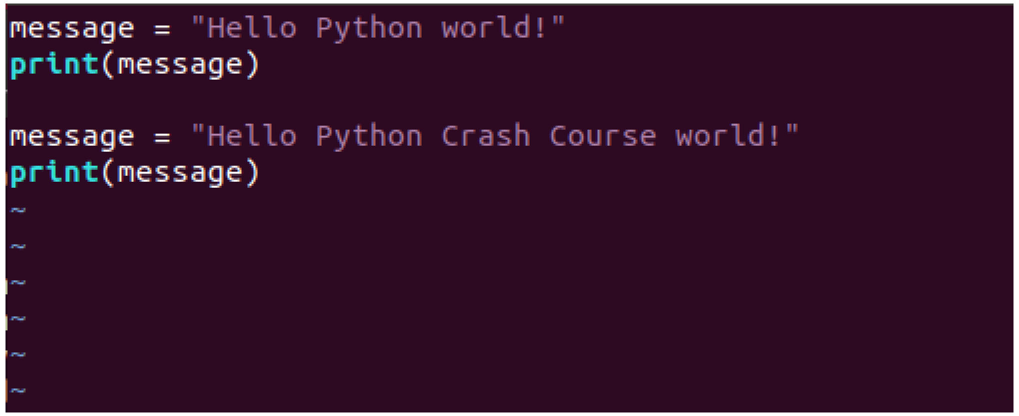
Variables and Simple Data Types

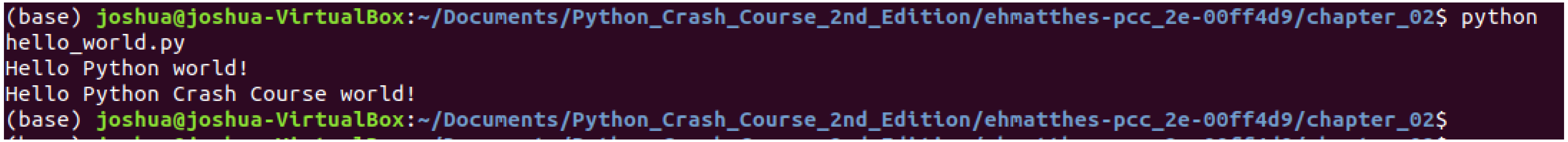
Variables

- Let’s try using a variable in `hello_world.py`.



```
message = "Hello Python world!"
print(message)

message = "Hello Python Crash Course world!"
print(message)
~
~
~
~
~
~
```



```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$ python
hello_world.py
Hello Python world!
Hello Python Crash Course world!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$
```

- Every variable is connected to a *value*, which is the information associated with that variable.
- You can change the value of a variable in your program at any time, and Python will always keep track of its current value.

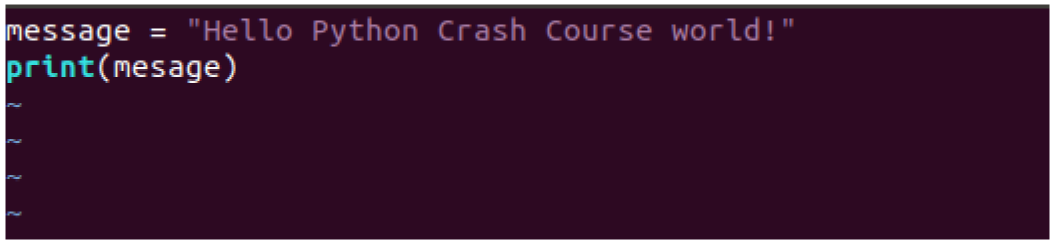
Naming and Using Variables

- When you’re using variables in Python, you need to adhere to a few rules and guidelines.
- Breaking some of these rules will cause errors; other guidelines just help you write code that’s easier to read and understand.

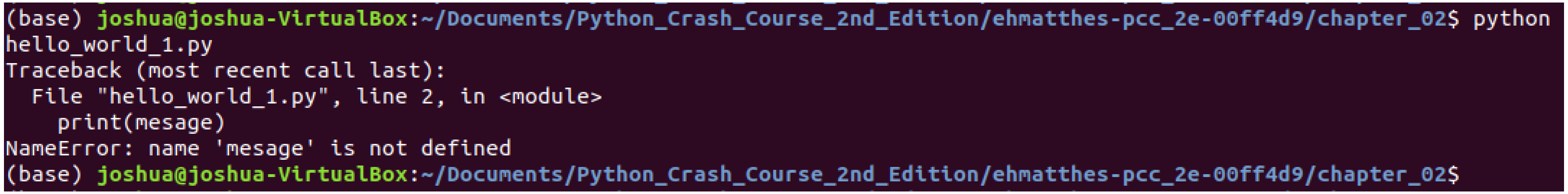
- Be sure to keep the following variable rules in mind:
  - Variable names can contain only letters, numbers, and underscores. They can start with a letter or an underscore, but not with a number. For instance, you can call a variable *message_1* but not *1_message*.
  - Spaces are not allowed in variable names, but underscores can be used to separate words in variable names. For example, *greeting_message* works, but *greeting message* will cause errors.
  - Avoid using Python keywords and function names as variable names; that is, do not use words that Python has reserved for a particular programmatic purpose, such as the word `print`.
  - Variable names should be short but descriptive. For example, *name* is better than *n*, *student_name* is better than *s_n*, and *name_length* is better than *length_of_persons_name*.
  - Be careful when using the lowercase letter *`l`* and the uppercase letter *`O`* because they could be confused with the numbers *`1`* and *`0`*.

Avoiding Name Errors When Using Variables

- We’ll write some code that generates an error on purpose.

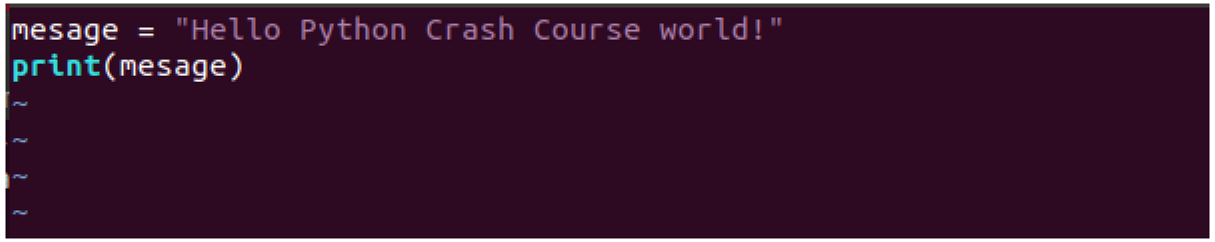


```
message = "Hello Python Crash Course world!"
print(mesage)
~
~
~
~
```

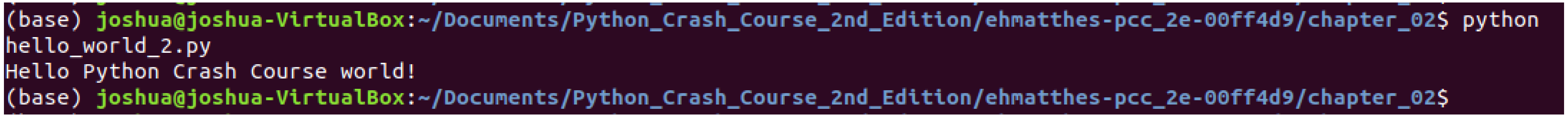


```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$ python
hello_world_1.py
Traceback (most recent call last):
  File "hello_world_1.py", line 2, in <module>
    print(mesage)
NameError: name 'mesage' is not defined
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$
```

- The Python interpreter doesn’t spellcheck your code, but it does ensure that variable names are spelled consistently.



```
mesage = "Hello Python Crash Course world!"
print(mesage)
~
~
~
~
```



```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$ python
hello_world_2.py
Hello Python Crash Course world!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$
```

- Programming languages are strict, but they disregard good and bad spelling.
- As a result, you don’t need to consider English spelling and grammar rules when you’re trying to create variable names and writing code.

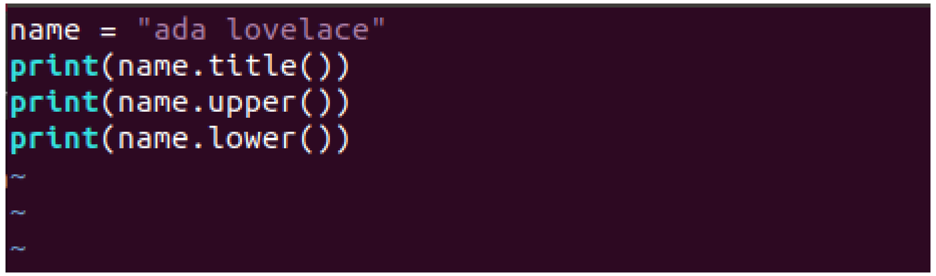
Strings

- A *string* is a series of characters.
- Anything inside quotes is considered a string in Python, and you can use single or double quotes around your strings like this:

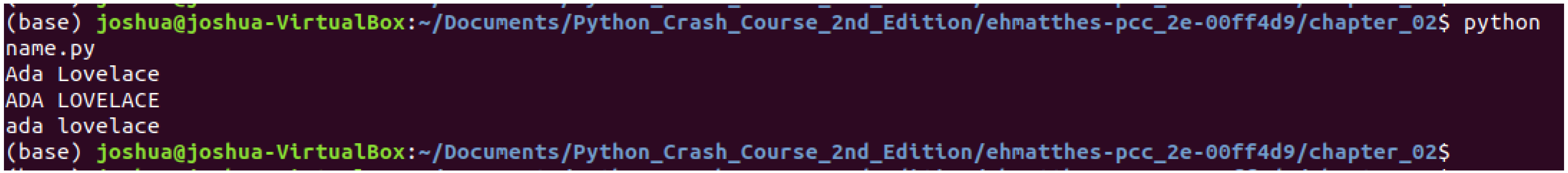
```
"This is a string."
'This is also a string.'
```

- This flexibility allows you to use quotes and apostrophes within your strings:
  - `'I told my friend, "Python is my favorite language!"'`
  - `"The language 'Python' is named after Monty Python, not the snake."`
  - `"One of Python's strengths is its diverse and supportive community."`

Changing Case in a String with Methods

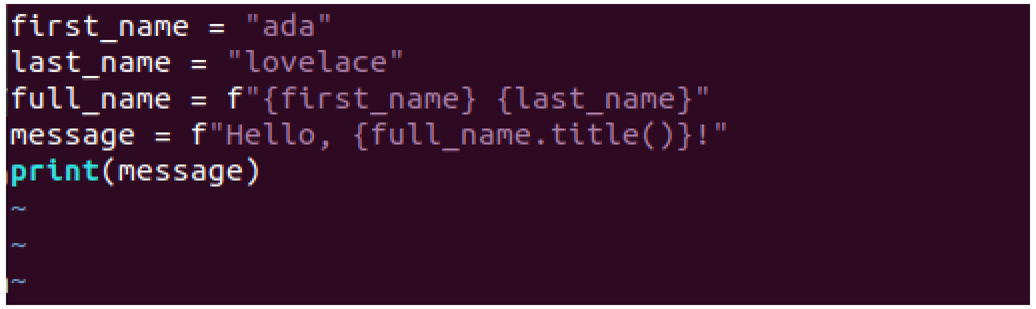


```
name = "ada lovelace"
print(name.title())
print(name.upper())
print(name.lower())
~
~
~
```

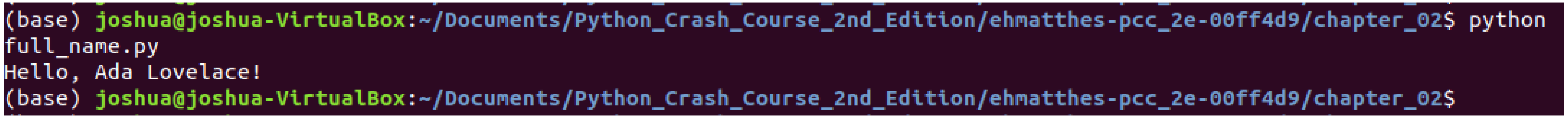


```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$ python
name.py
Ada Lovelace
ADA LOVELACE
ada lovelace
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$
```

Using Variables in Strings



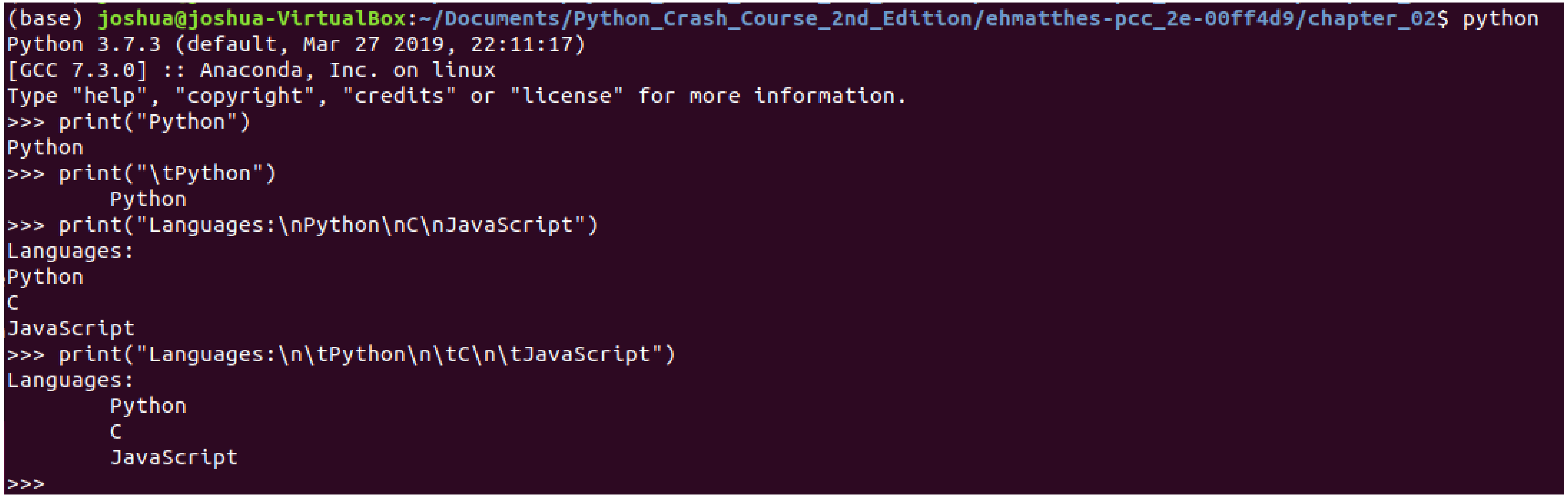
```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
message = f"Hello, {full_name.title()}!"
print(message)
~
~
~
```



```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$ python
full_name.py
Hello, Ada Lovelace!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$
```

- These strings are called *f-strings*.
- The *f* is for *format*, because Python formats the string by replacing the name of any variable in braces with its value.

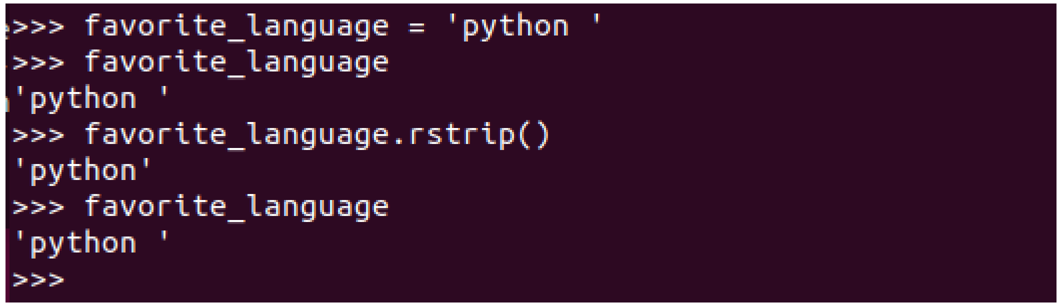
Adding Whitespace to Strings with Tabs or Newlines



```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$ python
Python 3.7.3 (default, Mar 27 2019, 22:11:17)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Python")
Python
>>> print("\tPython")
        Python
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
Python
C
JavaScript
>>> print("Languages:\n\tPython\n\tC\n\tJavaScript")
Languages:
        Python
        C
        JavaScript
>>>
```

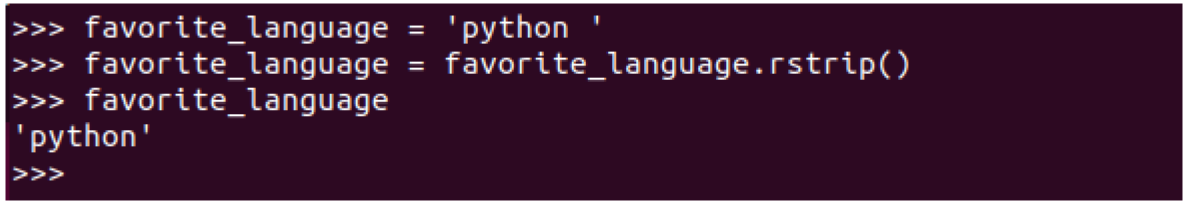
Stripping Whitespace

- It’s important to think about whitespace, because often you’ll want to compare two strings to determine whether they are the same.
- For example, one important instance might involve checking people’s usernames when they log in to a website.



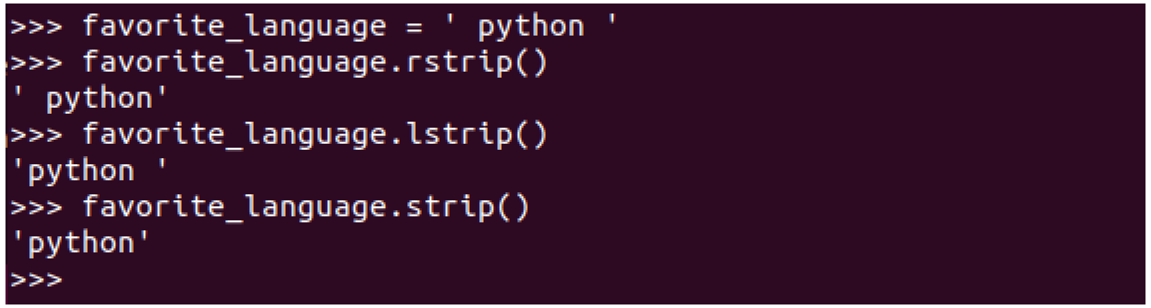
```
>>> favorite_language = 'python '
>>> favorite_language
'python '
>>> favorite_language.rstrip()
'python'
>>> favorite_language
'python '
>>>
```

- To remove the whitespace from the string permanently, you have to associate the stripped value with the variable name:



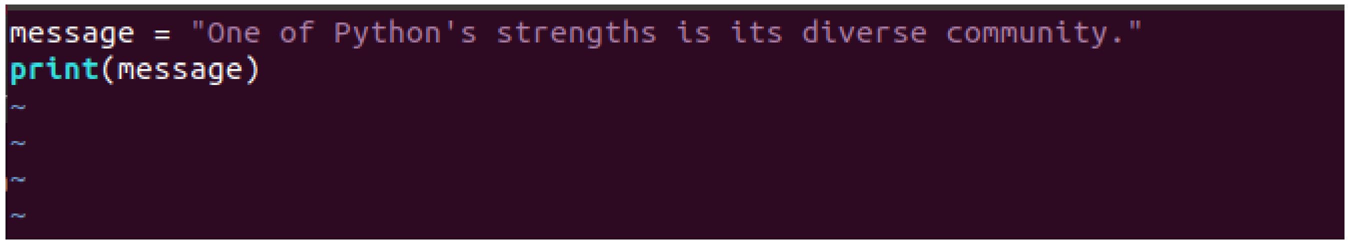
```
>>> favorite_language = 'python '
>>> favorite_language = favorite_language.rstrip()
>>> favorite_language
'python'
>>>
```

- You can also strip whitespace from the left side of a string using the `lstrip()` method, or from both sides at once using `strip()`:

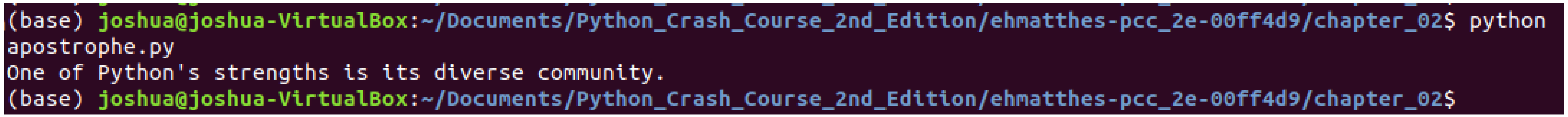


```
>>> favorite_language = ' python '
>>> favorite_language.rstrip()
' python'
>>> favorite_language.lstrip()
'python '
>>> favorite_language.strip()
'python'
>>>
```

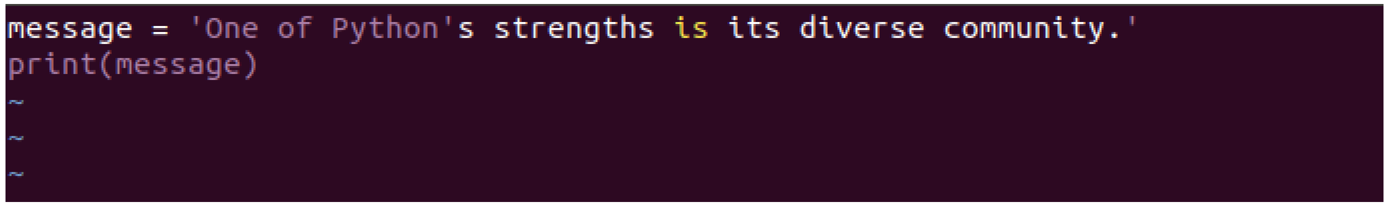
Avoiding Syntax Errors with Strings



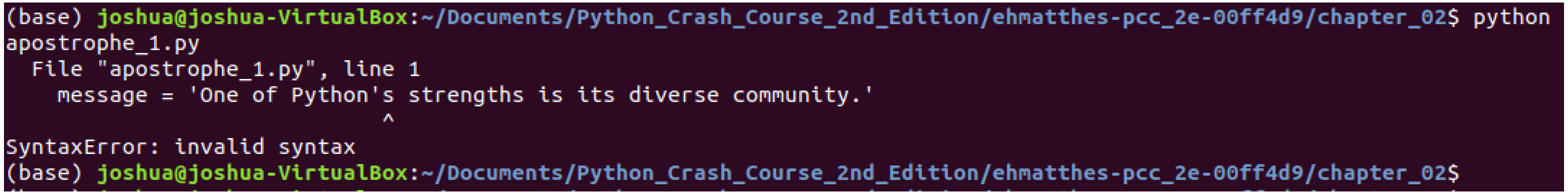
```
message = "One of Python's strengths is its diverse community."
print(message)
~
~
~
~
```



```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$ python
apostrophe.py
One of Python's strengths is its diverse community.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$
```



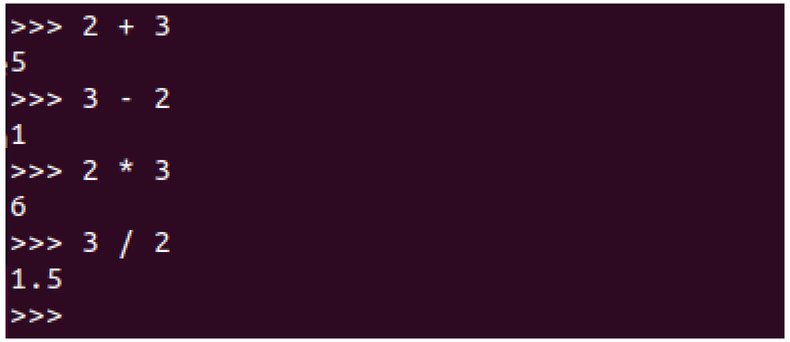
```
message = 'One of Python's strengths is its diverse community.'
print(message)
~
~
~
```



```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$ python
apostrophe_1.py
  File "apostrophe_1.py", line 1
    message = 'One of Python's strengths is its diverse community.'
                             ^
SyntaxError: invalid syntax
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$
```

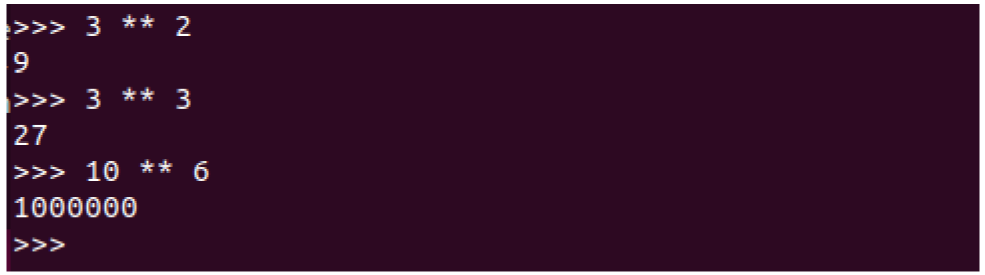
Numbers

Integers



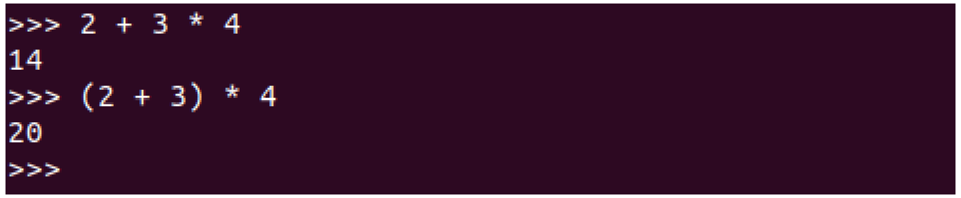
```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
>>>
```

- Python uses two multiplication symbols to represent exponents:



```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
>>>
```

- Python supports the order of operations too, so you can use multiple operations in one expression.
- You can also use parentheses to modify the order of operations so Python can evaluate your expression in the order you specify.

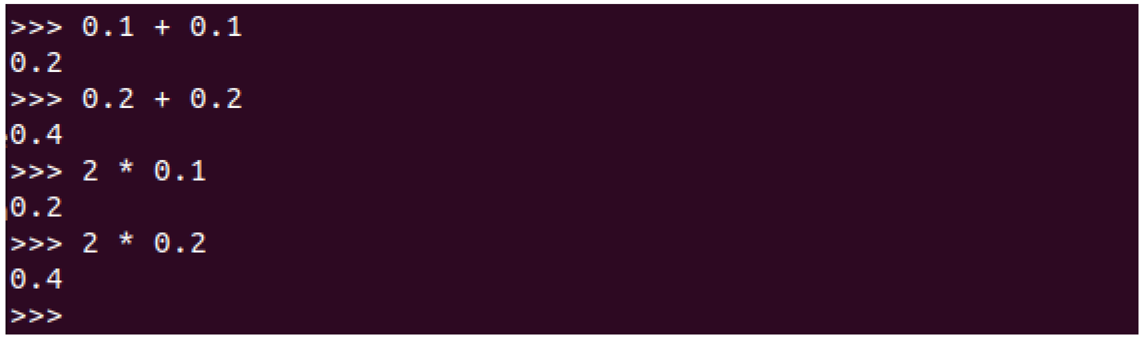


```
>>> 2 + 3 * 4
14
>>> (2 + 3) * 4
20
>>>
```

- The spacing in these examples has no effect on how Python evaluates the expressions; it simply helps you more quickly spot the operations that have priority when you’re reading through the code.

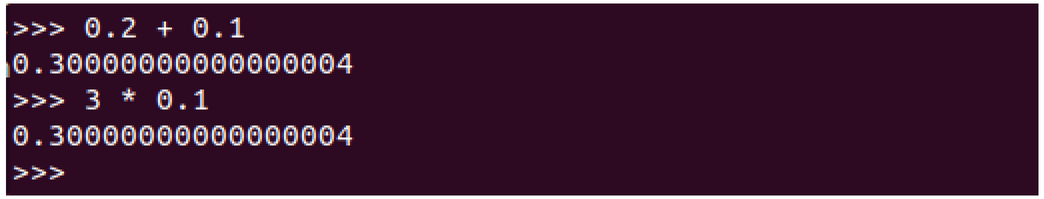
Floats

- Python calls any number with a decimal point a float.
- For the most part, you can use decimals without worrying about how they behave.



```
>>> 0.1 + 0.1
0.2
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4
>>>
```

- But be aware that you can sometimes get an arbitrary number of decimal places in your answer:

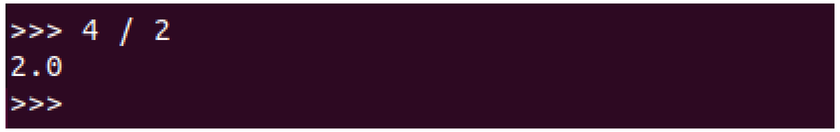


```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
>>>
```

- This happens in all languages and is of little concern.
- Python tries to find a way to represent the result as precisely as possible, which is sometimes difficult given how computers have to represent numbers internally.

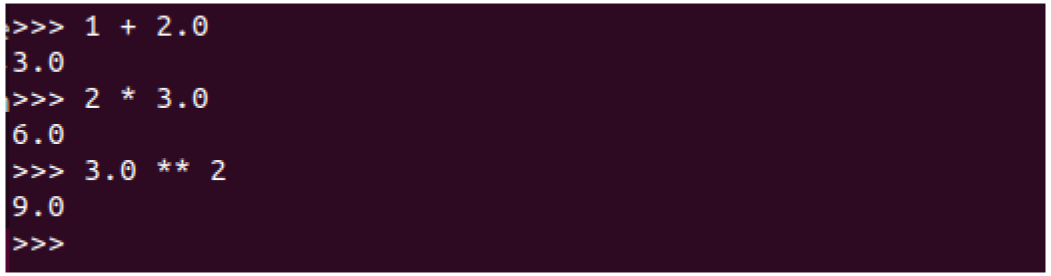
Integers and Floats

- When you divide any two numbers, even if they are integers that result in a whole number, you’ll always get a float:



```
>>> 4 / 2
2.0
>>>
```

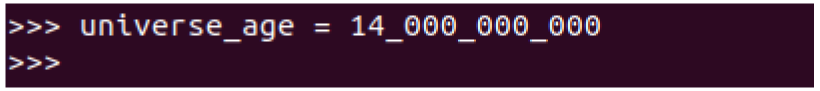
- If you mix an integer and a float in any other operation, you’ll get a float as well:



```
>>> 1 + 2.0
3.0
>>> 2 * 3.0
6.0
>>> 3.0 ** 2
9.0
>>>
```

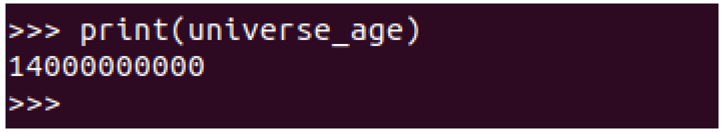
Underscores in Numbers

- When you’re writing long numbers, you can group digits using underscores to make large numbers more readable:



```
>>> universe_age = 14_000_000_000
>>>
```

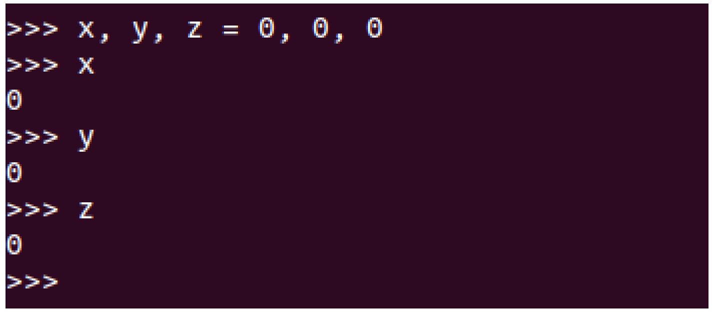
- When you print a number that was defined using underscores, Python prints only the digits:



```
>>> print(universe_age)
14000000000
>>>
```

Multiple Assignment

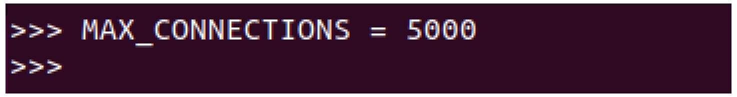
- You can assign values to more than one variable using just a single line.



```
>>> x, y, z = 0, 0, 0
>>> x
0
>>> y
0
>>> z
0
>>>
```

Constants

- A *constant* is like a variable whose value stays the same throughout the life of a program.
- Python doesn’t have built-in constant types, but Python programmers use all capital letters to indicate a variable should be treated as a constant and never be changed:

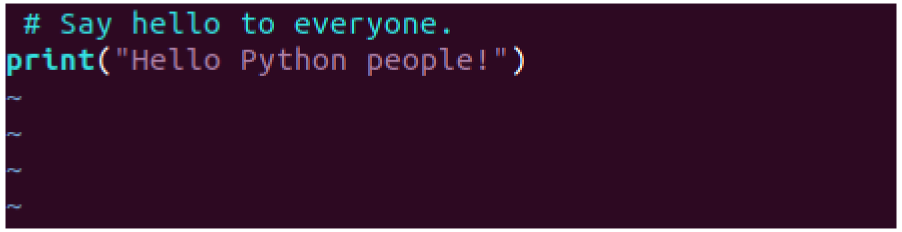


```
>>> MAX_CONNECTIONS = 5000
>>>
```

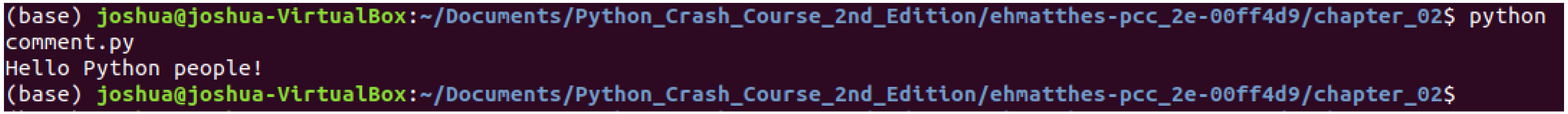
Comments

How Do You Write Comments?

- In Python, the hash mark (#) indicates a comment.
- Anything following a hash mark in your code is ignored by the Python interpreter.



```
 # Say hello to everyone.
print("Hello Python people!")
~
~
~
~
```



```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$ python
comment.py
Hello Python people!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_02$
```

What Kind of Comments Should You Write?

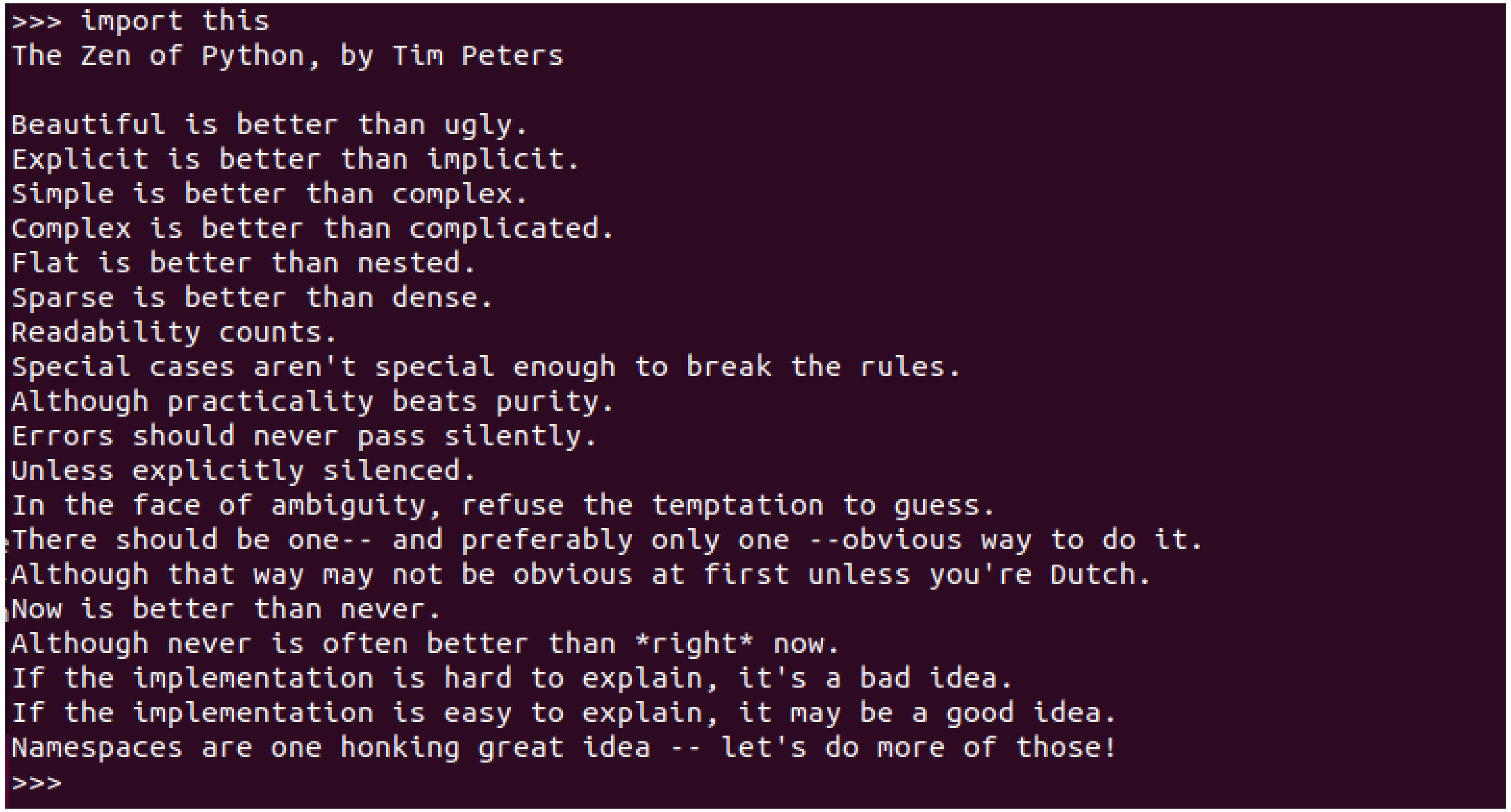
- The main reason to write comments is to explain what your code is supposed to do and how you are making it work.
- When you're in the middle of working on a project, you understand how all of the pieces fit together.
- But when you return to a project after some time away, you'll likely have forgotten some of the details.
- You can always study your code for a while and figure out how segments were supposed to work, but writing good comments can save you time by summarizing your overall approach in clear English.

- If you want to become a professional programmer or collaborate with other programmers, you should write meaningful comments.
- Today, most software is written collaboratively, whether by a group of employees at one company or a group of people working together on an open source project.
- Skilled programmers expect to see comments in code, so it’s best to start adding descriptive comments to your programs now.
- Writing clear, concise comments in your code is one of the most beneficial habits you can form as a new programmer.

- When you’re determining whether to write a comment, ask yourself if you had to consider several approaches before coming up with a reasonable way to make something work; if so, write a comment about your solution.
- It’s much easier to delete extra comments later on than it is to go back and write comments for a sparsely commented program.

The Zen of Python

- Experienced Python programmers will encourage you to avoid complexity and aim for simplicity whenever possible.
- The Python community's philosophy is contained in “The Zen of Python” by Tim Peters.
- You can access this brief set of principles for writing good Python code by entering `import this` into your interpreter.



```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```