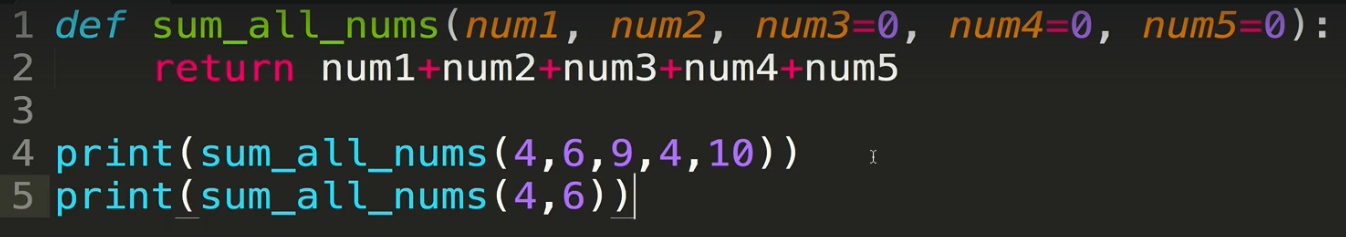
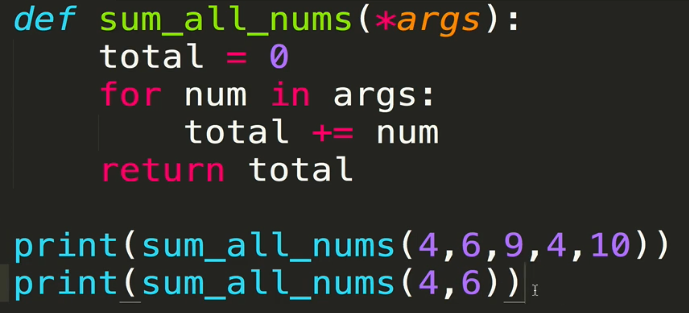
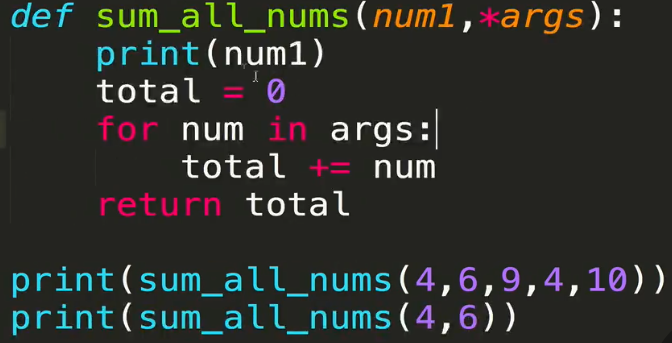
* Star args (\*args) are special operators that we can pass to functions
  + They gather the remaining positional arguments as a tuple that can be accessed by the function
  + Is a single parameter that we can call whatever we want
  + **\*args** is conventional, but you can call it anything you want so long as it begins with an asterisk (star)
  + Useful when you want to have a function that does not have a defined number of parameters
    - For example, maybe you want to multiply two numbers. Maybe another time you want to multiple four numbers, or ten numbers, or more
* Example: sum of numbers
  + In the example below, you must pass in the number of arguments that the function has parameters, unless you give the missing arguments default values. This is cumbersome and inflexible



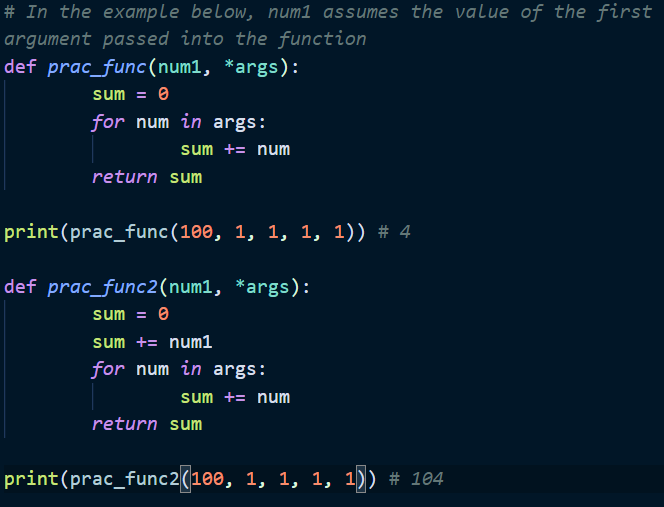
* + The alternative is to use \*args
    - Within the function, you refer to the parameters without the star. You only need the star in the function definition
    - The reworked function below will work for any number of parameters passed in



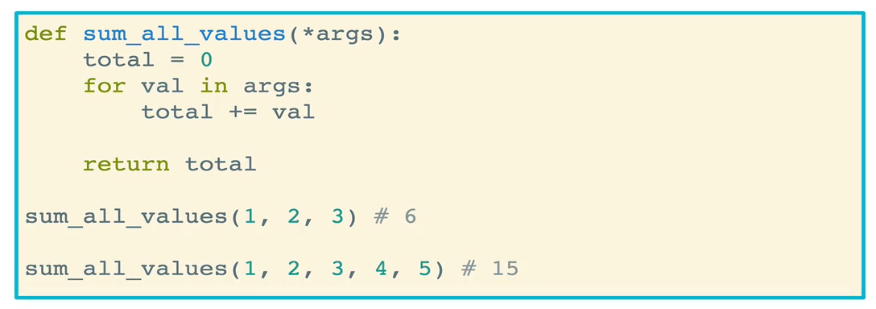
* + - Note that you can also have individual named parameters and have star arguments afterwards. Remember that the named parameters will be assigned in the order in which you provide them to the function, unless you use a keyword argument. In the example below, num1 would be assigned the value of 4.



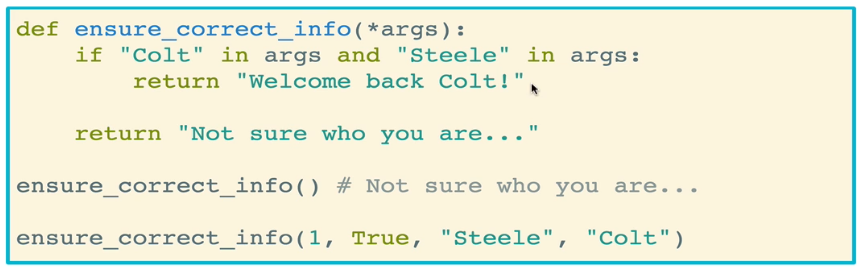
* + 1. Another illustrative example:



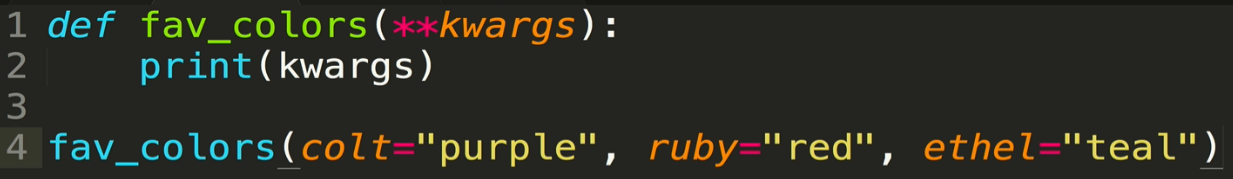
* Other examples
  + Summing

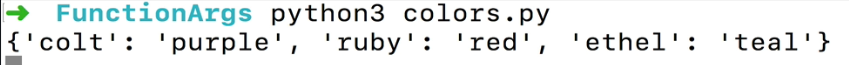


* + Ensuring correct information
    - Remember that the \*args is a tuple and you can iterate over them using conditionals

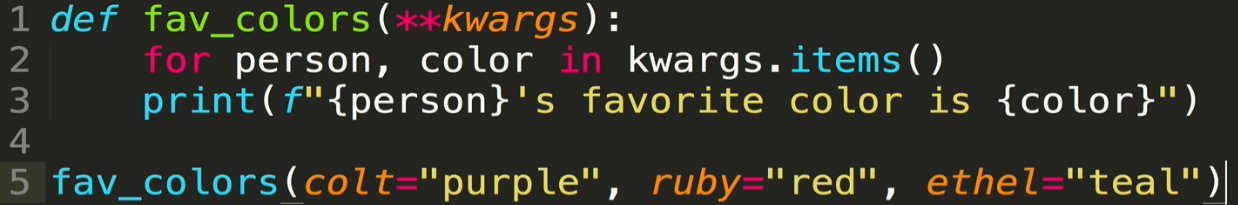


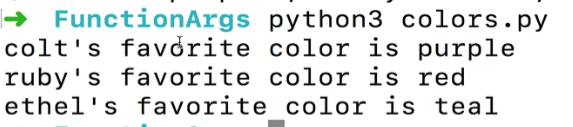
* **Keyword args**, or **\*\*kwargs** (pronounced QWARGS)
  + These are special operators that we can pass to functions
  + They gather the remaining **keyword arguments** that are passed into the function and store them together in a dictionary
    - Remember that keyword arguments are passed in to functions as *variable\_name* = *value*
    - All keyword arguments that are named variables will be assigned to those variables directly (see Functions Part I notes). The remaining keyword arguments that are not named variables will be assigned to the *kwargs* dictionary
  + You don’t have to call it \*\*kwargs; you can call it anything you want as long as you have the double asterisk (\*\*). But \*\*kwargs is standard
    - Note in the example below that the keyword names (e.g. colt, ruby, and ethel) are automatically converted into strings in kwargs



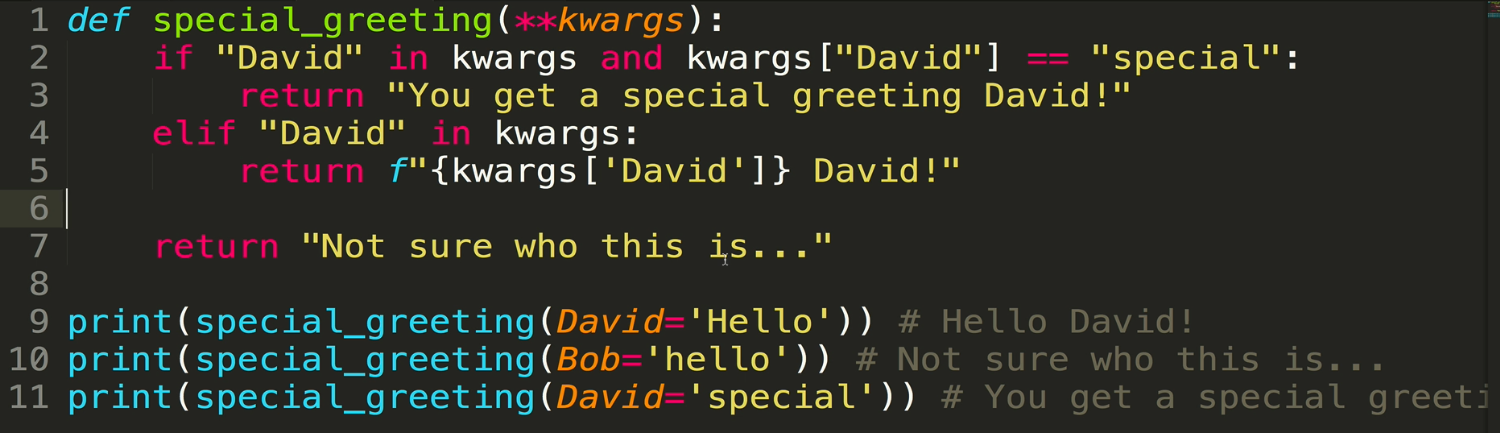


* + You can loop through the \*\*kwargs just like any other dictionary, and you can access the keys, values, and items just like any dictionary.

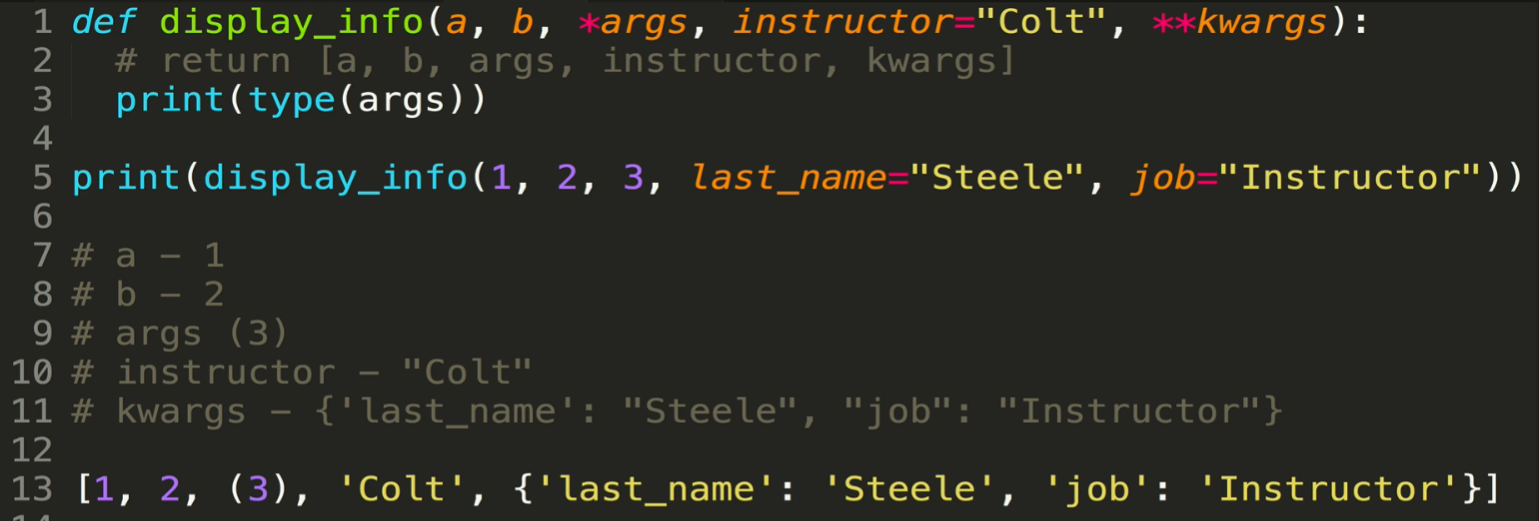




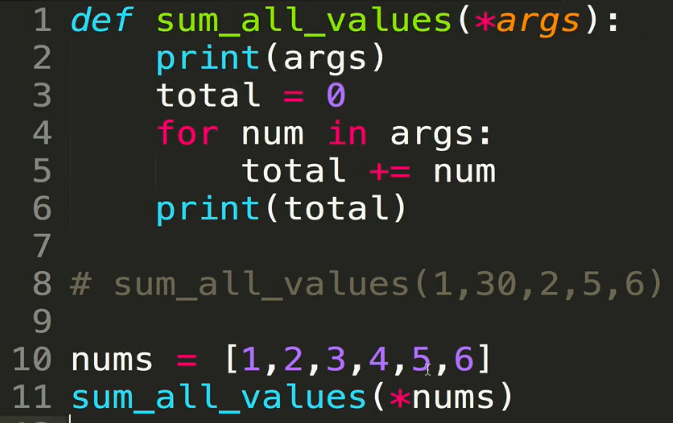
* + You can also use conditionals to check whether a key or a value is in kwargs. In the example below, a message is printed depending on whether David is a key in kwargs and further if the value for David is a particular string
    - Again remember that the keyword variable names will become strings the kwargs dictionary



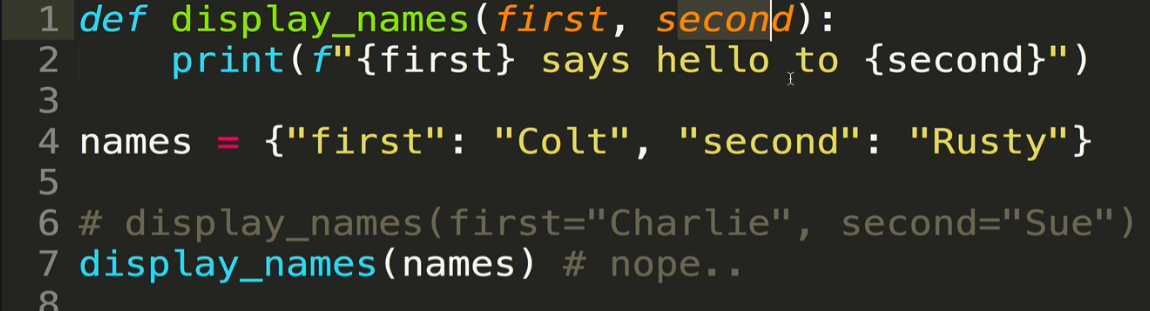
* Parameter ordering
  + There is an order that you have to follow for declaring parameters in a function
    1. Named parameters
    2. \*args
    3. Default parameters
    4. \*\*kwargs
  + You may only have one, two, or three of the above, but whatever subset you have, they have to be declared in that order



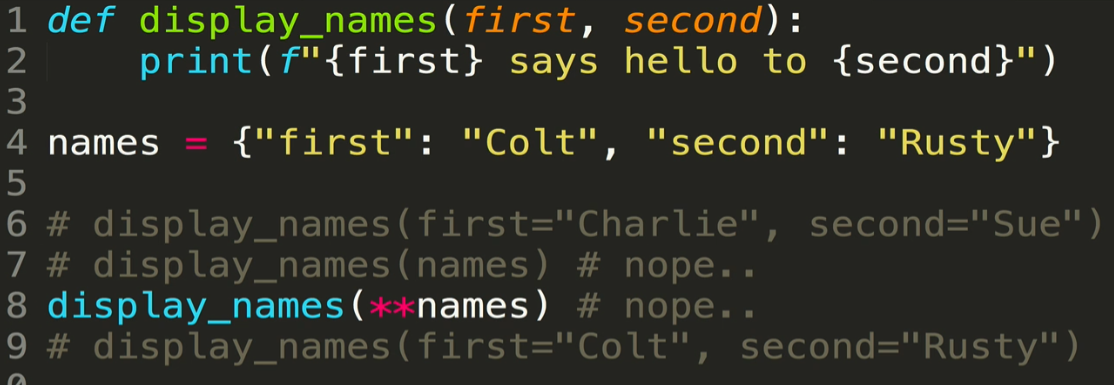
* The \* can also be used as an argument that is passed into the function (as opposed to a parameter in the function definition), where it is used to **unpack** values
  + This allows you to split up a list, tuple, or set of arguments and unpack them, to send it to a function that is not expecting multiple arguments
    - Remember that when you pass in a value (number, string, list, etc.) into a function as a \*args, it becomes a tuple
  + All you have to do is pass in the collection (list, etc.) with a \* in front of it! So long as the function is expecting a \*arg, it will unpack the collection and put each element into a tuple that can be used within the function
    - If you do not include the asterisk, then the collection will be thrown into a tuple with all of the elements together as one item in the tuple



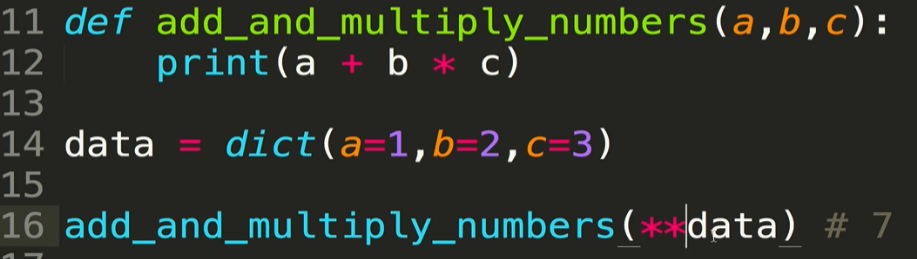
* \*\* can also be used in an argument passed into functions! It will let you **unpack dictionaries**, similarly to how \* will let you unpack lists or tuples
  + Sometimes you’ll have data that’s in a dictionary and you’ll want to use the individual
  + \*\* unpacks dictionaries into *keyword arguments*. In order to be referenced within the body of the function, keys in the dictionary MUST match the variable names in the function declaration
  + In the example below, when the dictionary *names* is passed into the function, it is assigned to the parameter *first*. The parameter *second* receives no argument assignment



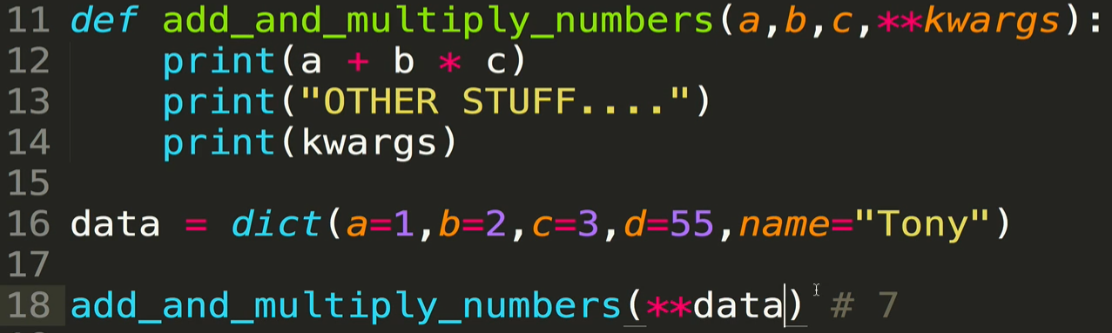
* + If you simply put two stars in front of the dictionary, the dictionary will unpack into separate keyword arguments.
    - Remember that the keys of the dictionaries must match the names of the parameters



* + Another example. The dictionary cannot be passed into this function directly. One must unpack it using \*\*



* + Final example: what happens if you combine dictionary unpacking with the \*\*kwargs argument in the function declaration?
    - The first three elements of the dictionary will be assigned to the first three parameters, while the rest of the dictionary will be placed into the *kwargs* dictionary



* + - You are not limited to just the arguments in the dictionary. You can continue to pass in keyword arguments after the \*\*data. Any keyword arguments coming after the names variable are “full” will be assigned to the *kwargs* dictionary

