

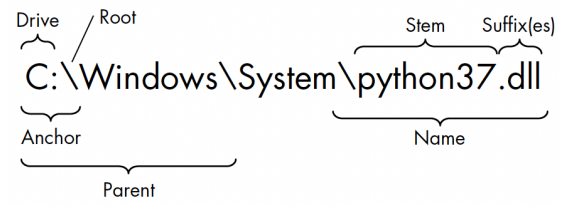
# Dead Simple Python

## Idiomatic Python for the Impatient Programmer

by Jason C. McDonald

errata updated to print 1

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 <b>to</b> C and C++ . . .	Print 2
5	On Fedora, RHEL, or CentOS, you can run this: <pre>sudo dnf python3 python3-pip</pre>	On Fedora, RHEL, or CentOS, you can run this: <pre>sudo dnf install python3 python3-pip</pre>	Print 2
30	If I ran the linter again, I'd only see the other <b>two</b> linter errors:	If I ran the linter again, I'd only see the other <b>three</b> linter errors:	Print 2
47	<pre>foo %= 51 # value is now 42.0 (144.0 % 15)</pre>	On Fedora, RHEL, or CentOS, you can run this: <pre>foo %= 51 # value is now 42.0 (144.0 % 51)</pre>	Print 2
52	<b>The assignment expression is enclosed in parentheses for readability, although I technically could have omitted them.</b>	<b>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.</b>	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: <pre>answer = 42 print(f"{{answer}}") # prints "{42}" print(f"{{{answer}}}") # prints "{{42}}" print(f"{{{answer}}}") # prints "{{{42}}}"</pre>	First, if you want to wrap an expression in literal curly braces, you must use two curly braces ({{ }}) for every one you want displayed, <b>plus an additional pair to enable substitution.</b> <pre>answer = 42 print(f"{{answer}}") # prints "{42}" print(f"{{{answer}}}") # prints "{{42}}" print(f"{{{answer}}}") # prints "{{{42}}}" print(f"{{{answer}}}") # prints "{{{{{42}}}"</pre>	Print 2
114	<pre>Hot: ["Lettuce", "Tomato", "Cheese", "Beef", "Salsa"] Mild: ["Lettuce", "Tomato", "Cheese", "Beef"] Default: ["Lettuce", "Tomato", "Cheese", "Beef"]</pre>	<pre>Hot: ["Lettuce", "Tomato", "Beef", "Salsa"] Mild: ["Lettuce", "Tomato", "Beef"] Default: ["Lettuce", "Tomato", "Beef"]</pre>	Print 2
162	In this case, I assume this is some sort of string, which I run through the <b>static</b> method <code>_encode()</code> I defined earlier and then store in the list <code>self._secrets</code> .	In this case, I assume this is some sort of string, which I run through the <b>class</b> method <code>encrypt()</code> I defined earlier and then store in the list <code>self._secrets</code> .	Print 2

Page	Error	Correction	Print corrected
162	You actually don't need to define a deleter if you have no need for special behavior when the <b>decorator</b> is deleted. Consider what you want to happen if del is called on your <b>decorator</b> , 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 <b>property</b> is deleted. Consider what you want to happen if del is called on your <b>property</b> , 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	<b>If</b> case <i>exceptions</i> . . .	<b>In</b> case <i>exceptions</i> . . .	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 <i>multiset</i> . <b>Multisets are not the same as counters, but are sometimes used in place of them, as a side effect of how multisets work.</b>	Print 2
318	Figure update	 <p>Figure 11-1: Parts of a Windows absolute path</p>	Print 2
326	path.touch() Creates an empty file at path. <b>Normally, nothing happens if it already exists.</b> If the optional exist_ok= argument is False and the file exists, a FileExistsError is raised.	path.touch() Creates an empty file at path. <b>If one already exists, it updates the access timestamp on file, but does nothing else.</b> If the optional exist_ok= argument is False and the file exists, a FileExistsError is raised.	Print 2
358	<pre>left = int.from_bytes(left, byteorder=byteorder) right = int.from_bytes(right, byteorder=byteorder)</pre>	<pre>left = int.from_bytes(left, byteorder, signed=False) right = int.from_bytes(right, byteorder, signed=False)</pre>	Print 2
359	<pre>result = left &amp; right return result.to_bytes(size, byteorder, signed=True)</pre> <p>Listing 12-38: bitwise_via_int.py:3</p> <p>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, <b>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.</b></p>	<pre>result = left &amp; right return result.to_bytes(size, byteorder, signed=False)</pre> <p>Listing 12-38: bitwise_via_int.py:3</p> <p>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, <b>and the byteorder passed to my function. I can safely assume signed=False, as left and right can only ever be positive integers.</b></p>	Print 2

Page	Error	Correction	Print corrected
450	<pre data-bbox="174 186 1012 326"> from functools import singledispatchmethod from typing import overload class Element:     # --snip-- </pre>	<pre data-bbox="1041 186 1885 293"> from functools import singledispatchmethod class Element:     # --snip-- </pre>	Print 2
451	<p data-bbox="174 370 1012 451">In this case, I'll create <b>two</b> more versions of the function: one that works with a string argument <b>and another that works with either an integer or a floating-point number</b> argument:</p> <pre data-bbox="174 475 1012 784"> @_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 </pre> <p data-bbox="174 805 1012 886">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 (<code>str</code>) in this first case.</p> <p data-bbox="174 889 1012 1076">The second method here accepts either an integer or a float, and it is made possible with the <code>@typing.overload</code> decorator. When type hinting, you can mark one or more function headings with <b>@overload</b>, to indicate that they overload an upcoming function or method with the same name. The <i>Ellipsis</i> (...) 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 <code>@overload</code> must come immediately after all the overloaded versions thereof.</p>	<p data-bbox="1041 370 1885 451">In this case, I'll create <b>three</b> more versions of the function: one that works with a string argument, <b>another that works with a floating-point number, and a third with an integer</b>:</p> <pre data-bbox="1041 475 1885 813"> @_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 </pre> <p data-bbox="1041 834 1885 943">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 (<code>str</code>) in this first case. <b>The second method here accepts a float, and the third an int.</b></p> <p data-bbox="1041 946 1885 1162">When type hinting, you can <b>ordinarily</b> mark one or more function headings with a <b>special @typing.overload</b>, to indicate that they overload an upcoming function or method with the same name. The <i>Ellipsis</i> (...) 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 <code>@overload</code> must come immediately after all the overloaded versions thereof. <b>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.</b></p>	Print 2

Page	Error	Correction	Print corrected
453	<pre data-bbox="174 201 1012 496">def __str__(self):     s = ""     formula = self.components.copy()     # Hill system     if 'C' in formula.keys():         s += f"C{formula['C']}"         del formula['C']     if 'H' in formula.keys():         s += f"H{formula['H']}"         del formula['H']</pre>	<pre data-bbox="1041 201 1879 496">def __str__(self):     s = ""     formula = self.components.copy()     # Hill system     if 'C' in formula.keys():         s += f"C{formula['C']}"         del formula['C']     if 'H' in formula.keys():         s += f"H{formula['H']}"         del formula['H']</pre>	Print 2
627	<p data-bbox="174 542 1012 597">It can also be used on a number of Raspberry Pi and <b>Arduinio</b> microcontrollers, as well as hardware from many other brands.</p>	<p data-bbox="1041 542 1879 597">It can also be used on a number of Raspberry Pi and <b>Arduino</b> microcontrollers, as well as hardware from many other brands.</p>	Print 2