migo_6_funcs

June 1, 2024

bring in functionality from other modules or packages into your current code.

import : import module_name

[ ]:
```
import math
print(math.sqrt(25))
```

as : import module_name as alias

[ ]:
```
import math as m
print(m.sqrt(25))
```

from : from module_name import item1, item2, ...

[ ]:
```
from math import sqrt
print(sqrt(25))
```

'' : *from module_name import*

[ ]:
```
from math import *
print(sqrt(25))
```

help

[1]:
```
help()
```

```
Welcome to Python 3.12's help utility! If this is your first time using
Python, you should definitely check out the tutorial at
https://docs.python.org/3.12/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To get a list of available
modules, keywords, symbols, or topics, enter "modules", "keywords",
"symbols", or "topics".

Each module also comes with a one-line summary of what it does; to list
the modules whose name or summary contain a given string such as "spam",
enter "modules spam".

To quit this help utility and return to the interpreter,
```

```
enter "q" or "quit".


help>  q


You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
```

def Defining a Function:

[4]:
```
def greet(name):
    print("Hello, " + name + "!")
```

Calling a Function

[5]:
```
greet("Meny")
```

```
Hello, Meny!
```

return : Return Statement

[3]:
```
def add(a,b):
    a,b=b,a
    return a+b
a=3;b=4
res = add(a,b)
print(res)
print(a,b)
```

```
7
3 4
```

None

[29]:
```
def greet(name):
    print("hello "+name)

a=greet("lala")
print(a)
if not a:
    print("a has a value")
else:
    print("a hasn't a value")
if a==None:
    print("a is None")
print(not None)
print(not False)
print(False == None)
```

```
hello lala
None
a has a value
a is None
True
True
False
```

Parameters and Arguments:#Parameters are the variables listed inside the parentheses in the function definition. Arguments are the values passed into the function when it is called.

[ ]:
```python
def multiply(x, y):
    return x * y

result = multiply(5, 6)
print(result)
```

Default Arguments:

[10]:
```python
def power(base, exponent=2):
    return base ** exponent

print(power(3))
print(power(2, 3))
print(power(exponent=5,base=2))
```

```
9
8
32
```

Docstrings: **doc**

[12]:
```python
def greet(name):
    """This function greets the person with the given name.""" # triple "
    print("Hello, " + name + "!")
print(greet("lala"))
print(greet.__doc__)
```

```
Hello, lala!
None
This function greets the person with the given name.
```

Scope of Variables:

local

[13]:
```python
def func(a):
    a=20
a=5
func(a)
print(a)
```

```
5
```

global

[8]:
```python
x = 10
y = 5
def func():
    global x
    x = 20
    y = 15
    print(f"Inside function : x={x} , y={y}")

func()
print(f"Outside function: x={x} , y={y}")
```

```
Inside function : x=20 , y=15
Outside function: x=20 , y=5
```

nonlocal : within a nested function

[7]:
```python
x=7
def outer():
    x = 10

    def inner():
        nonlocal x
        x = 20
        print("Inner:", x)

    inner()
    print("Outer:", x)

outer()
print(x)
```

```
Inner: 20
Outer: 20
7
```

specify the types of parameters using type annotations

[14]:
```python
def greet(name: str, age: int) -> str:
    """
    Greets a person by name and age.

    Parameters:
    name (str): The name of the person.
    age (int): The age of the person.

    Returns:
```

```
    str: A greeting message.
    """
    return f"Hello, {name}! You are {age} years old."

message = greet("Alice", 30)
print(message)
```

```
Hello, Alice! You are 30 years old.
```

[15]:
```
def linear_search(lst: list[int], key: int) -> int:
    """
    Performs a linear search for a key in a list of integers.

    Parameters:
    lst (list[int]): The list of integers to search.
    key (int): The key to search for.

    Returns:
    int: The index of the key if found, otherwise -1.
    """
    for index, value in enumerate(lst):
        if value == key:
            return index
    return -1

# Example usage:
numbers = [4, 2, 9, 7, 5, 6]
key = 7
result = linear_search(numbers, key)

if result != -1:
    print(f"Key found at index: {result}")
else:
    print("Key not found in the list.")
```

```
Key found at index: 3
```

*args and **kwargs

Use *args when you want to pass a variable number of positional arguments to a function. Use **kwargs when you want to pass a variable number of keyword arguments to a function. Use both *args and **kwargs when you want to accept any combination of positional and keyword arguments in a function.

[6]:
```
def my_function(*args, **kwargs):
    for arg in args:
        print(arg,end=",")
    print()
    for key, value in kwargs.items():
```

```
        print(key, value ,sep=":",end=",")

my_function('apple', 'banana', 'orange', name='John', age=30)
```

```
apple,banana,orange,
name:John,age:30,
```

[ ]: