Exec () function in Python

The exec () function used to execute Python programs dynamically can be string or object code. How does exec () function syntax, what parameters do it have, and how is it used? Invites you to read the track.

The exec () function used to execute Python programs dynamically can be string or object code. How does exec () function syntax, what parameters do it have, and how is it used? Invites you to read the track.

The exec () function syntax in Python

```
exec(doituong, global, local)
```

The parameters of exec ():

Exec () has three parameters:

1. *doituong:* a string or object code
2. *global:* a dictionary that specifies methods and global variables available. Global is an optional parameter.
3. *local:* a map object. Local is an optional parameter.

Global and local use will be analyzed by Quantrimang later in this article.

Example 1: How does exec () function?

```
# ???c vi?t b?i TipsMake.com
program = 'a = 5nb=10nprint("T?ng =", a+b)'
exec(program)
```

Run the program, the result is:

```
T?ng = 15
```

In this example, the object string is passed into *exec (),* the global and local parameters are ignored in this case.

Example 2: Allow users to enter input

```
program = input('Enter a program:')
exec(program)
```

Run the program, the result is:

```
Enter a program: [print(item) for item in [1, 2, 3]]
1
2
3
```

If you want to get Python code from users, you can use the *compile ()* function before using *exec ().*

You should be careful when using exec!

You need to pay a little attention if you are using a Unix system like macOS, Linux . and perform the functions of the os module. The os module is built to provide methods to help you create, delete, and change folders.

If you then enter the value as *exec (input ()),* the user may have problems changing the file even if it deletes the file with the *os.system ('rm* -rf *') command.

So if you use *exec (input ())* in your code, it is reasonable to check which variables and methods you can use. You should do this with the function *dir ().*

```
from math import *
 exec('print(dir())' )
```

Running the code and the resulting result will look like this:

```
['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil',
 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factoria
 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi
 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

The case does not pass global and local parameters

When using *exec ()* without passing these two parameters as the examples above, the expression will be executed in the current scope. You can check the variables and methods available with *dir ()* as mentioned above.

Where to pass global parameters and ignore local

Global and *local* (dictionary) parameters are used for global and local variables respectively. If dictionary local is omitted, this default parameter program is *global.* That is, *global* will be used for both global and local variables.

You can check local and global dictionary with *global ()* and *local ()* built-in functions *()*

Example: Transfer an empty dictionary to the global parameter

```
from math import *
exec('print(dir())', {})

# Code d??i ?ây s? x?y ra exception
# print(exec('sqrt(25)', {}))
```

If you pass a blank dictionary into the global, only the *__builtins__* function is valid for doituong. You will not be able to access any function of the math module even though you have imported the math module into the program above.

Run the program, the result is:

```
['__builtins__']
```

In this example, the expression can use the *sqrt ()* and *pow () functions* together with *__builtins__*.

```
from math import *
exec('print(dir())', {'sqrt': sqrt, 'pow': pow})

exec('print(sqrt(9))', {'sqrt': sqrt, 'pow': pow})
```

Alternatively, you can change the name of the available method for your desired expression.

```
from math import *
exec('print(dir())', {'canbachai': sqrt, 'pow': pow})

# object can have canbachai() module
exec('print(canbachai(9))', {'canbachai': sqrt, 'pow': pow})
```

The case transmits both global and local parameters

```
from math import *

globalsParameter = {'__builtins__' : None}
localsParameter = {'print': print, 'dir': dir}
exec('print(dir())', globalsParameter, localsParameter)
```

Run the program, the result is returned

```
['dir', 'print']
```

Here, only the *print ()* and *dir ()* built-in functions can be executed by the *exec ()* function .

The important thing to note is that *exec ()* executes but does not return any values ??(returns *None).* Therefore, you cannot use *return* and *yield statements* outside of defined functions.

Last lesson: float () function in Python

Next lesson: Hex () function in Python