GeeksforGeeks
A computer science portal for geeks

Custom Search

Suggest a Topic
Login
Write an Article

*args and **kwargs in Python

***args**

The special syntax **args* in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-keyworded, variable-length argument list.

- The syntax is to use the symbol * to take in a variable number of arguments; by convention, it is often used with the word args.
- What **args* allows you to do is take in more arguments than the number of formal arguments that you previously defined. With **args*, any number of extra arguments can be tacked on to your current formal parameters (including zero extra arguments).
- For example : we want to make a multiply function that takes any number of arguments and able to multiply them all together. It can be done using *args.
- Using the *, the variable that we associate with the * becomes an iterable meaning you can do things like iterate over it, run some higher order functions such as map and filter, etc.
- **Example for usage of *arg:**

```python
# Python program to illustrate
# *args for variable number of arguments
def myFun(*argv):
    for arg in argv:
        print (arg)

myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')
```

Run on IDE Copy Code

Output:

```
Hello
Welcome
to
GeeksforGeeks
```

```python
# Python program to illustrate
# *args with first extra argument
def myFun(arg1, *argv):
    print ("First argument :", arg1)
    for arg in argv:
        print("Next argument through *argv :", arg)

myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')
```

Run on IDE Copy Code

Output:

```
First argument : Hello
Next argument through *argv : Welcome
Next argument through *argv : to
Next argument through *argv : GeeksforGeeks
```

**kwargs

The special syntax ***kwargs* in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name *kwargs* with the double star. The reason is because the double star allows us to pass through keyword arguments (and any number of them).

- A keyword argument is where you provide a name to the variable as you pass it into the function.
- One can think of the *kwargs* as being a dictionary that maps each keyword to the value that we pass alongside it. That is why when we iterate over the *kwargs* there doesn't seem to be any order in which they were printed out.
- **Example for usage of **kwargs:**

```
# Python program to illustrate
# *kargs for variable number of keyword arguments

def myFun(**kwargs):
    for key, value in kwargs.items():
        print ("%s == %s" %(key, value))

# Driver code
myFun(first ='Geeks', mid ='for', last='Geeks')
```

Run on IDE Copy Code

Output:

```
last == Geeks
mid == for
first == Geeks
```

```
# Python program to illustrate  **kargs for
# variable number of keyword arguments with
# one extra argument.
```

```
def myFun(arg1, **kwargs):
    for key, value in kwargs.items():
        print ("%s == %s" %(key, value))

# Driver code
myFun("Hi", first ='Geeks', mid ='for', last='Geeks')
```

Run on IDE Copy Code

Output:

```
last == Geeks
mid == for
first == Geeks
```

Using *args and **kwargs to call a function

Examples:

```
def myFun(arg1, arg2, arg3):
    print("arg1:", arg1)
    print("arg2:", arg2)
    print("arg3:", arg3)

# Now we can use *args or **kwargs to
# pass arguments to this function :
args = ("Geeks", "for", "Geeks")
myFun(*args)

kwargs = {"arg1" : "Geeks", "arg2" : "for", "arg3" : "Geeks"}
myFun(**kwargs)
```

Run on IDE Copy Code

Output:

```
arg1: Geeks
arg2: for
arg3: Geeks
arg1: Geeks
arg2: for
arg3: Geeks
```

This article is contributed by **Kishlay Verma**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

- Coroutine in Python
- Python Closures
- Precision Handling in Python
- Type Conversion in Python
- copy in Python (Deep Copy and Shallow Copy)
- Packing and Unpacking Arguments in Python
- Get() method for dictionaries in Python
- Generators in Python
- Object Oriented Programming in Python | Set 2 (Data Hiding and Object Printing)
- Function Decorators in Python | Set 1 (Introduction)
- Python Modules
- str() vs repr() in Python
- Returning Multiple Values in Python
- When to use yield instead of return in Python?
- Class or Static Variables in Python

Improved By : DivyaVishwakarma

Article Tags : Python Python-Functions

GeeksforGeeks
A computer science portal for geeks
TECHNICAL
SCRIPTER
2 OCT to 2 DEC
A technical content writing event by GeeksforGeeks

Geeks Classes
Classroom program on Algorithms in Noida
Mentored by Mr Sandeep Jain
Batch starts from 27 Oct, 2018

Be the First to upvote.

2.5

☐ To-do ☐ Done

Based on **17** vote(s)

Feedback Add Notes Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!

▲