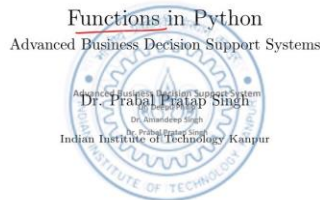


**Advanced Business Decision Support Systems**  
**Professor Deepu Philip**  
**Department of Industrial Engineering and Management Engineering**  
**Indian Institute of Technology, Kanpur**  
**Professor Amandeep Singh**  
**Imagineering Laboratory**  
**Dr. Prabal Pratap Singh**  
**Indian Institute of Technology, Kanpur**  
**Lecture 32**  
**Functions in Python**

Hello everyone, I welcome you all to the lecture series on Advanced Business Decision Support Systems. I am Doctor Prabal from IIT Kanpur and we are discussing the fundamentals of Python programming language.

*Data Types  
Control Flow Statements  
- for  
- while  
- if-elif-else*



So, till now, we have discussed all our major data types. We have covered the control flow statements like for Loop, While and If, Elif, Else statements. The next major concept is Functions. Why do we need it? Why developers introduced it in Python? How can we use this new concept with all other previous concepts? So, let us see the agenda.


Agenda

- Functions *→ Define a function*  
*→ Call a function*
- Function Arguments *→ Argument and Return*
- args & kwargs
- Global and Local Scope



So, here we are trying to see, how to declare a function, how to use it, how to call a function and then, each function can have its own arguments. So, what are the different parameters and their types and when to use which one. Again next, we will try to see, what are this very famous phrases like args and kwargs. Finally, we will see the global and local scope of a variable using Functions.

Functions

- \* Mathematical functions  $\rightarrow$  
- \* Functions in Python act similar to their counterpart in Maths.
- \* Useful when we need to perform similar computation at different instances in a source code.
- \* Syntax of function  $\rightarrow$ 
  - def - keyword
  - name of function  $\rightarrow$  `def my_func(x, y):` - (Underscore) to stick
  - arguments  $\rightarrow$  `def my_func(x, y):`
  - DOCSTRING  $\rightarrow$  `""" """`
  - statements
  - return
- Nested Function  $\rightarrow$ 

```
def my_func():
    def my_func():
        def my_func():
            statements
```

Dr. Prabhat Prasad Singh | Functions in Python | 3/4

So, functions are nothing but very similar to mathematical functions. So, in your earlier school days, we have all seen the basic definition of function as something like this. This is a kind of black box, where some input comes and it releases some output, this can be a function. So, we define function in mathematics as  $y = f(x)$ . So, the functionality remains the same, but since we are using a programming language.

So, we need some different notations and syntax to implement this functionality in python. So, functions in python act similar to their counterpart in maths and they are useful in the code. Now, the syntax of functions includes these major things like, a def keyword def, this is a keyword. By writing this, interpreter will know that the next few statements will come under functions. Then, we need a name of a function.

Now, the naming convention in python for this function is use lower case letters and use underscore to stick in different words. So, if you are trying to make a function with the name my function, so you can write my\_func. So, what it means is, you should not have a space character in between two words while defining the name of the function. Next is the argument of function. So, any function let us say, def my\_func should have some arguments that is, the input here which we try to give to our function.

Now, the next thing is these colons, these colons will tell that the basic definition of function starts ends here and the next line contains the statements of function. So, after this line, we can use three quotes to create a string, which is known as DOCSTRING. So, the role of this DOCSTRING is to tell the user what this function can do.

So, this gets populated by the programmer who is writing this function and whenever a user try to use this function in its own code, so it can use the help function and see what does the DOCSTRING contains and usually a good programmer will populate this


DOCSTRING as beautifully as he or she can be, so that all the major information to run this function successfully is provided. Now, these you can see here that, this is the indentation level, so anything after this first line will get indented to its second indentation level right.

So, these three lines are indented to the same level and everything like other statements of this function will have the same indentation level until and unless, we need to get out of this function. So, we can use a return statement to tell the interpreter that the function ends here and now the next line of code will again start from this indentation level right and we can also have nested functions where function like def my func can contain a different function, def simple and you can see that, indentation level keeps on increasing. Now, everything for this function will start from here, so this is how we can declare nested functions.

Function Arguments

*[Task: Auto Generate Username for newly admitted students in an academic institution]*

- \* Positional Arguments
- \* Keyword Arguments
- \* Default Argument
- \* Things that we can return from a function → return
- \* args and \*\*kwargs



Advanced Business Decision Support System  
Dr. Pradyumn Singh  
Dr. Prabal Prasad Singh

So, let us see few more things about the function arguments, so here we will today try to create a simple function and we will keep on enhancing this function by doing modifications. So, our task is to auto-generate usernames for newly admitted students in an academic institution.

So, whatever examples of function I show you, we will try to see how to complete this task, this way, the implementation of functions will get clearer. So, the function arguments can have positional arguments and we will see, what are these positional arguments and keyword arguments. We can also initialize our functions with some default arguments so that, if user is not giving any parameter while calling the function, then these default values can be used.

Then, we will also see the things that we can return from a typical function and here we will use the return statement. The third part is, we will see the args and kwargs keyword arguments in our function.

So, let us try to perform this task in our VS code. So, we will first try to create a new environment by running our command `python -m venv`. Let us write `func` and so we can see that, VS code is now using our newly created environment and here is the directory and we can start writing our code. So, let us first write a very basic function that can just report some string, so we will use the `def` keyword and let us type the name as `authentication_success`. So, this is the name of a function and you can see that, the color has been changed by the VS code editor and here we are not providing any arguments to this function.

So, write a colon and press enter. Now, this is an indentation indented block of code, so the cursor automatically comes after four space. Now, the first task of initializing any function is to create a doc string. You can skip this part but it is very beneficial for anybody else who will read your code afterwards, it will help him or her to understand what this function does. So, let us write this doc string by filling the details like this function will inform the user that, authentication is successful.

So, we can just write `print authentication is successful` and we can write a blank return statement here. So, we can also skip writing this return statement and the interpreter will understand that the function has ended, if we again start from here like, `print out of function`. So, even if we remove this statement like, I have commented it, so even in this way, this block of code will run smoothly. Now, let us again bring this back and see how we can call a function. To call a function, we need to just use its name `authentication success` and we can run this and this after calling the function, the function will print this statement `return` it printed that authentication is successful.

Now, you may ask, what is the use of all of this code. So, what happens is, let us say, we are just having 17 lines of code, but what if we are having 1700 lines of code and in at different points, in those huge lines of code, we need to write this print statement multiple times like, whenever user gets authenticated, we need to show this at the screen that the authentication is successful.

So, we need to write this let us say, we are authenticating 50 users, so we need to write this 50 times this whole statement but now by making this into a function, we can write we can just call this function and it will do all those the statements that are within this function. So, this way, we are trying to keep the code base as modular as possible. Now, we can do one simple variation by also adding the username of the function.

So, let us write the parameter as `user_name` and we can create an f string here by writing `f` and we can change, modify our string to say, authentication is successful for `user_name` user. So, now writing just this without any argument, this call will not work. So, we need to amend this and we need to write `user name` or we can just write `user name` as `aman`. So, if we run this again, this function definition and then call this function, then

this function is now telling that, authentication is successful for aman. Now, if we just changed to a different user let us say, abhishek, so the string gets changed successful for abhishek.

So, this way, we can create a very basic function and this function call can also be done by mentioning the name of the argument as username and writing `th equal`. So, now we are telling the interpreter that, we are providing this information for this argument, so we can see that, it works in a similar way. So, now let us move further and we can see one more modification in this by writing a default argument, so here we are just providing the username argument.

Now, what if we just write a general argument as user? So, if we don't provide anything in this argument here while doing the function call, then this default value will get printed. So, let us see this variation as well.

So, if we run this without argument and we need to run this function again to update this definition and now this will work. So, authentication is successful for users in capital and user. So, this is our string. Now, these are the default arguments and we can provide as many arguments to a function as possible. Now, let us move back to our task creating.

An automatic username generator, so we will again create a new function name `def make_username`. So, this is the name of our function and it will take `first _ name` and `last_name`. So, let us now provide the information about, what this function will do. So, it will take two arguments and generate a username using the first two characters of the `first _ name` and complete last name.

So, let us say, this is the policy of a particular institute that it will use the first two characters of the first name and the complete last name to create a new username for a newly admitted student.

The username will distinguish the first and last name using a `.`. So, this is what we need to accomplish by creating this function `make username`. So, we can do all these things in just single line. So, we can write `user_name` is equal to, we will take the first name and now you can see that, first name will be a string which gets provided by the user. So, since it is a sequential data type, so we can use indexing and we will only need the first two characters.

So, we will need the first two indexes which are 0 and 1. Now, we know that the last value during slicing gets excluded. So, we will write 2 and we can write 0 here but it is known to interpreter that, if there is nothing, then it will start from its very first value. Now, we can add this string. These first two characters with a `.` operator because the task

says that, the . operator is between the first and the last name and now again, we will add the last name.

So, last name is used completely. Now, this is the username which the authorities want in that institute right. So, we can print this. Let us run this function and we can call this function with some first name and last name by using the function name as make\_user\_name, first name = Dharmendra, last name = Gupta. So, what it should do is, it will take these first two characters dh, then it will add a . operator in between and then write this complete Gupta.

So, let us run this line. So, now it is showing that, username generated is Dh.Gupta but you can see here that, these are not in a same kind of cases, so we can use our string methods to write lower so that, all the string characters get into lower case. Similar thing, we can use with last name by writing lower method.

Now, again run this function and run this line of code. Now, the username is dh.Gupta right. So, this way, we can create our own function. Now, let us see one more thing and let us say that, the authorities also want to generate the email for this username. So, we can write user\_email and it will only add a particular domain to the generated username. So, we can write user\_name + string at some email.ac.in, since it is an academic substitute. So, now we can see one more property here for functions is the return statements.

Till now, we are not returning anything from the functions, but now we can return. So, let us just return the username from the function and we can store this into a new variable or we can just print it. So, let us run this and try to run this. So, now you can see that, this function has first used this print statement to write this string username generated and the username is dh.Gupta.

But again, the interpreter has printed this dh.Gupta in strings. Why did this happen? Because we have included a new statement named return. So, this function call first printed this and then returned this to the interpreter. Now, we can store this variable into some other variable.

So, let us write this as user\_name\_return and now run this. So, now it is only printing this for only one single time and we can print this user\_name\_return. So, this variable should be storing the username generated by this function. So, let us run this and you can see that, it contains the same value right. So, this way, we can return various kinds of data types from a function after our own computation.

Now, if we can return one variable, then we can also return number of different kinds of variables and different data types that can be stored in a variable. So, let us just write username comma user\_email. Now, we are writing two things without any other

punctuation marks. So, it means that, when this function will return something, it will return it as a tuple. So, let us again modify this and we also learned in our last lecture that, we can use tuple and packing to assign different number of variables in a single line.

So, let us write `user_email_return`. Now, this variable should store this `user_email` value. So, let us again run this and call this function here and print the new value as well. So, `username` contains the same thing and `user_email` now contains a valid email id `dh . gupta at email.ac.in` right. Now, let us move to the other concept of `args` and `kwargs`.

So, for this let us create a very basic calculator, which can calculate any number of data and give the output as the addition of all those things. So, aggregate calculator. Now, if the programmer knows that, the user wants to add only two numbers, then it can provide two arguments here. If the programmer knows that, we need ten numbers, then it can provide ten arguments here. But what if the programmer doesn't know what are the number of elements that a user wants to add? So, in those scenarios, it is very difficult to create a function that is of dynamic nature.

So, to compensate for these situations, developers in python introduced this `*args` Now, you can write anything instead of `args`, but you should use this operator star here. So, what this will do is, it will use the tuple unpacking and unpack all the input items into this argument. So, now here we are using arguments to create a dynamic function with unknown number of arguments. So, we can simply type the return statement and we can use the `sum` function available in standard library of python and we can use `args`. You can run this definition and let us call this Function.

So, let us just call it with three values 23, 45, 87 and it should give the sum of these values and we can see here that, it is returning 155 as the sum. Now, we can again use this dynamic function without changing any definition in the function definition, we can just add let us say, two more arguments. So, the fourth is 78, 56 and it should again work for all these arguments and we can see here that, the summation has changed to 289. So, this is a very good example of how to create dynamic functions.

Now, you must be wondering, what if I need to provide dictionary kind of things like named variables? So, for that, we can use keyword arguments that is, `kwargs` and you can write anything instead of this `kwargs` here but you have to use these two the asterisk operators twice.

So, let us now just print these quarks and how they store things. So, we need to again run this new definition and provide some keyword arguments as well. So, let us say, we only need to add these two things 43 and 32, but the first name is a new argument for this

function which is a keyword argument. So, let us write here as Aman and last \_ name as Singh.

So, this should now also print the keyword arguments. So, the first print statement is writing Aman and last name as Singh and it is writing it in a dictionary right and this return statement is providing the sum of these two values 43 and 32 and we can also return all of these things without printing this by again modifying this. This is also an example to how to return dictionaries from a function. So, we can just open a dictionary by a curly brace and let us also close it.

Now, we need to first provide a index keyword, the key for this first dictionary element. So, this is the sum and we can provide the other new key to the keyword arguments as the name and we can also merge the first name and the last name or anything here also but for now we are just printing it.

So, or we can use our operators we learned earlier to merge the all the keyword arguments in a single dictionary. So, let us use these double asterisk signs here as well and write key. So, what it will do is, if we provide five different keyword arguments, then it will create a new dictionary under this key name.

Let us run this as well and let us try to call this function again. So, now you can see that, it is printing nothing, but it is returning this whole dictionary and under the name key, we are having another dictionary with all the keyword arguments and we provided like first name and last name right.

So, let us now move to our last concept of local and global scope. So, create a new file name in python. So, what happens is, when we try to define different variables, then they have some kind of restrictions about, where can we use those variables. So, like we have seen here in Functions, there are some variables that are inside these Functions like, this username and user \_ email and some functions like here, this user \_ name \_ ret. So, these variables have different scopes like, we should understand whether we can use this variable here or not or we can use this variable inside a function or not.

So, this is under the concept of scopes in python. So, let us try to understand this. So, any variable that we define in a file which is out of any other functions or any other block of code, is by default under global scope. So, let us say, first \_ num as 12. So, when we define this the interpreter automatically registered it as a global variable right. But when we use a function, so what if we try to write the same number first \_ num? Now, this variable is available here in a global scope and this is a local scope of this function show underscore num.

This function will show the value of a variable. So, let us run this function as well and see what this will print. So, I am calling this, it is giving the output as 12. So, it is using this



global variable and printing inside a local scope of this. But if we modify this function a little bit and again initialize a variable with the same name of first num and try to write it as 32. So, what we did here is, we introduced a new variable inside this local scope of this function, which is until this and initialized it with a different value.

Now, the print function should print this value 32 right. But the question arises is, whether this first num the global variable changed or not. So, to understand this, we should provide a new print statement here and check the value of this global variable. So, you can see here that, this variable first num is still reporting the value as 12. However, this first num under the function definition is reporting the value as 32.

So, even with the same name of variable, their scope is different. So, this is a very special property and if programmers do not use this cautiously, then there could be some errors, which are hard to debug. So, let us say, in as our next task, we try to modify the value of this global variable by writing statements under this function. So, to do that, we need to use a keyword global and we will write first \_ num.

Now, what happened is, when the interpreter will reach to this line, it will understand that the programmer wants to assess the global variable named first num inside this local scope and now whatever happens to this first num container, it will keep on doing all the things to this global variable also.

So, let us now again run this function and see what this reports. So, this function is working fine, it is showing the value of first num is 32 which we have modified here but what is the value of this global scope first num. So, now this time, this is also 32. So, this global keyword has overwritten the original value of 12 to 32 here. So, let us comment this. Now, to understand nested functions and scopes together, we will try to take our old definition of this function from here and let us copy this definition into the scopes file.

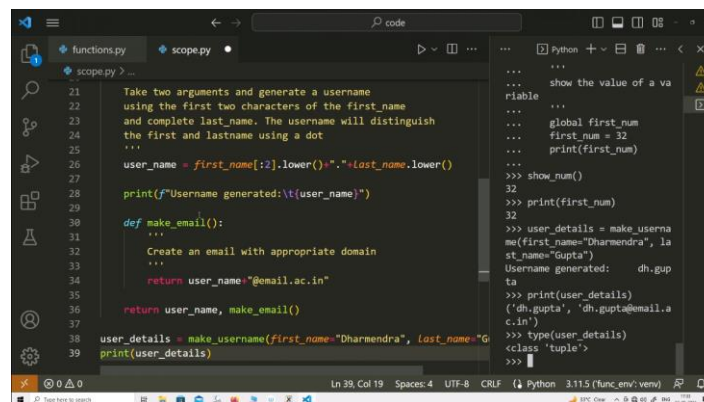
Now, to create a nested function, we need to define a new function inside a function. So, let us say, that we were creating this user name and its email in this task and earlier, we simply defined that, user email is this but what if this is a very different and large computation and we need to seclude it from the original function. So, we can create a new function inside this make user name function and write make email and we can write the argument for this as let us say, we don't provide any argument here and we can write the doc string as create an email with appropriate domain.

So, now we have a function with the name make user name inside, which we have a function make email and we are writing this user name here but we need to understand that the scope of this user name is taking from the scope of this user name defined in the make \_ user name function. So, for a nested function, all the variables that are defined in

a function above it are available, so if we just run this function and since, we don't have this user\_email used anywhere, we can write here return.

Now, we will return this thing and we can directly call this make\_email function here. So, now what is happening is, we will first take the first\_name and last\_name from the user, then this statement will create a user name by using the first two characters and the complete last name and include a . operator here, then it will print this line for the user name and then, this internal nested function will take the user name from here and try to add this domain to this user name and this return statement is for the first function and using this return statement, we are returning two different entities, the original user name created by the first function and the calling this nested function on the return statement itself.

So, we can catch the return statements by providing a new variable named user\_details and let us call the original function make\_user\_name and we can write a same name as the main last name is, so let us run this and it is showing that, there are no errors and this print statement worked right. Now, we can print our user\_details variable as well, so this is interesting because now it is reporting that, user name is dh.Gupta and make\_email called itself from the nested function and returned an email id which is this and this if you try to check the type of user\_detail, then this should be tuple because we are not using any punctuation marks and we are returning more than one item from the function, so it is automatically gets converted into a tuple right. So, this way, the different scopes of functions can use the same name of variables and provide an output as per the need of the programmer.



```
functions.py | scope.py | Python 3.11.5 (func.env: venv)
21 Take two arguments and generate a username
22 using the first two characters of the first_name
23 and complete last_name. The username will distinguish
24 the first and lastname using a dot
25 ...
26 user_name = first_name[:2].lower()+"."+last_name.lower()
27
28 print("Username generated:\t(user_name)")
29
30 def make_email():
31     ...
32     Create an email with appropriate domain
33     ...
34     return user_name+"@email.ac.in"
35
36 return user_name, make_email()
37
38 user_details = make_user_name(first_name="Dharmendra", last_name="G")
39 print(user_details)

... show the value of a va
... riable
...
... global first_num
... first_num = 32
... print(first_num)
...
>>> show_num()
32
>>> print(first_num)
32
>>> user_details = make_userna
me(first_name="Dharmendra", la
st_name="Gupta")
Username generated: dh.gup
ta
>>> print(user_details)
('dh.gupta', 'dh.gupta@email.a
c.in')
>>> type(user_details)
<class 'tuple'>
>>>
```

### Final Python Program

So, we can stop here and we will try to provide some practice assignments on these different kinds of data types control flow statements and the local and global scopes of functions and next we will try to see the Scientific Computing in python and other aspects to create our final decision support system. Thank you.