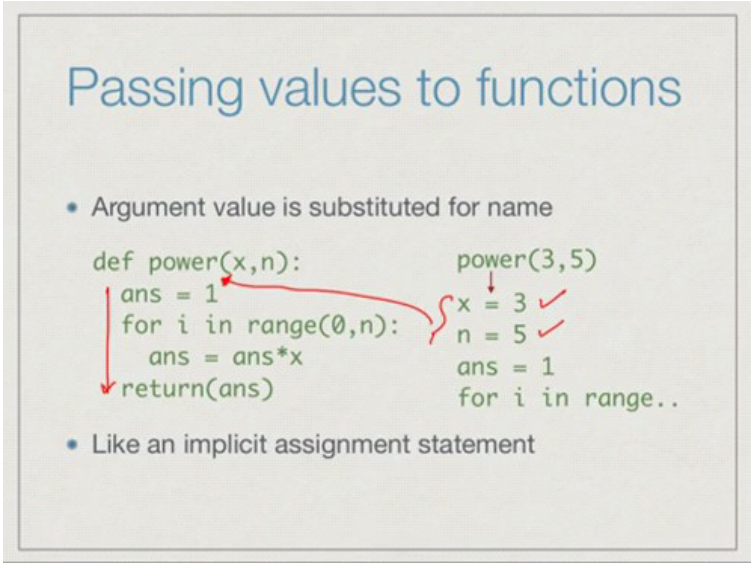
Programming Data Structures and Algorithms in Python
Prof. Madhavan Mukund
Department of Computer Science and Engineering
Chennai Mathematical Institute,Madras

Week - 04
Lecture - 06
Function Definitions

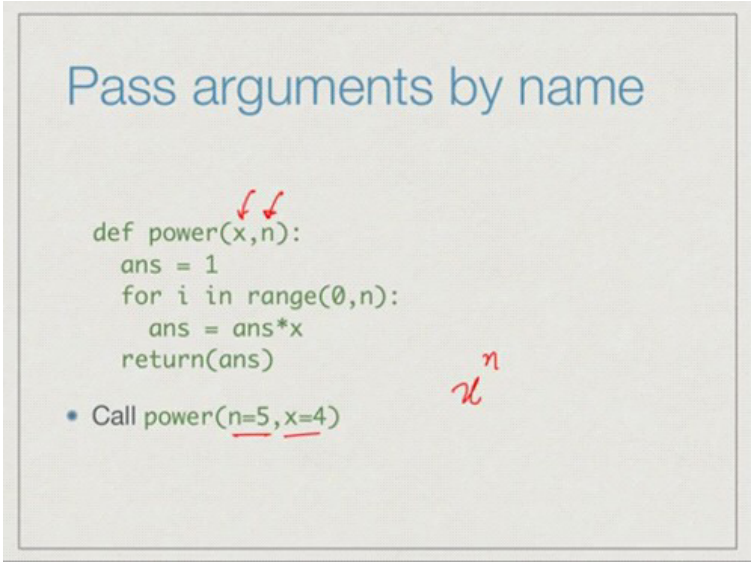
We have seen that we pass values to functions by substituting values for the argument set when defining the function.

(Refer Slide Time: 00:02)



And, this is effectively the same as having an implicit assignment. So, when we say power x n, and we call it values with 3 and 5, then we have this assignment x equal to 3 and n equal to 5. It is not really there, but it is as though, this code is executed by preceding this assignment there and of course, the advantage of calling it as the function is that, we do not have to specify x and n in the function definition; it comes with the call. So, for different values of x and n, we will execute the same code.

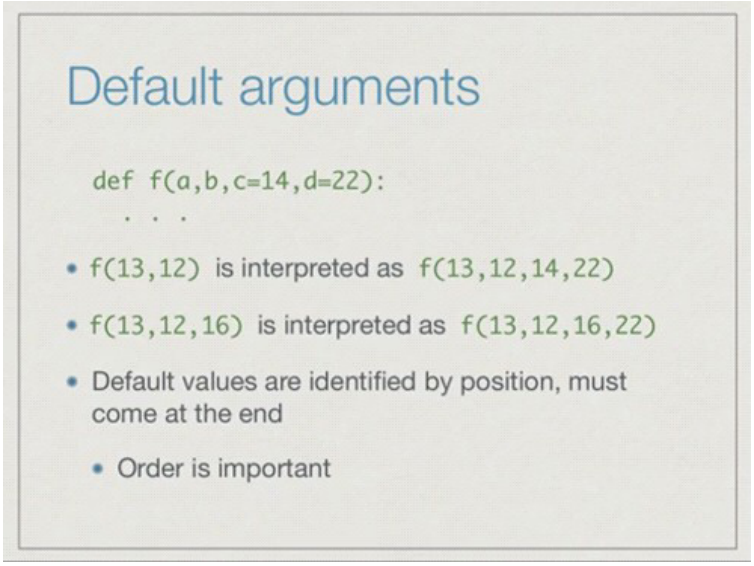
(Refer Slide Time: 00:02)



The first thing that python allows us to do flexibly, is to not go by the order; it is not that, the first is x, and the second is n; we can, if you do not remember the order, but we do know the values, the names assigned to them, we can actually call them by using the name of the argument.

So, we can even reverse the thing, and say, call power. And I know that, x is the bottom value I know it is x to the power n, but I do not remember whether x comes first, or n comes first. I can say, let us just play safe and say power of n equal to 5, x equal to 4 and this will correctly associate the value according to the name of the argument and not according to the position.

(Refer Slide Time: 01:24)



Another nice feature of python is that, it allows some arguments to be left out and implicitly have default values. Recall that, we had defined this type conversion function int of s, which will take a string and try to represent it as an integer, if s is a valid representation of an integer. So, we said that, if we give it the string "76", then, int would convert it to the number 76. If on the other hand, we gave it a string like "A5", since A is not a valid number, "A5" would actually generate an error.

Now, it turns out that, int is actually not a function of one argument, but two arguments; and the second argument is the base. So, we give it a string and convert it to a number in base b, and if we do not provide b, then, by default b has value 10. So, what is happening in the earlier int conversions is that, it is as though we are saying, int "76" with base 10, but since, we do not provide the 10, python has a mechanism to take the value that is not provided, and substitute to the default value 10.

Now, if we do provide it a value, then, for instance, we can even make sense of "A5". If you have base 16, if you have studied base 16 ever in school, you would know that, you have the digit zero to 9, but base 16 has numbers up to 15. So, the numbers beyond 9 are usually written using A, B, C, D, E, F. So, A corresponds to, what we would think of is the number 10 in base 10. So, if you write "A5" in base 16, then, this is the sixteenth position and this is the ones

position. So, we have 16 times 10, because the A is 10, plus 5. In numeric terms, this will return 165 correctly.

How does this work in python. This would be how internally, if you were to write a similar function, you would write it. So, you provide the arguments, and for the argument for which you want an optional default argument, you provide the value in the function definition. So, what this definition says is that, int takes 2 arguments s and b and b is assumed to be 10, and is hence, optional; if the person omits the second argument, then it will automatically take the value 10. Otherwise, it will take the value provided by the function call. The default value is provided in the function definition and if that parameter is omitted, then, the default value is used instead. But, one thing to remember is that, this default value is something that is supposed to be available when the function is defined. It cannot be something which is calculated, when the function is called.

So, we saw various functions like Quick sort and Merge sort and Binary search, where we were forced to pass along with the array the starting position and the ending position. Now, this is fine for the intermediate calls, but, when we want to actually sort a list, the first time we have to always remember to call it with zero, and the length of the list. So, it would be tempting to say that, we define the function as something which takes an initial array A as the first argument, and then, by default takes the left boundary to be zero, which is fine, and the right boundary to be the length of A.

But, the problem is that, this quantity, the length of A, depends on A itself. So, when the function is defined, there will be, or may not be a value for A and whatever value you have chosen for A, if there is one, that length will be taken as a default. It will not be dynamically computed each time we call Quicksort. So, this does not work, right. So, when you have default values, the default value has to be a static value, which can be determined when the definition is read for the first time, not when it is executed.

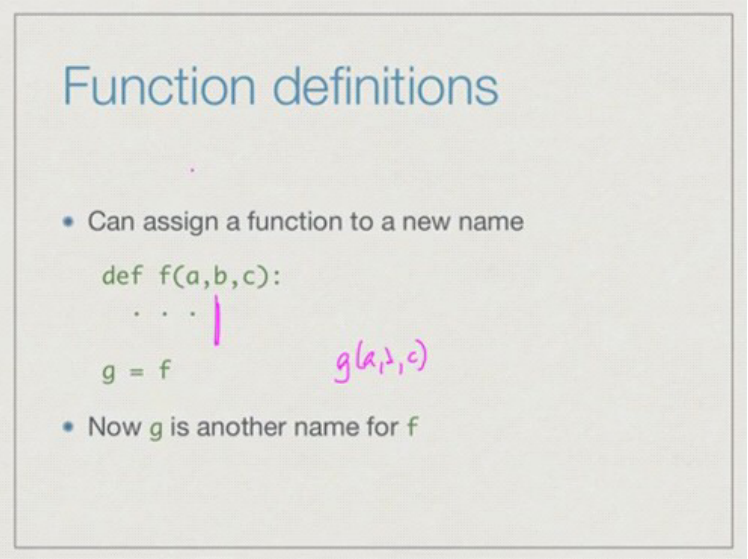
Here is a simple prototype. Suppose we have a function with 4 arguments a, b, c, d and we have, c has the default value 14, and d has a default value 22. Then, if you have a call with just 2 arguments, then, this will be associated with a and b, and so, this will be interpreted as f 13, 12, and for the missing argument c and d, you get the defaults 14 and 22. On the other hand, you

might provide 3 arguments, in which case, a becomes 13, b becomes 12 as before, and c becomes 16, but d is left unspecified; so, it pick up the default value.

This is interpreted as f of 13, 12, 16 and the default value 22. So, the thing to keep in mind is that, the default values are given by position. There is no way in this function to say that, 16 should be given for d, and I want the default value for c; you can only drop values by position from the end. So, if I have 2 default values, and if I want to only specify the second of them, it is not possible; I will have to redefine the function to reorder it.

Therefore, you must make sure that, when you use these default values, they come at the end, and they are identified by position. And do not mix it up, and do not confuse yourself by combining these things randomly. So, the order of the arguments is important.

(Refer Slide Time: 06:43)



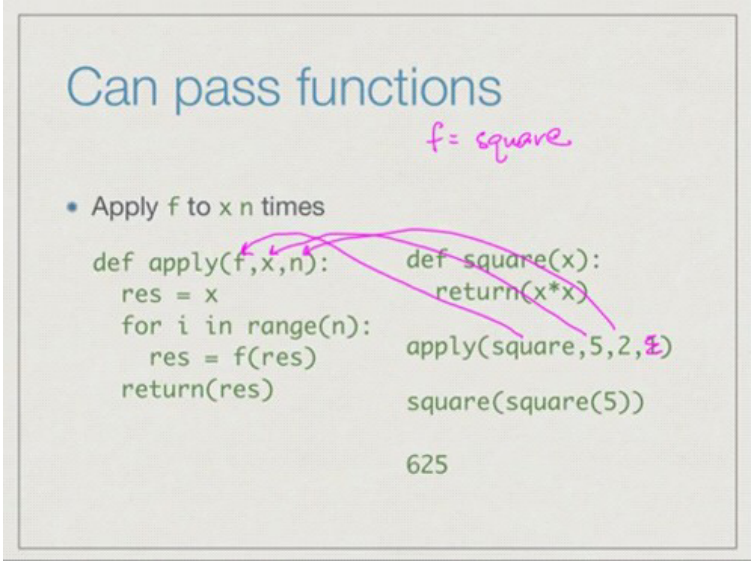
A function definition associates a function body with a name. It says, the name f will be interpreted as a function which takes some arguments and does something. In many ways, python interprets this like any other assignment of a value to a name. For instance, this value could be defined in different ways, multiple ways, in conditional ways. So, as you go along, a function can be redefined, or it can be defined in different ways depending on how the computation proceeds. Here is an example of a conditional definition. You have a condition; if it

is true, you define f one way; otherwise, you define f another way. So, depending on which of these conditions held when this definition was executed, later on the value of f will be different.

Now, this is not to say that, this is a desirable thing to do, because you might be confused as to what f is doing. But, there are situations where you might want to write f in one way, or another way, depending on how the computation is proceeding; and python does allow you to do this. Probably, at an introductory take to python, this is not very useful; but, this is useful to know that such a possibility exists and in particular, you can go on and redefine f as you go ahead.

Another thing you can do in python, which may seem a bit strange to you, is you can take an existing function, and map it to a new name. So, we can define a function f, which as we said, associates with the name f, the body of this function; at a later stage, we can say g equal to f. And what this means is now that, we can also use g of a, b, c and it will mean the same as f of a, b, c. So, if you use g in the function, it will use exactly the same function as a, its exactly like assigning one list to another, or one dictionary to another and so on. Now, why would you want to do this? So, one useful way in which you can do this, use this facility is to pass a function to another function.

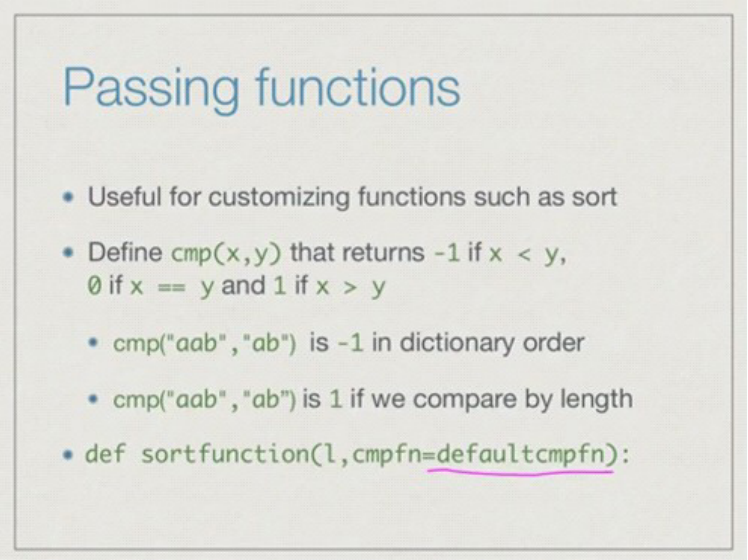
(Refer Slide Time: 08:40)



Suppose, we want to apply a given function f to its argument n times, then we can write a generic function like this called apply, which takes 3 arguments. The first is the function, the second is the argument, and the third is the number of times, the repetitions. So, we start with the value that you are provided, and as many times as you are asked to, you keep iterating function f. So, let us look at a concrete example.

Supposing, we have defined a function square of x, which just returns x times x; and now we can say, apply square to the value 5 twice. So, what this means is, apply square of 5, and then, square of that; so, do square twice. Therefore, you get 5 squared 25; 25 squared 625. So, what is happening here is that, square is being assigned to f, 5 is being assigned to x, and 2 is being assigned to n. This is exactly as we said like, before, like, saying f is equal to square. So, in this sense, being able to take a function name and assign it to another name is very useful, because, it allows us to pass functions from one place to another place, and execute that function inside the another function, without knowing in advance what that function is.

(Refer Slide Time: 10:08)



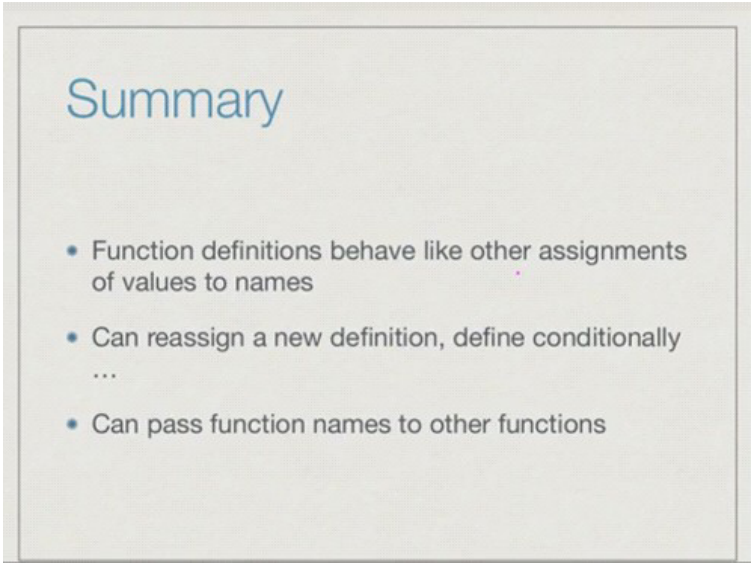
One practical use of this is to customize functions such as sort. Sometimes, we need to sort values based on different criteria. So, we might have an abstract compare function, which returns minus 1 if the first argument is smaller, zero if the 2 arguments are equal, and plus 1 if the first argument is bigger than the second. So, when comparing strings, we may have 2 different ways

of comparing strings in mind, and we might want to check the difference, when we sort by these 2 different ways.

We might have one sort in which we compare strings in dictionary order. So, string like aab will come before ab, because, the second position a is smaller than b. So, this will result in minus 1, because, the first argument is smaller than the second argument. If, on the other hand, we want to compare the strings by length, then, the same argument would give us plus 1, because, aab has length 3 and is longer than ab. So, we could write a sort function, which takes a list, and takes a second argument, which is, how to compare the entries in the list.

The sort function itself does not need to know what the elements in a list are; whenever it is given a list of arbitrary values, it is also told how to compare them. So, all it needs to do is, apply this function to 2 values, and check if their answer is minus 1, zero, or plus 1 and interpret it as less than, equal to or greater than. Then, if you want, you can combine it with the earlier feature, which is, you can give it a default function. If you do not specify a sort function, there might be an implicit function that the sort function uses; otherwise it will use the comparison function that you provide.

(Refer Slide Time: 11:50)



To summarize, function definitions behave just like other assignments of values to names. You can reassign a new definition to a function. You can define it conditionally and so on. Crucially, you can use one function and make it point, name point to another function, and this is implicitly used when we pass functions to other functions and in situations like sorting, you can make your sorting more flexible by passing your comparison function which is appropriate to the values we will sort.