**Functions in Bash Shell Scripting**

-Functions in Bash are blocks of reusable code that can be called multiple times within a script.

-They help modularize and simplify scripts.

**Syntax:**

function\_name() {

# Commands or statements

}

-------

function function\_name {

# Commands or statements

}

**To call the function:**

function\_name

--------------

greet() {

echo "Hello, $1!"

}

greet "Shailaja"

-----------

function greet() {

echo "Hello, $1!"

}

greet "Shailaja"

------------

**Function with Arguments**

-Arguments to a function are accessed like positional parameters in a script ($1, $2, etc.).

-$@ contains all arguments passed to the function.

add\_numbers() {

local sum=$(( $1 + $2 )) # 'local' makes the variable scoped to the function

echo "Sum: $sum"

}

add\_numbers 5 10

---------------------

**Return Values**

Functions can return values using the return statement (integer only) or echo (any value).

Use $? to capture the integer return value.

**-Using return for integers**

multiply() {

return $(( $1 \* $2 ))

}

multiply 3 4

echo "Product: $?"

Output:

Product: 12

**-Using echo for other values**

divide() {

local result=$(( $1 / $2 ))

echo $result

}

result=$(divide 10 2)

echo "Quotient: $result"

Output:

Quotient: 5

--------------------------

**Function Scope**

-**Global Variables**: By default, variables inside functions are global.

-**Local Variables**: Use local keyword to restrict the scope of a variable to the function.

my\_function() {

local local\_var="I am local"

global\_var="I am global"

echo $local\_var

}

my\_function

echo $global\_var # Accessible outside

echo $local\_var # Not accessible outside

### ****Bash Arrays****

Arrays in Bash are used to store multiple values in a single variable. They can be indexed or associative (key-value pairs). Below is a detailed guide to using arrays in Bash.

### ****1. Indexed Arrays****

Indexed arrays use numeric indices starting from 0.

#### 1. Declaring and Initializing

* Explicit declaration:

declare -a array\_name

* Assigning values:

array\_name=(value1 value2 value3)

* Adding individual elements:

array\_name[0]="value1"

array\_name[1]="value2"

### ****2. Accessing Elements****

* Access a specific element:

echo ${array\_name[0]} # Access the first element

* Access all elements:

echo ${array\_name[@]}

* Access the length (number of elements):

echo ${#array\_name[@]}

### ****3. Modifying Arrays****

* Update an element:

array\_name[1]="new\_value"

* Remove an element:

unset array\_name[1] # Removes the second element

* Add elements:

array\_name+=(new\_value1 new\_value2)

### ****4. Iterating Over Arrays****

* Using a for loop:

array=(one two three)

for element in "${array[@]}"; do

echo $element

done

**Output:**

one

two

three

* Using indices:

declare array=(1 2 3)

for i in "${!array[@]}"; do

"Index: $i, Value: ${array[i]}"

Done

**Output:**

Index: 0, Value: 1

Index: 1, Value: 2

Index: 2, Value: 3

### ****5. Array Operations****

#### ****Length of Array****

* For indexed arrays:
* echo ${#array[@]} # Number of elements
* For a specific element (string length):
* echo ${#array[0]}

#### ****Slicing****

Extract a subset of elements:

array=(one two three four five)

echo ${array[@]:1:3} # Output: two three four

### ****6. Useful Commands for Arrays****

* **Check if an element exists**:

if [[ " ${array[@]} " =~ "value" ]]; then

echo "Exists!"

fi

### ****Example Script****

#!/bin/bash

# Declare and initialize an array

fruits=("apple" "banana" "cherry")

# Add an element

fruits+=("date")

# Print all elements

echo "Fruits: ${fruits[@]}"

# Length of the array

echo "Number of fruits: ${#fruits[@]}"

# Loop through the array

for fruit in "${fruits[@]}"; do

echo "Fruit: $fruit"

done

**Output:**

Fruits: apple banana cherry date

Number of fruits: 4

Fruit: apple

Fruit: banana

Fruit: cherry

Fruit: date

**Strings:**

Strings are an essential part of shell scripting in Bash. Various operations on strings, such as creating, manipulating, and extracting information.

**1. Declaring Strings**

* Single-quoted strings:

str='Hello, World!' # Single quotes prevent variable and command substitution

echo $str

* Double-quoted strings:

str="Hello, $USER!" # Double quotes allow variable and command substitution

Example:

str='Hello, World!'

echo $str

str='Hello, World $str!'

echo $str

str="Hello, $str !"

echo $str

**Output:**

Hello, World!

Hello, World $str!

Hello, Hello, World $str! !

**2. Concatenating Strings**

Combine strings using simple syntax:

str1="Hello"

str2="World"

str3="$str1, $str2!"

echo $str3

**3. String Length**

Find the length of a string:

str="Hello, World!"

echo ${#str}

**Output**

?

**4. Extracting Substrings**

**syntax**

${string:position:length}:

.

str="Hello World!"

echo ${str:6:5} # Output: World

-Starts extracting the substring from index **6** (0-based indexing)

-Specifies the length of the substring to extract. It extracts **5 characters** starting from index 6

**5. Check if a String Contains a Substring**

Using [[ with \*:

str="Hello, World!"

if [[ $str == \*"World"\* ]]; then

echo "Substring found!"

fi

**6. Convert Case**

* Convert to lowercase:

str="Hello, World!"

echo "${str,,}" # Output: hello, world!

* Convert to uppercase:

echo "${str^^}" # Output: HELLO, WORLD!

**Example Script**

#!/bin/bash

# Declare a string

str="Welcome to Bash Scripting"

# Print the string

echo "Original String: $str"

# Get length

echo "Length: ${#str}"

# Substring

echo "Substring: ${str:11:4}"

# Replace

echo "Replace: ${str/Scripting/Programming}"

# Convert case

echo "Lowercase: ${str,,}"

echo "Uppercase: ${str^^}"

**6. Search and Replace**

* Replace the first occurrence:

str="Hello, World!"

echo ${str/World/Bash} # Output: Hello, Bash!

* Replace all occurrences:

echo ${str//l/L} # Output: HeLLo, WorLd!

* Remove part of a string:

echo ${str/World/} # Output: Hello, !