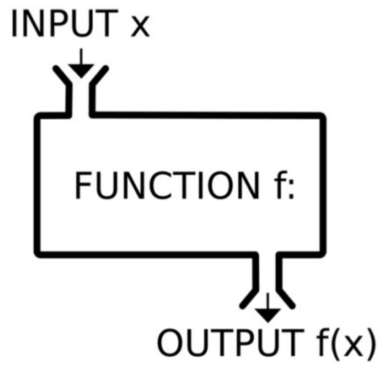
BATCH	BATCH 48
LESSON	**BATCH SCRIPTING**
DATE	04.01.2021
SUBJECT	**BASH FUNCTIONS**

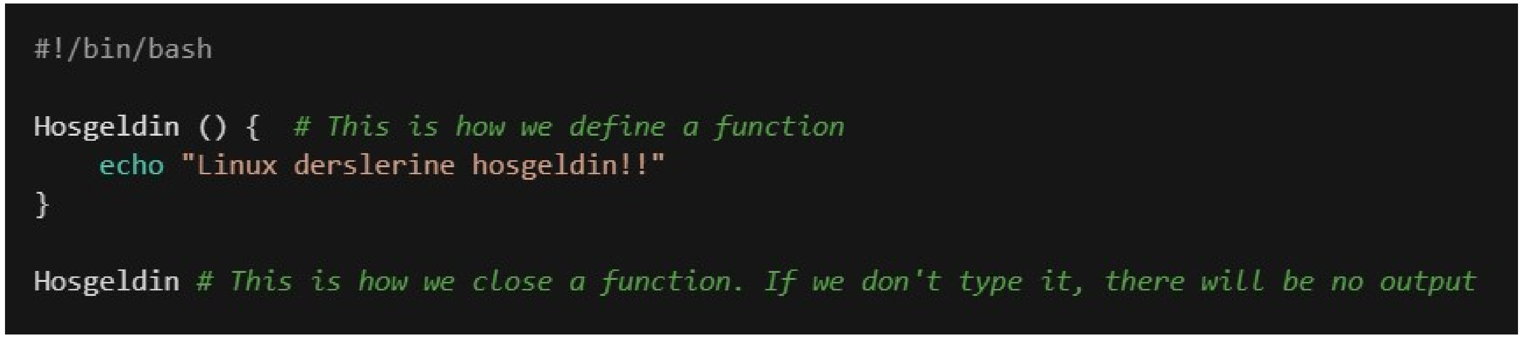
FUNCTIONS

- Functions within a shell script is a piece of code or a block of code that perform a particular function that can be reused.
- Functions enable you to reuse code. We can call the functions numerous times.
- When a shell script runs it runs line-by-line. So your Function must **always be defined first** before calling it, if not then it will give error.
- The return statement within a function call helps in specifying the exit code for that function. It is just like the exit code for the entire script but in this case it wont exit the script but the function.



FUNCTIONS – When to use?

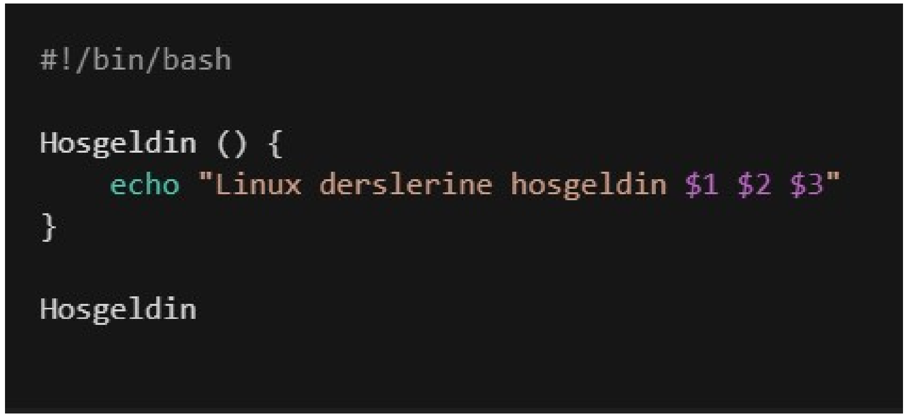
- Break up large script that performs many different tasks
- Installing packages
- Adding users
- Configuring firewalls
- Perform Mathematical calculations



```
#!/bin/bash

Hosgeldin () {  # This is how we define a function
    echo "Linux derslerine hosgeldin!!"
}

Hosgeldin # This is how we close a function. If we don't type it, there will be no output
```

FUNCTIONS – Passing Arguments to Functions

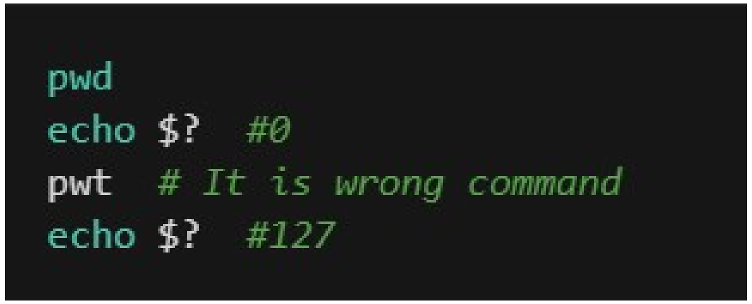
- We can pass any number of arguments to the bash function in a similar way to passing command line arguments to a script. We simply supply them right after the function’s name, separated by a space. These parameters would be represented by $1, $2 and so on, corresponding to the position of the parameter after the function’s name.



```
#!/bin/bash

Hosgeldin () {
    echo "Linux derslerine hosgeldin $1 $2 $3"
}

Hosgeldin
```

FUNCTIONS – Returning Values from Functions

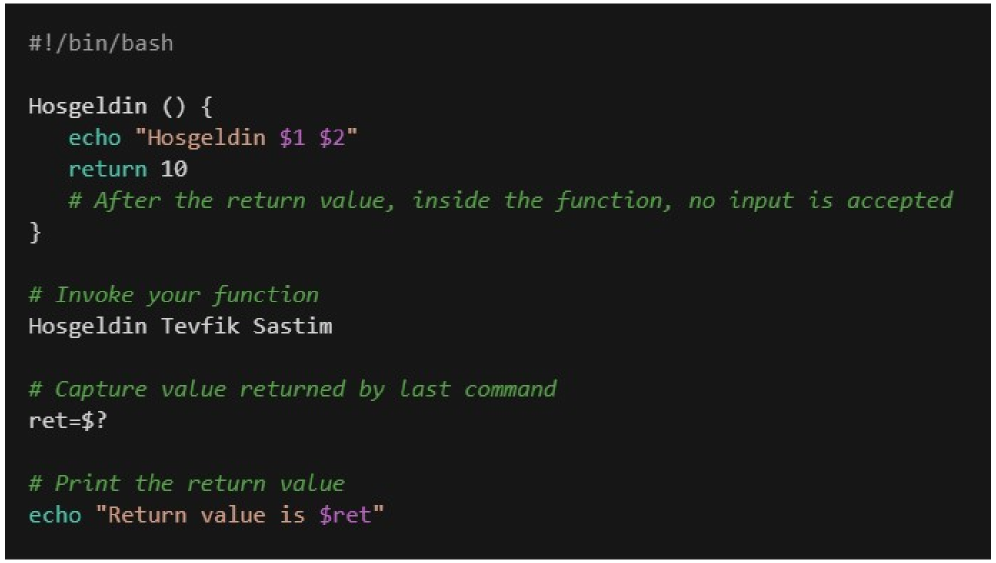
- Functions in other programming languages return a value when called. But, Bash functions don’t return a value when called. But we can define a return status similar to exit status of a command.
- When any shell command terminates, it returns an exit code, which indicates 0 for success and non-zero decimal number in the 1 - 255 range for failure. The special variable $? returns the exit status of the last executed command.



```
pwd
echo $?  #0
pwt  # It is wrong command
echo $?  #127
```

FUNCTIONS – Returning Values from Functions

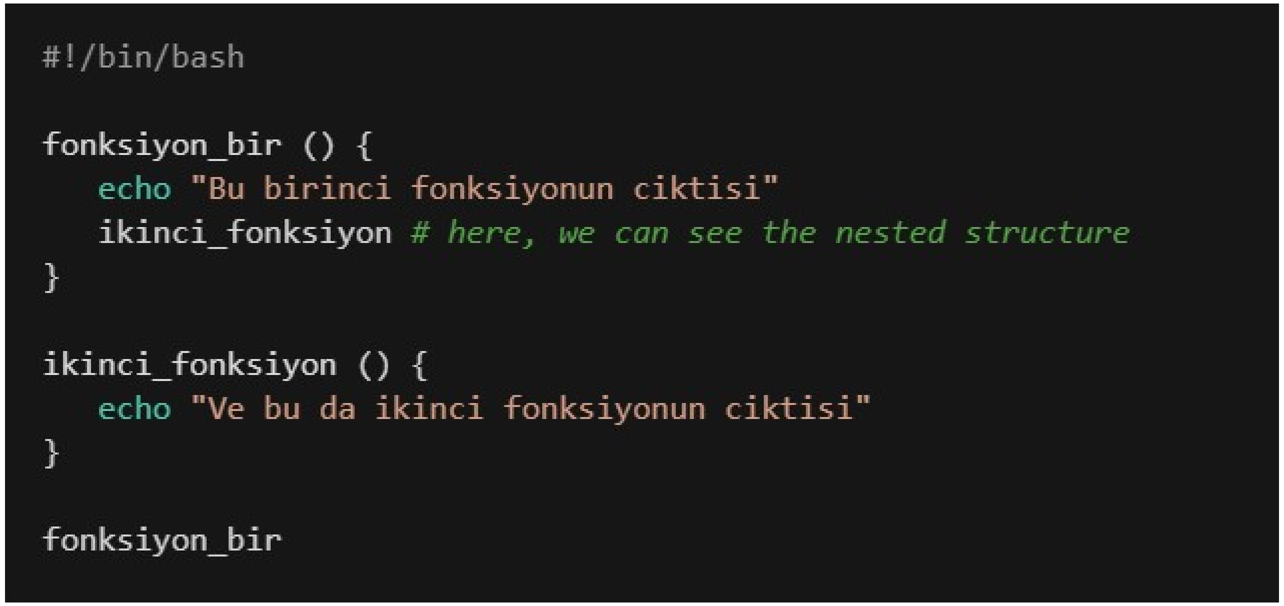
- When a return statement is used in a function body, the execution of the function is stopped.
- When a bash function completes, its return value is the status of the last statement executed in the function. We can specify return status by using the return keyword. We can think the return keyword as exit status of function.



```
#!/bin/bash

Hosgeldin () {
   echo "Hosgeldin $1 $2"
   return 10
   # After the return value, inside the function, no input is accepted
}

# Invoke your function
Hosgeldin Tevfik Sastim

# Capture value returned by last command
ret=$?

# Print the return value
echo "Return value is $ret"
```

FUNCTIONS – Nested Functions

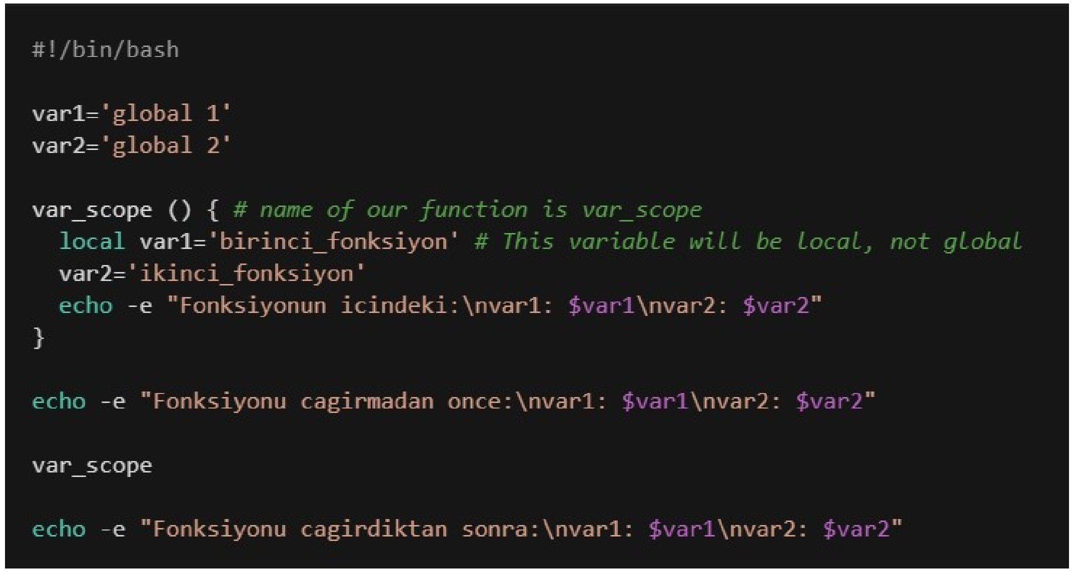
- One of the useful features of functions is that they can call themselves and other functions.



```
#!/bin/bash

fonksiyon_bir () {
   echo "Bu birinci fonksiyonun ciktisi"
   ikinci_fonksiyon # here, we can see the nested structure
}

ikinci_fonksiyon () {
   echo "Ve bu da ikinci fonksiyonun ciktisi"
}

fonksiyon_bir
```

FUNCTIONS – Variable Scope

- Global variables are variables that can be accessed from anywhere in the script regardless of the scope. In Bash, by default all variables are defined as global, even if declared inside the function.
- Local variables can be declared within the function body with the local keyword and can be used only inside that function.



```
#!/bin/bash

var1='global 1'
var2='global 2'

var_scope () { # name of our function is var_scope
  local var1='birinci_fonksiyon' # This variable will be local, not global
  var2='ikinci_fonksiyon'
  echo -e "Fonksiyonun icindeki:\nvar1: $var1\nvar2: $var2"
}

echo -e "Fonksiyonu cagirmadan once:\nvar1: $var1\nvar2: $var2"

var_scope

echo -e "Fonksiyonu cagirdiktan sonra:\nvar1: $var1\nvar2: $var2"
```