Learn to Code by Solving Problems

A Python Programming Primer

by Daniel Zingaro

errata updated to print 3

Page	**Error**	**Correction**	**Print corrected**
xxiii	The latest version of Python is Python 3.9.	The latest version of Python is Python 3.11.	Print 3
xxiii	. . . click either **Add Python 3.9 to PATH** or **Add Python to environment variables** . . .	. . . click either **Add Python 3.11 to PATH** or **Add Python to environment variables** . . .	Print 3

3	`Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:30:23)`
	`[MSC v.1928 32 bit (Intel)] on win32`
		`Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35)`
		`[MSC v.1934 64 bit (AMD64)] on win32`
			Print 3

4	`Python 3.9.2 (default,  Mar 15 2021,  17:23:44)`
	`[Clang 11.0.0 (clang-1100.0.33.17)] on darwin`
		`Python 3.11.2 (v3.11.2:878ead1ac1,  Feb  7 2023,  10:02:41)`
		`[Clang 13.0.0 (clang-1300.0.29.30)] on darwin`
			Print 3

5	`Python 3.9.2 (default, Feb 20 2021,  20:57:50)`
	`[GCC 7.5.0] on linux`
		`Python 3.11.2 (main, Feb 8 2023, 14:49:29)`
		`[GCC 7.5.0] on linux`
			Print 3

32	❶ `>>> if apple_total > banana_total:`
	`...     print('A')`
	❷ `... elif banana_total > apple_total:`
	`...     print('B')`
	`... elif apple_total == banana_total:`
	`...     print('T')`
		❶ `>>> if apple_total > banana_total:`
		`...     print('A')`
		❷ `... elif banana_total > apple_total:`
		`...     print('B')`
		❸ `... elif apple_total == banana_total:`
		`...     print('T')`
			Print 3

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

Page	Error	Correction	Print corrected

50–51	`>>> for char in secret_word:`
	`...     print('Letter: ' + char)`
	`...`
	`5 iterations, coming right up!`
		`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:
	`>>> num = 7788`
	`>>> output_file = open('blah.out', 'w')`
	`>>> output_file.write(str(num) + '\n')`
	`5`
		To write a number to a file, convert it to a string first. **You can do that using an f-string**:
		`>>> num = 7788`
		`>>> output_file = open('blah.out', 'w')`
		`>>> output_file.write(f'{num}\n')`
		`5`
			Print 3

180	`for word in words:`
	`  ❺ if chars_on_line + len(word) <= k:`
	`        line = line + word + ' '`
	`        chars_on_line = chars_on_line + len(word)`
	`    else:`
	`      ❻ output_file.write(line[:-1] + '\n')`
	`        line = word + ' '`
	`        chars_on_line = len(word)`
	`❼output_file.write(line[:-1] + '\n')`
		`for word in words:`
		`  ❺ if chars_on_line + len(word) <= k:`
		`        line = line + word + ' '`
		`        chars_on_line = chars_on_line + len(word)`
		`    else:`
		`      ❻ output_file.write(f'{line[:-1]}\n')`
		`        line = word + ' '`
		`        chars_on_line = len(word)`
		`❼output_file.write(f'{line[:-1]}\n')`
			Print 3

181	Second, **you may have expected me to use an f-string here,** like this: `output_file.write(f'{line[:-1]}\n')` **However, at the time of writing, the USACO judge is running an older version of Python that doesn't support f-strings.**	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: `output_file.write(line[:-1] + '\n')`	Print 3

Page	Error	Correction	Print corrected
196, 198	`output_file.write(output + '\n')`	`output_file.write(f'{output}\n')`	Print 3
234	`output_file.write(str(total // 2) + '\n')`	`output_file.write(f'{total // 2}\n')`	Print 3
241	`output_file.write(str(max_covered) + '\n')`	`output_file.write(f'{max_covered}\n')`	Print 3
248	`output_file.write(str(min_cost) + '\n')`	`output_file.write(f'{min_cost}\n')`	Print 3
251	`output_file.write(str(total) + '\n')`	`output_file.write(f'{total}\n')`	Print 3
254	`output_file.write(str(total) + '\n')`	`output_file.write(f'{total}\n')`	Print 3
256	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.	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.	Print 3
261	`output_file.write(str(total) + '\n')`	`output_file.write(f'{total}\n')`	Print 3
271	then **8**n is **4**0,000. The number 8 is so small compared to **4**0,000	then **2**n is **1**0,000. The number 8 is so small compared to **1**0,000	Print 3
277	`output_file.write(str(total) + '\n')`	`output_file.write(f'{total}\n')`	Print 3