

Function

Using functions in shell scripts allows you to organize your code, promote reusability, and make your scripts more manageable. Here's a guide on how to define and use functions in a Bash shell script.

1. Defining a Function

```
function_name() {  
    # Commands to be executed  
}
```

or

```
function function_name {  
    # Commands to be executed  
}
```

Example:

- Here's a simple example of a function that prints a greeting:

```
greet() {  
    echo "Hello, $1!"  
}
```

- To call the function, simply use its name followed by any required arguments:

```
greet "Alice" # Output: Hello, Alice!
```

2. Using Return Values:

Functions can return a status code (an integer) using the **return** statement. By convention, a return value of **0** indicates success, while any non-zero value indicates an error.

Exmp:

A.

```
add() {  
    result=$(( $1 + $2 ))  
    return $result  
}
```

```
# Call the function  
add 5 3  
echo $? # Output: 8 (the exit status of the  
last command)
```

B. To return a value that can be used outside of the function, you can use **echo** and capture the output when calling the function:

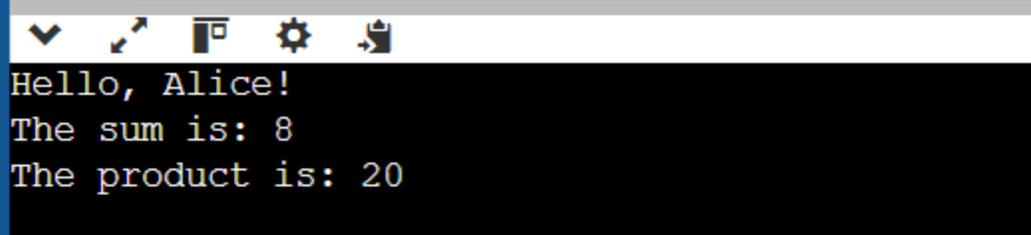
```
multiply() {  
    echo=$(( $1 * $2 ))  
}
```

```
result=$(multiply 4 5)  
echo "The result is $result" # Output: The  
result is 20
```

Example:

Here's a complete example script that uses functions:

```
8
9 # Function to greet a user
10 greet() {
11     echo "Hello, $1!"
12 }
13
14 # Function to add two numbers
15 add() {
16     echo $(( $1 + $2 ))
17 }
18
19 # Function to multiply two numbers
20 multiply() {
21     echo $(( $1 * $2 ))
22 }
23
24 # Calling functions
25 greet "Alice"
26
27 sum=$(add 5 3)
28 echo "The sum is: $sum"
29
30 product=$(multiply 4 5)
31 echo "The product is: $product"
32
```



Terminal window showing the execution of a shell script. The output is:

```
Hello, Alice!
The sum is: 8
The product is: 20
```

