embedded in them. You embed variables inside a string by using specialized format sequences and then putting the variables at the end with a special syntax that tells Python, "Hey, this is a format string, put these variables in there."

As usual, just type this in even if you don't understand it and make it exactly the same.

```
1 my_name = 'Zed A. Shaw'
2 my_age = 35 # not a lie
3 my_height = 74 # inches
4 my_weight = 180 # lbs
5 my_eyes = 'Blue'
6 my_teeth = 'White'
7 my_hair = 'Brown'
8
9 print "Let's talk about %s." % my_name
10 print "He's %d inches tall." % my_height
11 print "He's %d pounds heavy." % my_weight
12 print "Actually that's not too heavy."
13 print "He's got %s eyes and %s hair." % (my_eyes, my_hair)
14 print "His teeth are usually %s depending on the coffee." % (my_hair)
15
16 # this line is tricky, try to get it exactly right
17 print "If I add %d, %d, and %d I get %d." % (
18     my_age, my_height, my_weight, my_age + my_height + my_weight)
```

7.1 What You Should See

```
$ python ex5.py
Let's talk about Zed A. Shaw.
He's 74 inches tall.
He's 180 pounds heavy.
Actually that's not too heavy.
```

```
He's got Blue eyes and Brown hair.  
His teeth are usually Brown depending on the coffee.  
If I add 35, 74, and 180 I get 289.  
$
```

7.2 Extra Credit

1. Change all the variables so there isn't the `my_` in front. Make sure you change the name everywhere, not just where you used `=` to set them.
2. Try more format characters. `%r` is a very useful one, it's like saying "print this no matter what".
3. Search online for all of the Python format characters.
4. Try to write some variables that convert the inches and pounds to centimeters and kilos. Don't just type in the measurements, but work out the math in Python.

EXERCISE 6: STRINGS AND TEXT

You have already been writing strings but haven't really known what they do. In this exercise we create a bunch of variables with complex strings so you can see what they're for. First an explanation of strings.

A string is usually a bit of text you want to display to someone, or “export” out of the program you are writing. Python knows you want something to be a string when you put either " (double-quotes) or ' (single-quotes) around the text. You saw this many times with your use of `print` where you would put text you want to go to the string inside " or ' after the `print`. Then Python prints it.

Strings also can contain the format characters you've discovered so far. You simply put the formatted variables in the string, and then a % (percent) character, followed by the variable. The *only* catch is that if you want multiple formats in your string to print multiple variables, then you need to put them inside () (parenthesis) separated by , (commas). It's as if you were telling me to buy you a list of items from the store and you said, “I want milk, eggs, bread, and soup.” Only as a programmer we can say, “(milk, eggs, bread, soup)” and be done with it.

We will now type in a whole bunch of strings, variables, formats, and print them. You will also practice using short abbreviated variable names. Programmers love saving themselves time at your expense by using annoying cryptic variable names, so let's get you started being able to read and write them early on.

```
1 x = "There are %d types of people." % 10
2 binary = "binary"
3 do_not = "don't"
4 y = "Those who know %s and those who %s." % (binary, do_not)
5
6 print x
7 print y
8
9 print "I said: %r." % x
10 print "I also said: '%s'." % y
11
12 hilarious = False
13 joke_evaluation = "Isn't that joke so funny?! %r"
14
15 print joke_evaluation % hilarious
16
17 w = "This is the left side of..."
18 e = "a string with a right side."
19
20 print w + e
```

8.1 What You Should See

```
$ python ex.py
There are 10 types of people.
Those who know binary and those who don't.
I said: 'There are 10 types of people.'.
I also said: 'Those who know binary and those who don't.'.
Isn't that joke so funny?! False
This is the left side of...a string with a right side.
$
```

8.2 Extra Credit

1. Go through this program and write a comment above each line explaining it.
2. Find all the places where a string is put inside a string. There are 4 places.
3. Are you sure there's only 4 places? How do you know? Maybe I like lying.
4. Try to explain why adding the two string `w` and `e` with `+` makes a longer string.

EXERCISE 7: MORE PRINTING

We are now going to do a bunch of exercises where you just type code in and make them run. There won't be much talking since it's just more of the same. The purpose is to build up your chops. See you in a few exercises, and *don't skip!*

```
1 print "Mary had a little lamb."
2 print "It's fleece was white as %s." % 'snow'
3 print "And everywhere that Mary went."
4 print "." * 10 # what'd that do?
5
6 end1 = "C"
7 end2 = "h"
8 end3 = "e"
9 end4 = "e"
10 end5 = "s"
11 end6 = "e"
12 end7 = "B"
13 end8 = "u"
14 end9 = "r"
15 end10 = "g"
16 end11 = "e"
17 end12 = "r"
18
19 # watch that comma at the end. try removing it to see what happens
20 print end1 + end2 + end3 + end4 + end5 + end6,
21 print end7 + end8 + end9 + end10 + end11 + end12
```

9.1 What You Should See

```
$ python ex.py
Mary had a little lamb.
It's fleece was white as snow.
And everywhere that Mary went.
.....
Cheese Burger
$
```

9.2 Extra Credit

For these next few exercises you will have the exact same extra credit.

1. Go back through and write a comment on what each line does.
2. Read each one backwards or out loud to find your errors.
3. From now on, when you make mistakes write down on a piece of paper what kind of mistake you made.
4. When you go to the next exercise, look at the last mistakes you made and try not to make them in this new one.
5. Remember that everyone makes mistakes. Programmers are like magicians who like everyone to think they're perfect and never wrong, but it's all an act. They make mistakes all the time.

EXERCISE 8: PRINTING, PRINTING

```
1 formatter = "%r %r %r %r"
2
3 print formatter % (1, 2, 3, 4)
4 print formatter % ("one", "two", "three", "four")
5 print formatter % (True, False, False, True)
6 print formatter % (formatter, formatter, formatter, formatter)
7 print formatter % (
8     "I had this thing.",
9     "That you could type up right.",
10    "But it didn't sing.",
11    "So I said goodnight."
12 )
```

10.1 What You Should See

```
$ python ex8.py
1 2 3 4
'one' 'two' 'three' 'four'
True False False True
'%r %r %r %r' '%r %r %r %r' '%r %r %r %r' '%r %r %r %r'
'I had this thing.' 'That you could type up right.' "But it didn't sing." 'So I said goodnight.'
```

10.2 Extra Credit

1. Do your checks of your work, write down your mistakes, try not to make them on the next exercise.

EXERCISE 9: PRINTING, PRINTING

```
1  # Here's some new strange stuff, remember type it exactly.
2
3  days = "Mon Tue Wed Thu Fri Sat Sun"
4  months = "Jan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug"
5
6  print "Here's the days: ", days
7  print "Here's the months: ", months
8
9  print """
10 There's something going on here.
11 With the three double-quotes.
12 We'll be able to type as much as we like.
13 """
```

11.1 What You Should See

```
$ python ex9.py
Here's the days:  Mon Tue Wed Thu Fri Sat Sun
Here's the months:  Jan
Feb
Mar
Apr
May
Jun
Jul
Aug

There's something going on here.
With the three double-quotes.
We'll be able to type as much as we like.

$
```

11.2 Extra Credit

1. Do your checks of your work, write down your mistakes, try not to make them on the next exercise.

EXERCISE 10: WHAT WAS THAT?

In Exercise 9 I threw you some new stuff you haven't seen yet, just to keep you on your toes. In that exercise I showed you two ways to make a string that goes across multiple lines. In the first way, I put the characters `n` (back-slash `n`) between the names of the months. What these two characters do is put a new line character into the string at that point.

This use of the (back-slash) character is a way we can put difficult to type characters into a string. There's plenty of these "escape sequences" available for different characters you might want to put in, but there's a special one, the double back-slash which is just two of them `\\`. These two characters will print just one back-slash. We'll try a few of these sequences so you can see what I mean.

The second way is by doing the triple-quotes, which is just `"""` and works like a string, but what it does you can put as many lines of text you want until you type `"""` again. We'll also play with these some too.

```
1 tabby_cat = "\tI'm tabbed in."
2 persian_cat = "I'm split\non a line."
3 backslash_cat = "I'm \\ an \\ cat."
4
5 fat_cat = """
6 I'll do a list:
7 \t* Cat food
8 \t* Fishies
9 \t* Catnip\n\t*Grass
10 """
11
12 print tabby_cat
13 print persian_cat
14 print backslash_cat
15 print fat_cat
```

12.1 What You Should See

Look for the tab characters that you made. In this exercise the spacing is important to get right.

```
$ python ex.py
    I'm tabbed in.
I'm split
on a line.
I'm \ an \ cat.

I'll do a list:
```

```
* Cat food  
* Fishies  
* Catnip  
*Grass
```

\$

12.2 Extra Credit

1. Go search online to see what other escape sequences are available.
2. Try using `'''` instead. Can you see why you might use that instead of `"""`?
3. Try to combine escape sequences and format strings to create a more complex format.

EXERCISE 27: MEMORIZING LOGIC

Today is the day you start learning about logic. Up to this point you have done everything you possibly can reading and writing files, to the terminal, and have learned quite a lot of the math capabilities of Python.

From now on though, you will be learning about logic, but just enough logic to be dangerous. You won't learn complex theories that academics love to study, but instead just the simple basic logic that makes real programs work and that real programmers need every day.

However, learning logic has to come after you do some memorization. I want you to do this exercise for an entire week. Do not falter. Even if you are bored out of your mind, keep doing it. This exercise has a set of logic tables you must memorize to make it easier for you to do the later exercises.

I'm warning you this won't be fun at first. It will be downright boring and tedious but this is to teach you a very important skill you'll need as a programmer. You *will* need to be able to memorize important concepts as you go in your life. Most of these concepts will be exciting once you get them. You'll struggle with them, like wrestling a squid, then one day *snap* you'll understand it. All that work memorizing the basics pays off big later.

Here's a tip on how to memorize something without going insane: Do it a little tiny bit at a time throughout the day and mark down what you need to work on most. Don't try to sit down for 2 hours straight and memorize these tables as that won't work. Your brain will really only retain whatever you studied in the first 15 or 30 minutes anyway.

Instead, what you should do is create a bunch of index cards with each column on the left on one side (True or False) and the column on the right on the back. You should then pull them out, see the "True or False" and be able to immediately say "True!" Keep practicing until you can do this.

Once you can do that, start writing out your own truth tables each night into a notebook. Don't just copy them, but instead try to do them from memory, and when you get stuck glance quickly at the ones I have here to refresh your memory. Doing this will train your brain to remember the whole table.

Don't spend more than one week on this, because you'll be applying it as you go. You might want to keep your index cards as a "warm up" before you sit down to do further exercises though.

13.1 The Truth Terms

In python we have the following terms (characters and phrases) for determining if something is "True" or "False". Logic on a computer is all about determining if some combination of these characters and some variables is True at that point in the program.

- `and`
- `or`
- `not`
- `!=` (not equal)

- == (equal)
- >= (greater-than-equal)
- <= (less-than-equal)
- True
- False

You actually have ran into these characters before, but maybe not the phrases. The phrases (and, or, not) actually work the way you expect them to, just like in English.

13.2 The Truth Tables

We will now use these characters to make the truth tables you need to memorize.

NOT	True?
not False	True
not True	False

OR	True?
True or False	True
True or True	True
False or True	True
False or False	False

AND	True?
True and False	False
True and True	True
False and True	False
False and False	False

NOT OR	True?
not True or False	False
not True or True	True
not False or True	True
not False or False	True

NOT AND	True?
not True and False	True
not True and True	False
not False and True	True
not False and False	True

!=	True?
1 != 0	True
1 != 1	False
0 != 1	True
0 != 0	False

==	True?
1 == 0	False
1 == 1	True
0 == 1	False
0 == 0	True

Now use these tables to write up your own cards and spend the week memorizing them.

ADVICE FROM AN OLD PROGRAMMER

You've finished this book and now you have decided to continue on with programming. Maybe it will be a career for you, or maybe you'll just do it as a hobby. For whatever reason you'll need some advice to make sure you continue on the right path and get the most enjoyment out of your newly chosen hobby.

I have been programming for a very long time. So long that it is incredibly boring to me. At the time that I wrote this book I knew about 20 programming languages and could learn new ones in about a day to a week depending on how weird they were. Eventually though this just became boring and couldn't hold my interest.

What I discovered after this journey of learning was that the languages didn't matter, it was what you did with them. Actually, I always knew that, but I'd get distracted by the languages and forget it periodically. Now I never forget it, and neither should you.

The programming language you learn and use does not matter. Do *not* get sucked into the religion surrounding programming languages as that will only blind you to their true purpose of being your tool for doing interesting things.

Programming as an intellectual activity is the *only* art form that allows you to create interactive art. You can create projects that other people can play with and you can talk to them indirectly. No other art form is quite this interactive. Movies go out to the audience. Paintings don't move. Code goes both ways.

Programming as a profession is only moderately interesting. It can be a good job, but if you want to make about the same money and be happier you could actually just go run a fast food joint. You are much better off using code as your secret weapon in another profession.

People who can code in the world of technology companies are a dime a dozen and get no respect. People who can code in biology, medicine, government, sociology, physics, history, and mathematics are respected and can do amazing things to advance those disciplines.

Of course, all of this advice is pointless. If you liked learning to write software with this book then you should try to use it to improve your life anyway you can. You should go out and explore this weird wonderful new intellectual pursuit that barely anyone in the last 50 years has been able to explore. Might as well enjoy it while you can.

Finally, I will say that learning to create software changes you and makes you different. Not better or worse, just different. You may find that people treat you harshly because you can create software, maybe using words like "nerd". Maybe you'll find that because you can dissect their logic that they hate arguing with you. You may even find that simply knowing how a computer works makes you annoying and weird to them.

To this I only have one piece of advice: they can go to hell. The world needs more weird people who know how things work and who love to figure it all out. When they treat you like this, just remember that this is *your* journey, not theirs. Being different is not a crime, and people who tell you it is are just jealous that you've picked up a skill they never in their wildest dreams could acquire.

You can code. They cannot. That is pretty damn cool.

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*