Dead Simple Python

Idiomatic Python for the Impatient Programmer

by Jason C. McDonald

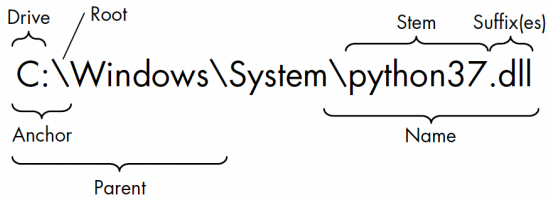
Errata updated to print 2

Page	Error	Correction	Print corrected
5	Nuitka can be used to transpile Python code C and C++ . . .	Nuitka can be used to transpile Python code **to** C and C++ . . .	Print 2
15	On Fedora, RHEL, or CentOS, you can run this: `sudo dnf python3 python3-pip`	On Fedora, RHEL, or CentOS, you can run this: `sudo dnf install python3 python3-pip`	Print 2
30	If I ran the linter again, I'd only see the other **two** linter errors:	If I ran the linter again, I'd only see the other **three** linter errors:	Print 2
47	`foo %= 51 # value is now 42.0 (144.0 % 15)`	On Fedora, RHEL, or CentOS, you can run this: `foo %= 51 # value is now 42.0 (144.0 % 51)`	Print 2
52	**The assignment expression is enclosed in parentheses for readability, although I technically could have omitted them.**	**The parentheses in the assignment expression is important, as it controls what part of the expression is stored as the value of `eggs`. If I omitted the parentheses, the value `True` would be stored instead of an integer.**	Print 2

57	First, if you want to wrap an expression in literal curly braces, you must use two curly braces ({{ }}) for every one you want displayed**:**
	`answer = 42`
	`print(f"{{answer}}") # prints "{42}"`
	`print(f"{{{{answer}}}}") # prints "{{42}}"`
	`print(f"{{{{{{answer}}}}}}") # prints "{{{42}}}"`
		First, if you want to wrap an expression in literal curly braces, you must use two curly braces ({{ }}) for every one you want displayed**, plus an additional pair to enable substitution.**
		`answer = 42`
		`print(f"{{{answer}}}") # prints "{42}"`
		`print(f"{{{{{answer}}}}}") # prints "{{42}}"`
		`print(f"{{{{{{{answer}}}}}}}") # prints "{{{42}}}"`
			Print 2

114	`Hot: ["Lettuce", "Tomato", "Cheese", "Beef", "Salsa"]`
	`Mild: ["Lettuce", "Tomato", "Cheese", "Beef"]`
	`Default: ["Lettuce", "Tomato", "Cheese", "Beef"]`
		`Hot: ["Lettuce", "Tomato", "Beef", "Salsa"]`
		`Mild: ["Lettuce", "Tomato", "Beef"]`
		`Default: ["Lettuce", "Tomato", "Beef"]`
			Print 2

149	In this chapter, I'll cover the essentials of object-oriented programming in Python: creating classes with attributes, **modules**, and properties.	In this chapter, I'll cover the essentials of object-oriented programming in Python: creating classes with attributes, **methods**, and properties.	Pending

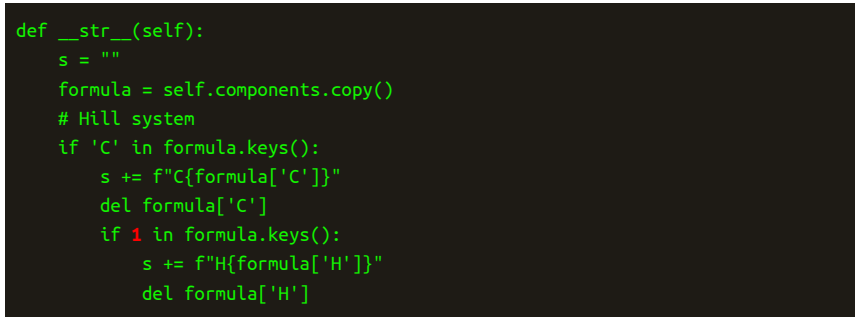
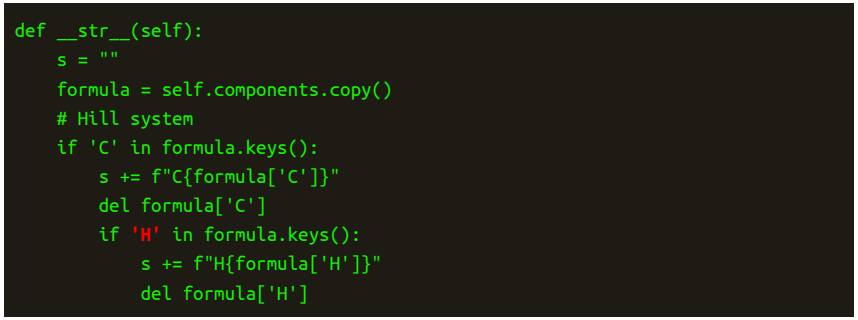
Page	Error	Correction	Print corrected
162	In this case, I assume this is some sort of string, which I run through the **static** method **`_encode()`** I defined earlier and then store in the list `self._secrets`.	In this case, I assume this is some sort of string, which I run through the **class** method **`encrypt()`** I defined earlier and then store in the list `self._secrets`.	Print 2
162	You actually don't need to define a deleter if you have no need for special behavior when the **decorator** is deleted. Consider what you want to happen if `del` is called on your **decorator**, such as when you are deleting an associated attribute that the property controls; if you can't think of anything, skip writing the deleter.	You actually don't need to define a deleter if you have no need for special behavior when the **property** is deleted. Consider what you want to happen if `del` is called on your **property**, such as when you are deleting an associated attribute that the property controls; if you can't think of anything, skip writing the deleter.	Print 2
184	**If** case *exceptions* . . .	**In** case *exceptions* . . .	Print 2
224	Insertion	`Counter` is designed specifically for counting hashable objects; the object is the key, and the count is an integer value. Other languages call this type of collection a *multiset*. **Multisets are not the same as counters, but are sometimes used in place of them, as a side effect of how multisets work.**	Print 2
318	Figure update	 *Figure 11-1: Parts of a Windows absolute path*	Print 2
326	`path.touch()` Creates an empty file at `path`. **Normally, nothing happens if it already exists.** If the optional `exist_ok=` argument is `False` and the file exists, a `FileExistsError` is raised.	`path.touch()` Creates an empty file at `path`. **If one already exists, it updates the access timestamp on file, but does nothing else.** If the optional `exist_ok=` argument is `False` and the file exists, a `FileExistsError` is raised.	Print 2
358	`left = int.from_bytes(left, byteorder=byteorder)` `right = int.from_bytes(right, byteorder=byteorder)`	`left = int.from_bytes(left, byteorder, signed=False)` `right = int.from_bytes(right, byteorder, signed=False)`	Print 2

359	`result = left & right`
	`return result.to_bytes(size, byteorder, signed=True)`
	Listing 12-38: bitwise_via_int.py:3
	I bind the result of the bitwise operation to result. Finally, I convert `result` back to a `bytes` object, using the `size` I determined earlier, **the `byteorder` passed to my function, and `signed=True` to handle conversion of any possible negative integer values. I return the resulting bytes-like object**.
		`result = left & right`
		`return result.to_bytes(size, byteorder, signed=False)`
		Listing 12-38: bitwise_via_int.py:3
		I bind the result of the bitwise operation to result. Finally, I convert `result` back to a `bytes` object, using the `size` I determined earlier, **and the `byteorder` passed to my function. I can safely assume `signed=False`, as `left` and `right` can only ever be positive integers**.
			Print 2

Page	Error	Correction	Print corrected

450	`from functools import singledispatchmethod`
	`from typing import overload`
	`class Element:`
	`  # --snip--`
		`from functools import singledispatchmethod`
		`class Element:`
		`  # --snip--`
			Print 2

450–451	In this case, I'll create **two** more versions of the function: one that works with a string argument **and another that works with either an integer or a floating-point number argument**:
	`  @__eq__.register`
	`  def _(self, other: str):`
	`  return self.symbol == other`
	`  @overload`
	`  def _(self, other: float):`
	`  ...`
	`  @__eq__.register`
	`  def _(self, other: int):`
	`  return self.number == other`
	The first of these methods accepts a string argument. The first parameter, the one being switched on, is annotated with a type hint for the expected type, which is a string (`str`) in this first case.
	The second method here accepts either an integer or a float, and it is made possible with the `@typing.overload` decorator. When type hinting, you can mark one or more function headings with `@overload`, to indicate that they overload an upcoming function or method with the same name. The *Ellipsis* (...) is used in place of the suite of the overloaded method, so it can instead share the suite of the method below it. The function or method not decorated with `@overload` must come immediately after all the overloaded versions thereof.
		In this case, I'll create **three** more versions of the function: one that works with a string argument**, another that works with a floating-point number, and a third with an integer**:
		`  @__eq__.register`
		`  def _(self, other: str):`
		`  return self.symbol == other`
		`  @__eq__.register`
		`  def _(self, other: float):`
		`  return self.number == other`
		`  @__eq__.register`
		`  def _(self, other: int):`
		`  return self.number == other`
		The first of these methods accepts a string argument. The first parameter, the one being switched on, is annotated with a type hint for the expected type, which is a string (`str`) in this first case. **The second method here accepts a `float`, and the third an `int`.**
		When type hinting, you can **ordinarily** mark one or more function headings with **a special `@typing.overload`**, to indicate that they overload an upcoming function or method with the same name. The *Ellipsis* (...) is used in place of the suite of the overloaded method, so it can instead share the suite of the method below it. The function or method not decorated with `@overload` must come immediately after all the overloaded versions thereof. **I first thought to use this here, since the second and third functions had the same body. Unfortunately, `@overload` does not work with other decorators, so I could not use this technique here.**
			Print 2

Page	Error	Correction	Print corrected
453			Print 2
627	It can also be used on a number of Raspberry Pi and **Ardunio** microcontrollers, as well as hardware from many other brands.	It can also be used on a number of Raspberry Pi and **Arduino** microcontrollers, as well as hardware from many other brands.	Print 2