Introduction to Function Wrapper in Python

Function wrappers are practical tools for changing a function's functionality. They are referred to as decorators in Python. Without altering the function's original implementation, decorators let us expand the behaviour of a function or a class. A good Python online course will help you learn more about Function Wrapper.

Decorators, usually referred to as wrappers over functions, are a very strong and helpful feature in Python because they let programmers change the behaviour of a function or class. With the help of decorators, we can wrap up another function to increase its functionality without fundamentally altering the original one. Functions are called inside the wrapper function when using decorators, where they are passed as arguments into another function.

Because decorators enable developers to track how long a function takes to execute and run correctly, this is one use for them that is especially helpful. The management of computing resources, such as time and expenses, depends on this process.
When researchers transfer their ideas into computer code, they frequently need to tweak the existing functions to account for their fresh concepts, such as by adding additional parameters or a few extra computations.

To avoid repeatedly changing your codes, wrapper functions can be utilised as an interface to adapt to the existing codes. You might create functions to perform calculations, for instance.

Note that both when the wrapper is defined and when the function is called, all arguments are supplied using the notation (*args, **kwargs).
The packing operation is carried out by the * and ** operators. True to its name, it packs all the arguments into a single dictionary parameter called kwargs and all the keyword arguments into a single tuple argument named args.

Functions can be wrapped without the use of decorators as well. As an illustration, suppose you want to change the input data format for my calculating functions but don't want to alter the my_add and my_deduct scripts because they are contained in various packages. To pre-process the data structure, you may first create a reform_argument function, and then write a wrapper function to wrap the pre-process together with my_add function.

Debugging other functions is further used for function wrappers. It is simple to create a debugger function wrapper in Python that prints the function parameters and return values. With just a few lines of code, you may use this application to examine the reasons why function executions failed.

Python's functools module makes it simple to create custom decorators that can "wrap" (modify/extend) the actions of other functions. In fact, as we'll see, creating function wrappers in Python is extremely similar to creating regular functions. The "@" sign and the name of the wrapper function are then used in the line of code that comes before the function we wish to extend or change once the function decorator has been defined. Similar steps are used to define timer and debugger function wrappers.

Python has a feature called wrappers that allows you to extend the behaviour of a function by wrapping it in another function. Now, the benefit of using wrappers in our code is that they allow us to make changes to a function without actually modifying it.

Python-Jenkins
A more traditional Pythonic method of controlling a Jenkins server is offered via Jenkins, a Python wrapper for the Jenkins REST API. It offers a higher-level API with a selection of useful features. In terms of abstraction, we will see that while interacting with Jenkins, we will work with Python objects rather than JSON objects and HTTP queries.

Here are some use cases of Python-Jenkins:

- Create new jobs
- copy existing jobs
- Get Jenkins plugin information
- Delete jobs
- Update jobs
- Get a job’s build information
- Start a build on a job
- Get Jenkins master version information
- Create nodes
- Enable/Disable nodes
- Get information on nodes
- Create/delete/reconfig views
- Put server in shutdown mode (quiet down)
- List running builds
- Delete builds
- Wipeout job workspace
- create/delete/update folders1

How to create a Wrapper function in Python?

In Python, you would normally utilise a concept known as decorators to build a wrapper function. A function called a "decorator" extends or alters the functionality of another function by passing it as an input without altering the code for that function. Here's how to use decorators to make a straightforward wrapper function:

1. Describe the decorator function, which requires a parameter of another function.
2. Construct the wrapper function that will extend or change the behaviour of the input function inside the decorator.
3. Within the wrapper function, call the input function, and then save the outcome.
4. If necessary, take extra steps or change the outcome.
5. Give the decorator's wrapping function back.
6. Use the decorator to wrap whatever function you need.

Conclusion

So far, we've shown how simple it is to interact with the Jenkins server using the Python-Jenkins wrapper rather than utilising the UI. And we can complete practically all tasks in a matter of seconds. You can learn more by checking the Python course online.

Tags: #onlinecertificatepython #bestpythononlinetraining #Toptenonlinetrainingpython #PythonCourseOnline