Tips:

The most straightforward solution is to use a third variable to temporarily store one of the old values. e.g.:

tmp = a

a = b

b = tmp

If you've read lots of Python code, you might have seen the following trick to swap two variables in one line:

a, b = b, a

2.Functions that operate on other functions are called "Higher order functions."

Functions that operate on other functions are called "Higher order functions." You probably won't write your own for a little while. But there are higher order functions built into Python that you might find useful to call.

Here's an interesting example using the max function.

By default, max returns the largest of its arguments. But if we pass in a function using the optional key argument, it returns the argument x that maximizes key(x) (aka the 'argmax').

In [15]:

def mod\_5(x):

*"""Return the remainder of x after dividing by 5"""*

return x % 5

print(

'Which number is biggest?',

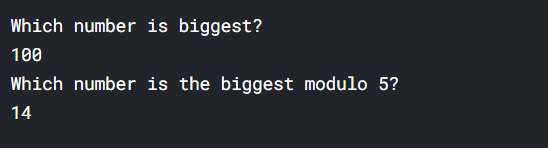
max(100, 51, 14),

'Which number is the biggest modulo 5?',

max(100, 51, 14, key=mod\_5),

sep='**\n**',

)



Adding optional arguments with default values to the functions we define turns out to be pretty easy:

In [13]:

def greet(who="Colin"):

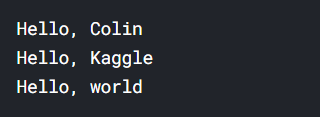
print("Hello,", who)

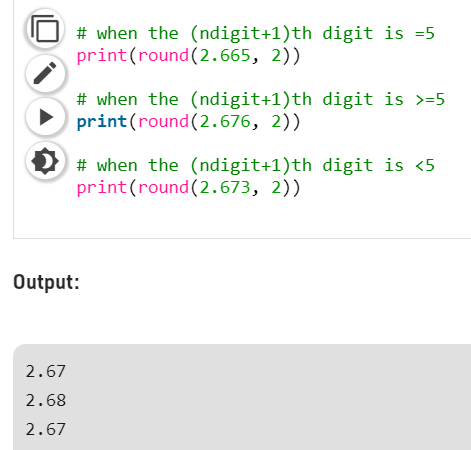
greet()

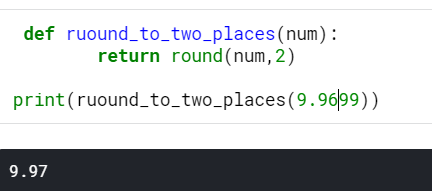
greet(who="Kaggle")

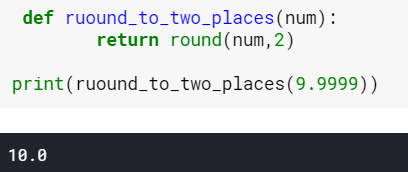
*# (In this case, we don't need to specify the name of the argument, because it's unambiguous.)*

greet("world")

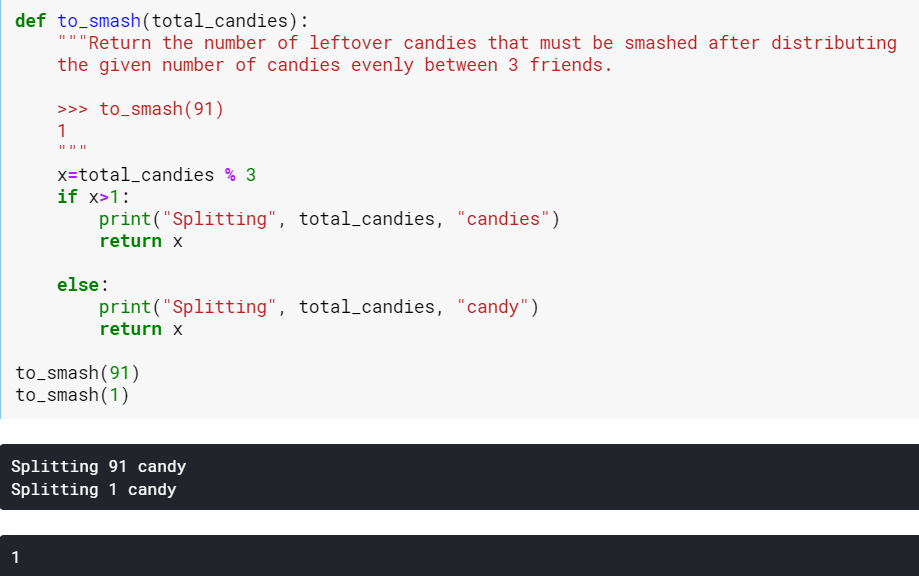


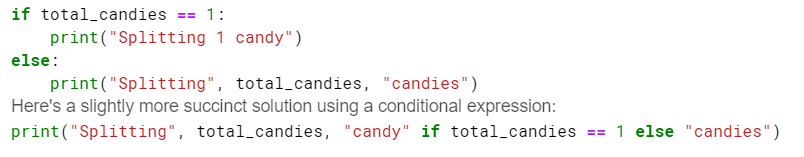






<https://docs.python.org/3/reference/expressions.html#operator-precedence>



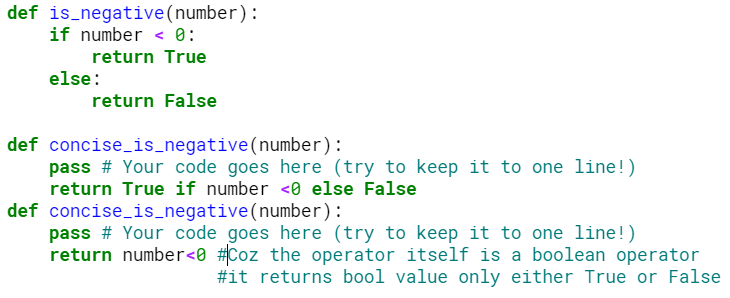


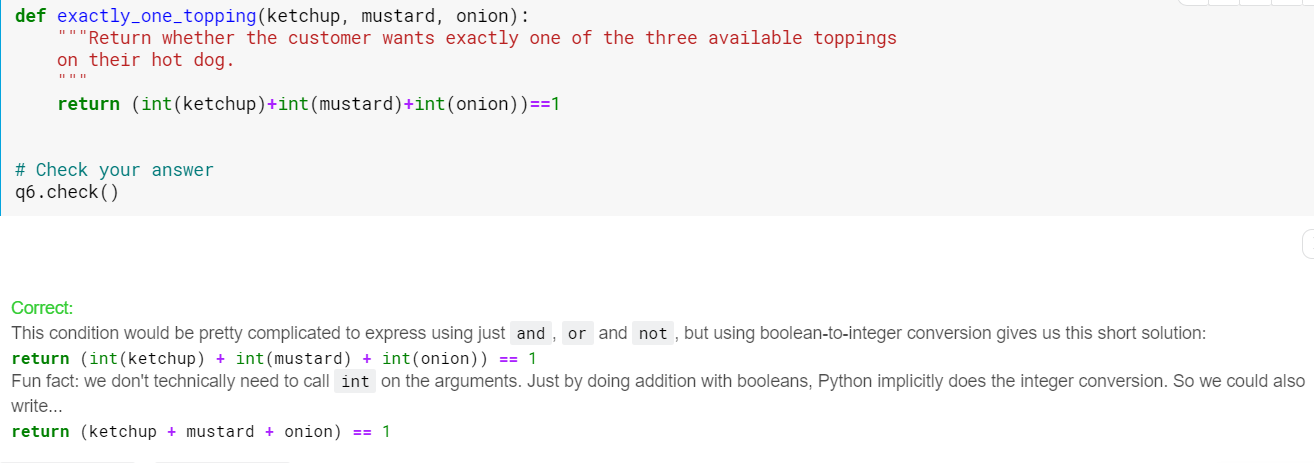
<https://www.geeksforgeeks.org/ternary-operator-in-python/>

The pass statement is used as a placeholder for future code.

When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.

Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

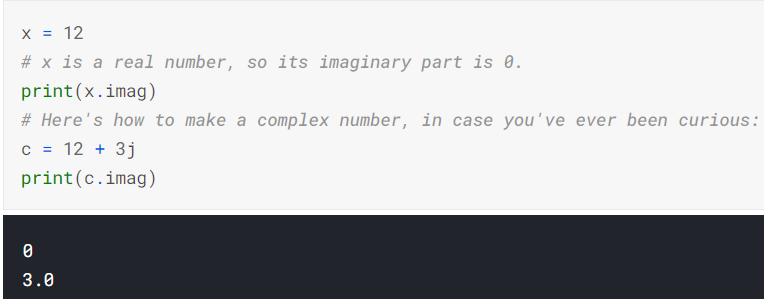




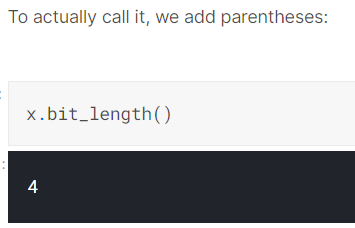
*everything* in Python is an object.

In short, objects carry some things around with them. You access that stuff using Python's dot syntax.

For example, numbers in Python carry around an associated variable called imag representing their imaginary part..

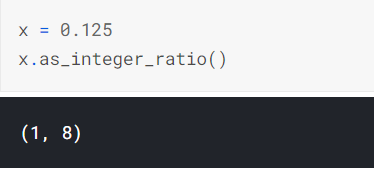
The things an object carries around can also include functions. A function attached to an object is called a method. (Non-function things attached to an object, such as imag, are called *attributes*).

For example, numbers have a method called bit\_length. Again, we access it using dot syntax:

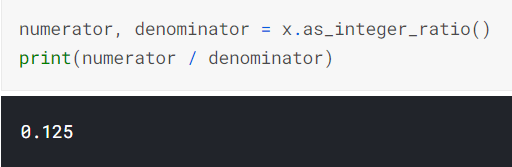


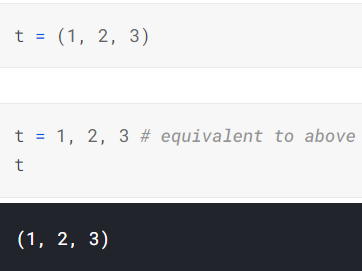
Tuples are often used for functions that have multiple return values.

For example, the as\_integer\_ratio() method of float objects returns a numerator and a denominator in the form of a tuple:

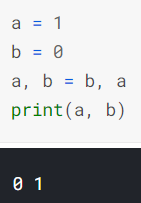


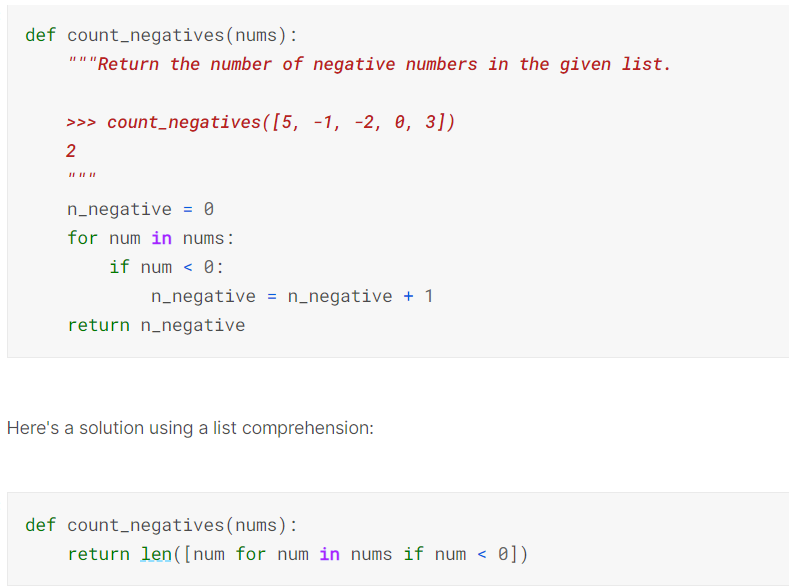
These multiple return values can be individually assigned as follows:

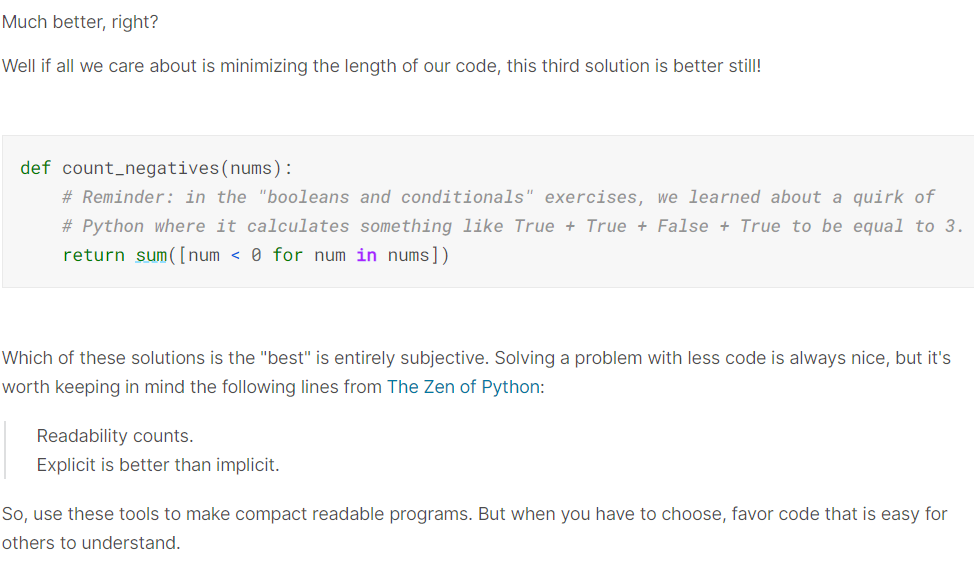


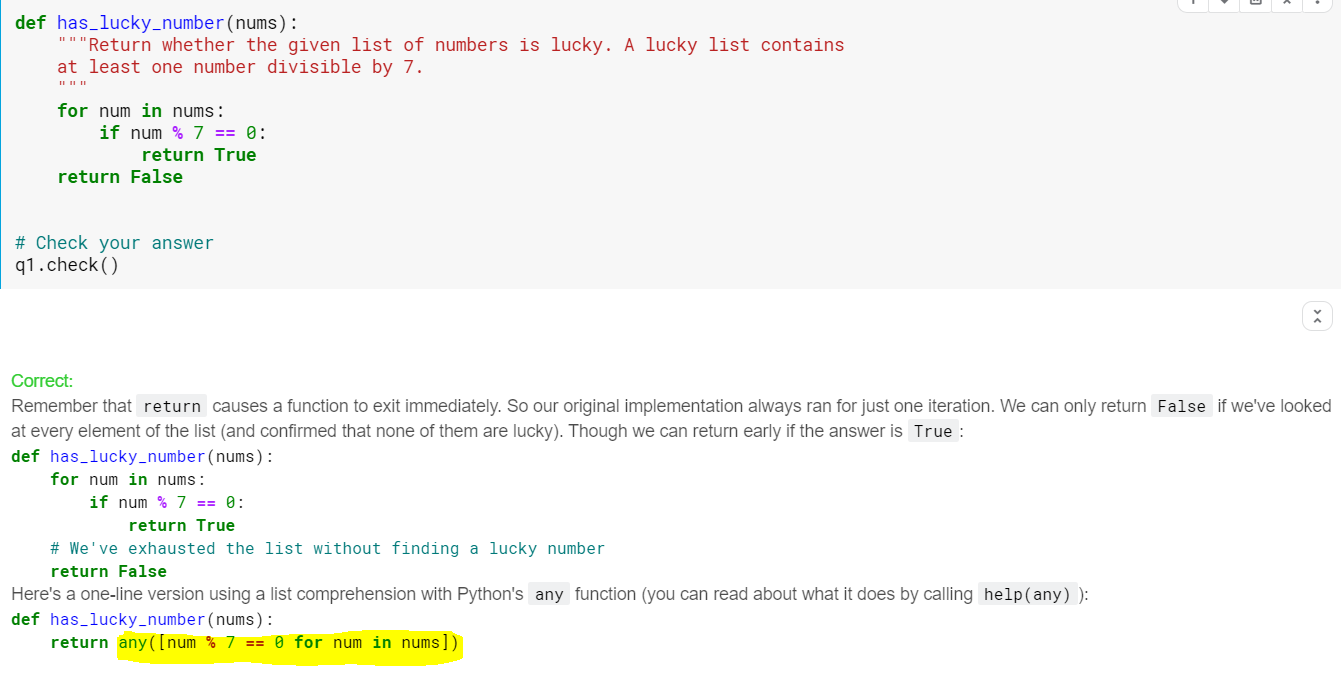


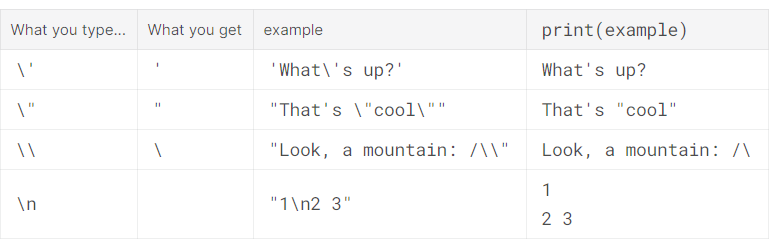
Finally we have some insight into the classic Stupid Python Trick™ for swapping two variables!

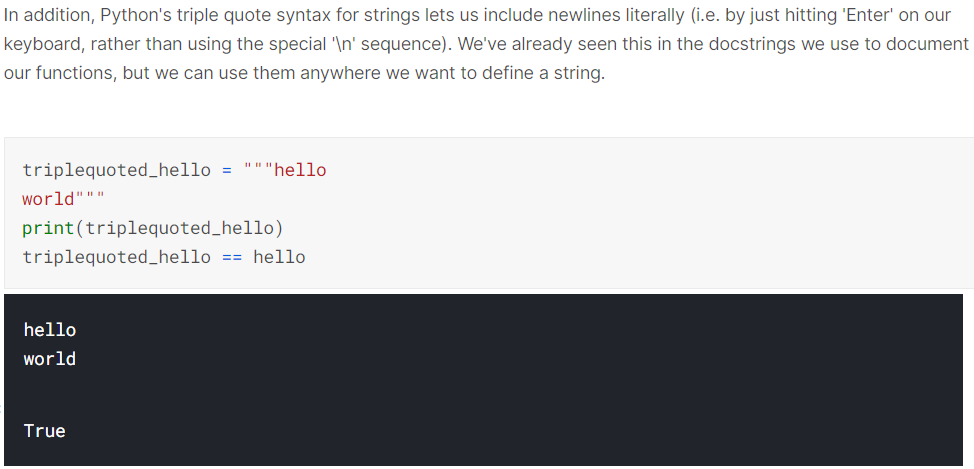


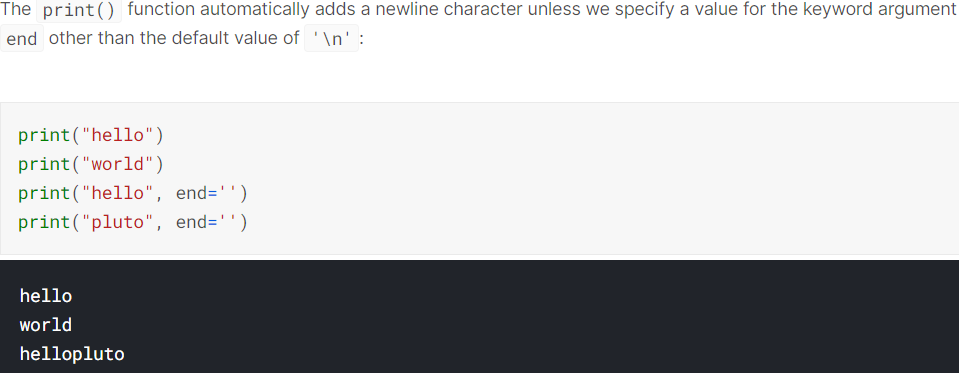


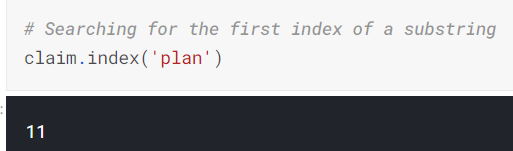








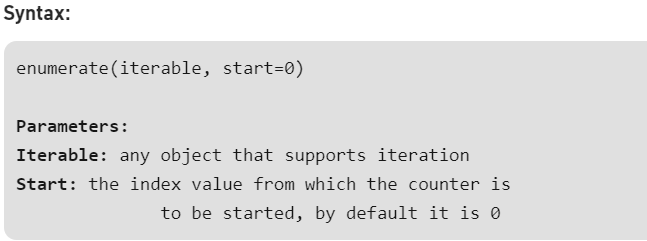


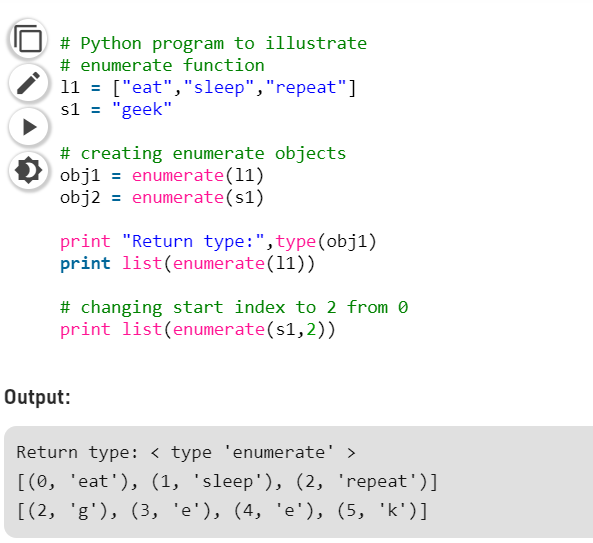


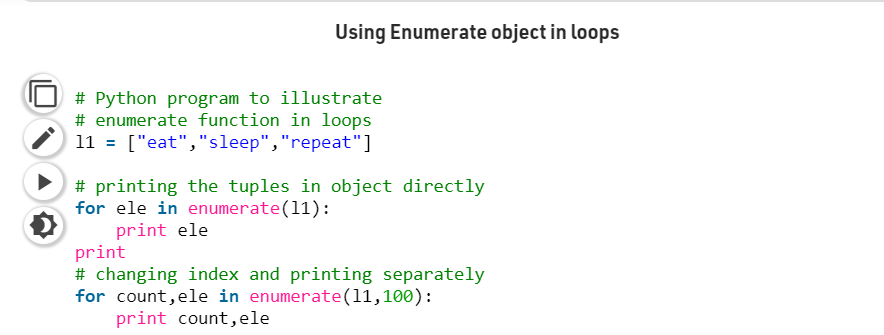


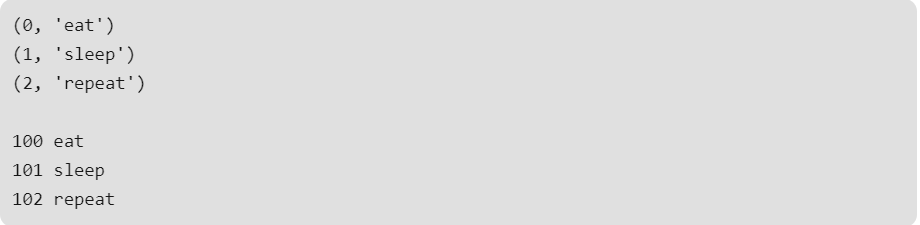
Enumerate() in Python

A lot of times when dealing with iterators, we also get a need to keep a count of iterations. Python eases the programmers’ task by providing a built-in function enumerate() for this task.  
Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.









Operator Overloading:

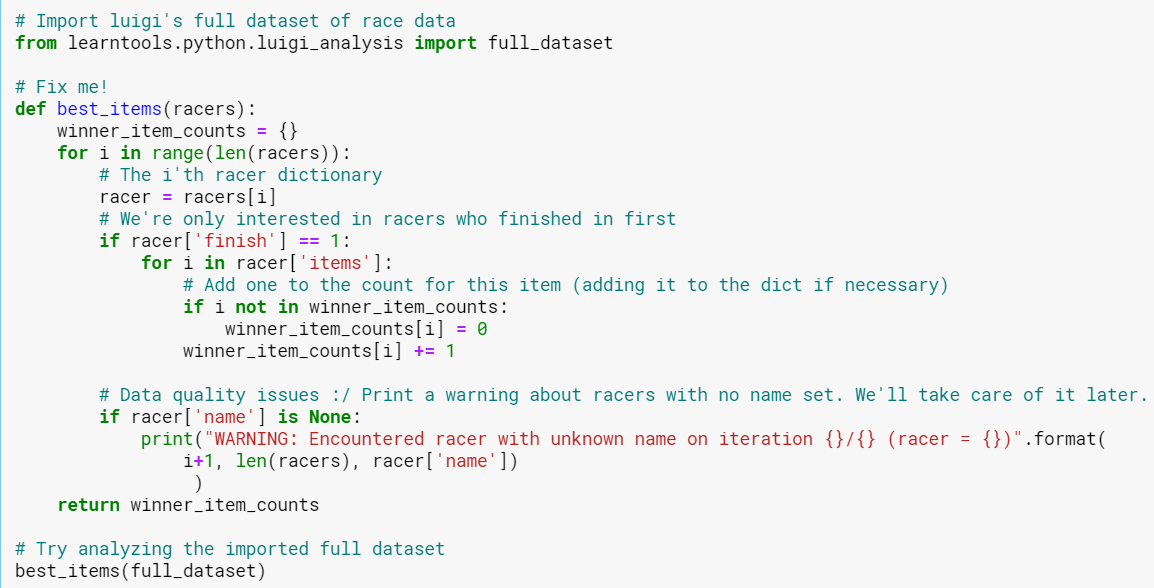
Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because ‘+’ operator is overloaded by int class and str class.

When Python programmers want to define how operators behave on their types, they do so by implementing methods with special names beginning and ending with 2 underscores such as \_\_lt\_\_, \_\_setattr\_\_, or \_\_contains\_\_. Generally, names that follow this double-underscore format have a special meaning to Python.

So, for example, the expression x in [1, 2, 3] is actually calling the list method \_\_contains\_\_ behind-the-scenes. It's equivalent to (the much uglier) [1, 2, 3].\_\_contains\_\_(x).

If you're curious to learn more, you can check out [Python's official documentation](https://docs.python.org/3.4/reference/datamodel.html#special-method-names), which describes many, many more of these special "underscores" methods.

Problems:





Solution: Luigi used the variable name i to represent each item in racer['items']. However, he also used i as the loop variable for the outer loop (for i in range(len(racers))). These i's are clobbering each other. This becomes a problem only if we encounter a racer with a finish of 1 and a name of None. If that happens, when we try to print the "WARNING" message, i refers to a string like "green shell", which python can't add to an integer, hence a TypeError.

This is similar to the issue we saw when we imported \* from math and numpy. They both contained variables called log, and the one we got when we tried to call it was the wrong one.

We can fix this by using different loop variables for the inner and outer loops. i wasn't a very good variable name for the inner loop anyways. for item in racer['items'] fixes the bug and is easier to read.

Variable shadowing bugs like this don't come up super often, but when they do they can take an infuriating amount of time to diagnose!

