# Learn to Code by Solving Problems

A Python Programming Primer

by Daniel Zingaro

errata updated to print 3

<table>
<thead>
<tr>
<th>Page</th>
<th>Error</th>
<th>Correction</th>
<th>Print corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxiii</td>
<td>The latest version of Python is Python 3.9.</td>
<td>The latest version of Python is Python 3.11.</td>
<td>Print 3</td>
</tr>
<tr>
<td>xxiii</td>
<td>... click either Add Python 3.9 to PATH or Add Python to environment variables ...</td>
<td>... click either Add Python 3.11 to PATH or Add Python to environment variables ...</td>
<td>Print 3</td>
</tr>
<tr>
<td>3</td>
<td>Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:30:23) [MSC v.1928 32 bit (Intel)] on win32</td>
<td>Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32</td>
<td>Print 3</td>
</tr>
<tr>
<td>4</td>
<td>Python 3.9.2 (default, Mar 15 2021, 17:23:44) [Clang 11.0.0 (clang-1100.0.33.17)] on darwin</td>
<td>Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [Clang 13.0.0 (clang-1300.0.29.39)] on darwin</td>
<td>Print 3</td>
</tr>
<tr>
<td>5</td>
<td>Python 3.9.2 (default, Feb 20 2021, 20:57:50) [GCC 7.5.0] on linux</td>
<td>Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [GCC 7.5.0] on linux</td>
<td>Print 3</td>
</tr>
</tbody>
</table>
| 32 | ```python
>>> if apple_total > banana_total:
    ...   print('A')
>>> elif banana_total > apple_total:
    ...   print('B')
>>> elif apple_total == banana_total:
    ...   print('T')
``` | ```python
>>> if apple_total > banana_total:
    ...   print('A')
>>> elif banana_total > apple_total:
    ...   print('B')
>>> elif apple_total == banana_total:
    ...   print('T')
``` | Print 3 |
<p>| 41 | Our solution is in Listing 2.2. | Create a text file called telemarketers.py and type the code in Listing 2-2. | Print 3 |</p>
<table>
<thead>
<tr>
<th>Page</th>
<th>Error</th>
<th>Correction</th>
<th>Print corrected</th>
</tr>
</thead>
</table>
| 50–51 | ```
>>> for char in secret_word:
...    print('Letter: ' + char)
...
5 iterations, coming right up!
``` | ```
>>> for char in secret_word:
...    print('Letter: ' + char)
...
``` | Print 3 |
| 132 | Our code to solve this problem is in Listing 5–6. | Our code to solve Baker Bonus is in Listing 5–6. | Print 3 |
| 148 | Is the following version of no_high correct? That is, does it return True if there is at least one high card in the list, and False otherwise? | Is the following version of no_high correct? That is, does it return True if there are no high cards in the list, and False otherwise? | Print 3 |
| 158, 165 | ```
for i in range(len(box)):
    box[i] = int(box[i])
``` | ```
for j in range(len(box)):
    box[j] = int(box[j])
``` | Print 3 |
| 178 | To write a number to a file, convert it to a string first: | To write a number to a file, convert it to a string first. You can do that using an f-string: | Print 3 |
| 180 | ```
for word in words:
    if chars_on_line + len(word) <= k:
        line = line + word + ' '
        chars_on_line = chars_on_line + len(word)
    else:
        output_file.write(line[:-1] + '\n')
        line = word + ' '
        chars_on_line = len(word)
output_file.write(line[:-1] + '\n')
``` | ```
for word in words:
    if chars_on_line + len(word) <= k:
        line = line + word + ' '
        chars_on_line = chars_on_line + len(word)
    else:
        output_file.write(f'{line[:-1]}\n')
        line = word + ' '
        chars_on_line = len(word)
output_file.write(f'{line[:-1]}\n')
``` | Print 3 |
| 181 | Second, you may have expected me to use an f-string here, like this: | Second, I used an f-string to simplify adding the newline character at the end of the line; equivalent code that doesn’t use an f-string looks like this: | Print 3 |
| | ```
output_file.write(f'{line[:-1]}\n')
``` | ```
output_file.write(line[:-1] + '\n')
``` | |
<table>
<thead>
<tr>
<th>Page</th>
<th>Error</th>
<th>Correction</th>
<th>Print corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td>196, 198</td>
<td><code>output_file.write(output + '\n')</code></td>
<td><code>output_file.write(f'(output)\n')</code></td>
<td>Print 3</td>
</tr>
<tr>
<td>234</td>
<td><code>output_file.write(str(total // 2) + '\n')</code></td>
<td><code>output_file.write(f'(total // 2)\n')</code></td>
<td>Print 3</td>
</tr>
<tr>
<td>241</td>
<td><code>output_file.write(str(max_covered) + '\n')</code></td>
<td><code>output_file.write(f'(max_covered)\n')</code></td>
<td>Print 3</td>
</tr>
<tr>
<td>248</td>
<td><code>output_file.write(str(min_cost) + '\n')</code></td>
<td><code>output_file.write(f'(min_cost)\n')</code></td>
<td>Print 3</td>
</tr>
<tr>
<td>251</td>
<td><code>output_file.write(str(total) + '\n')</code></td>
<td><code>output_file.write(f'(total)\n')</code></td>
<td>Print 3</td>
</tr>
<tr>
<td>254</td>
<td><code>output_file.write(str(total) + '\n')</code></td>
<td><code>output_file.write(f'(total)\n')</code></td>
<td>Print 3</td>
</tr>
<tr>
<td>256</td>
<td>Python has a binary search function that will put the finishing touches on Cow Baseball. That function, though, is inside of something called a module; we'll need to discuss them first.</td>
<td>Python has binary search functions that will put the finishing touches on Cow Baseball. Those functions, though, are inside of something called a module; we'll need to discuss them first.</td>
<td>Print 3</td>
</tr>
<tr>
<td>261</td>
<td><code>output_file.write(str(total) + '\n')</code></td>
<td><code>output_file.write(f'(total)\n')</code></td>
<td>Print 3</td>
</tr>
<tr>
<td>271</td>
<td>then 8n is 40,000. The number 8 is so small compared to 40,000</td>
<td>then 2n is 10,000. The number 8 is so small compared to 10,000</td>
<td>Print 3</td>
</tr>
<tr>
<td>277</td>
<td><code>output_file.write(str(total) + '\n')</code></td>
<td><code>output_file.write(f'(total)\n')</code></td>
<td>Print 3</td>
</tr>
</tbody>
</table>