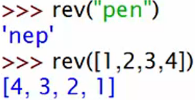
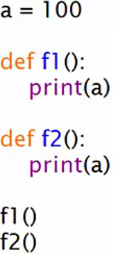
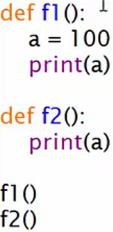
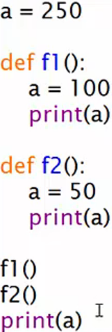
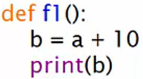
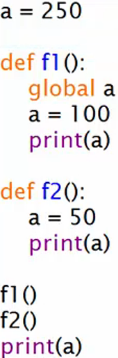
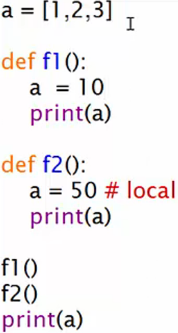
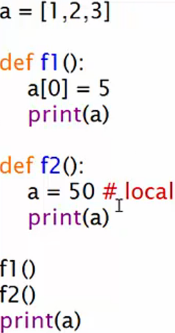
**Section Overview**

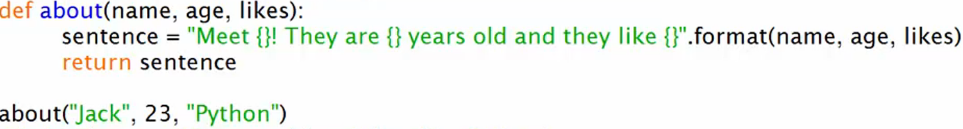
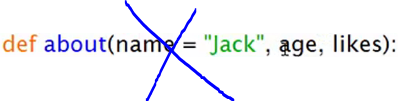
**What are functions?**  
\* **Function is a block of organized and reusable code that performs an action or gives some result**.  
**def** => **To define a function you use the `def` keyword**.  
**def name(parameters):**  
  
  
 **\* This function can reverse any Iterable.  
\* Functions can hide certain pieces of information from the outside world using Variable Scope.**

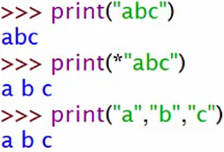
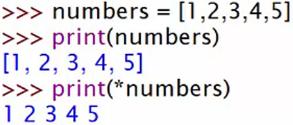
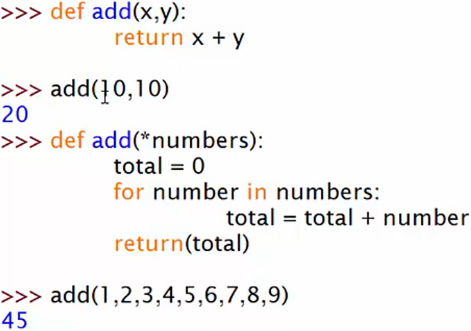
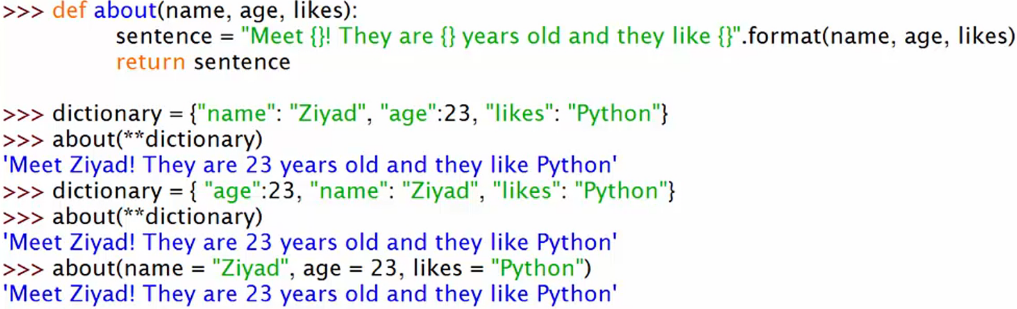
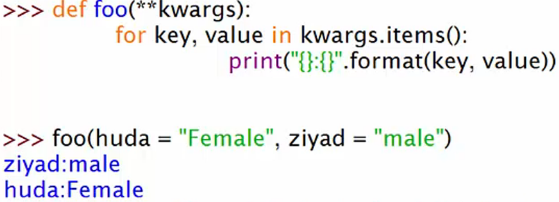
**Coding Exercise 10: Function Creation Challenge**

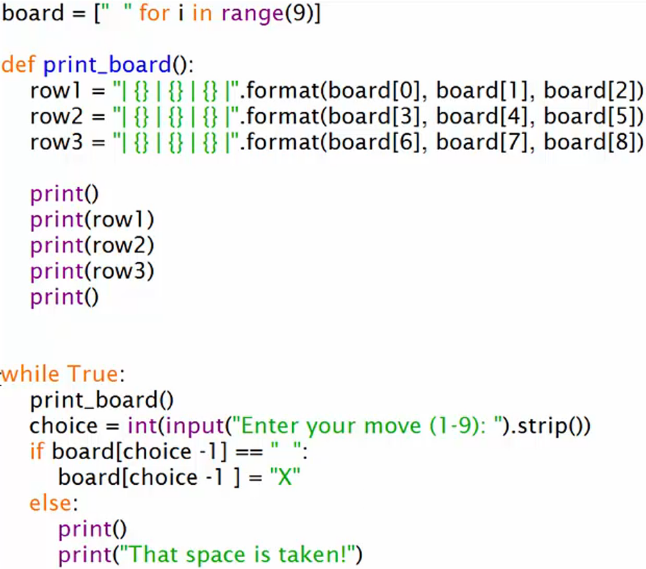
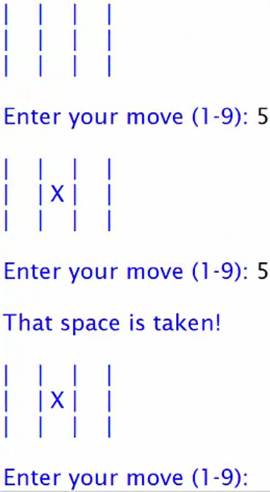
**Variable Scope - Part 1**  
**Variable Scope** => **allows you to keep certain parts of your program from interfering with others**.  
\* It’s especially applicable to Python functions.  
\* **There are only 2 types of Variable Scope in Python**.  
1) **Global => Global Variable, can be seen anywhere in the program.**  
2) **Local => Can only be seen within the specific local scope that it’s in.**  
\* **In Python, functions create local scopes but loops and if statements don’t**.  
\* **Global scope, visible everywhere**.  
   
\* **Only visible to f1()**.  
   
\* **Functions always create local scopes inside themselves**.  
  
  
  
  
  
  
   
\* **When you try to change a global variable from inside a local scope, Python stops that from happening by default and automatically creates a local variable with the same name**.  
\* **Local variables are destroyed when the function finished**.  
  
  
\* **The function f1() couldn’t find a variable called `a` inside its local scope, so it looked outside into the global scope and found a variable `a`.**

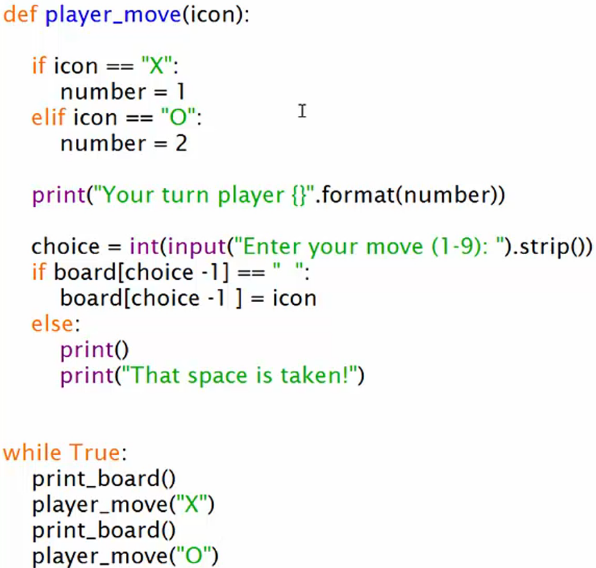
**Variable Scope - Part 2**  
**global keyword => Change a global variable in a local scope.  
global name** **=> You need to have 2 separate lines for it, you have to first type**  
\* **There’s a slightly special case of these rules when you use Lists or Dictionaries**.  
  
=> **The global value is protected as usual**.

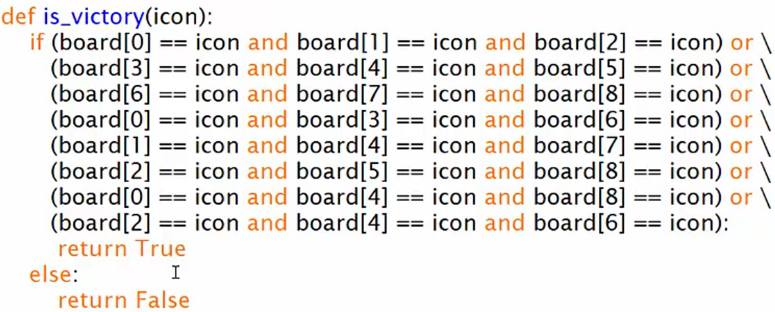
\* **However, we can change a piece if it, such as a[0] = 7, without using the global keyword**.  
   
  
\* **This works the same for a Dictionary**.

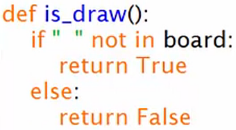
**Keyword Arguments and Default Parameters**  
**Function Parameters** => **What we use in the function definition ()**.  
**Positional Arguments** => **What we pass/give to the function when we call it**.  
Default Parameter =>  
  
=> **Parameters**: name, age, likes  
=> **Arguments**: “Jack”, 23, “Python”  
**Keyword Arguments** => **We can enter our arguments in any order we like, as long as we make it clear which parameter they associate with**.  
  
\* **What if we didn’t want to give a specific value but we still wanted the function to work?**  
**Default Parameters**  
  
\* **Default parameters MUST go at the end**  


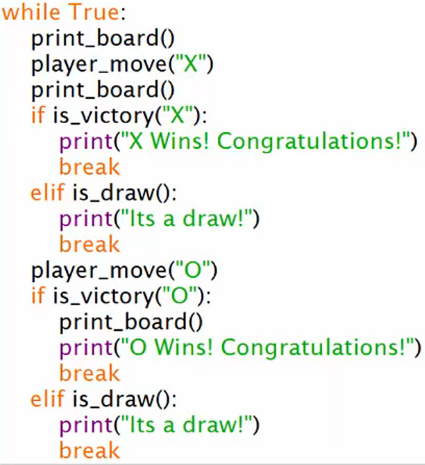
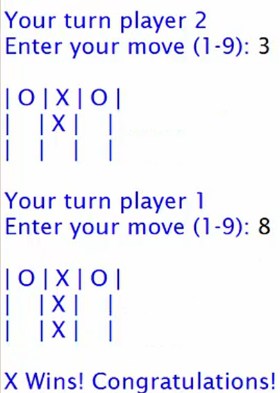
**Packing & Unpacking using \*args and \*\*kwargs**  
\* **Packing and Unpacking Arguments**.  
\* **Packing and Unpacking Keyword Arguments**.  
\* **Make much more flexible functions and using functions should be a lot easier**.  
**Unpacking** \* => **print(\*array) => We take each item from an Iterable datatype and each of those items becomes its own argument to the function.**  
**Packing** \* => **add(\*numbers) inside of a function, it’s going to pack whatever arguments that come in, into 1 Tuple.  
\*args** => **it’s a convention for positional arguments**  
 **=> So add(1,2,3,4,5,6,7,8,9) then basically creates numbers = (1,2,3,4,5,6,7,8,9) inside the function.**  
\* **Packing is very useful when you don’t know how many arguments you’re actually expecting**.  
**Unpacking \*\* Keyword Arguments => unpacks a dictionary into arguments**  
**Packing \*\* Keyword Arguments => packs arguments into a dictionary  
\*\*kwargs** => **it’s a convention for keyword arguments**\* The good thing about dictionaries is that we can type these in whatever order because it’s the key that becomes the parameter and the value becomes the argumemt, it doesn’t matter what order you put those in.  
\* You don’t have to type it by hand, you can just have some kind of an automatically generated Data Structure and then just pack them all in at once.  
\* **For this to be useful, you need to remember that the dictionary will very likely be generated automatically somewhere else in your program, so you won’t be typing it by hand like we did**.  
\* **for key, value in items**  
  
\* So if you’ve got a bunch of keyword arguments that you don’t know what they’re even going to be called yet.

**PROJECT 8 - Tic Tac Toe Game! - Part 1**  
\* **In Game Programming it’s called a Game Loop - the while True, so that the game keeps running**.  
 

**PROJECT 8 - Tic Tac Toe Game! - Part 2**  






**Section Review**