# Perl Assignment No.5

## Q1. What is Error Handling in which also define run time error and compile time error?

**Ans:** Error Handling in Perl is the process of taking appropriate action against a program that causes difficulty in execution because of some error in the code or the compiler. Processes are prone to errors. For example, if opening a file that does not exist raises an error, or accessing a variable that has not been declared raises an error.
The program will halt if an error occurs, and thus using error handling we can take appropriate action instead of halting a program entirely. Error Handling is a major requirement for any language that is prone to errors.

Errors can be classified by the time at which they occur:

a)Compile Time Errors.

b)Run Time Errors.

**Compile Time Error:** It is an error such as syntax error or missing file reference that prevents the program from successfully compiling.

**Run Time Error :** It is an error that occurs when the program is running and these errors are usually logical errors that produce the incorrect output.

## Q2. Write a short note on warn function, die function, and unless function.

**Ans:**

**A)The warn function:-**

The warn function just raises a warning, a message is printed to STDERR, but no further action is taken. So it is more useful if you just want to print a warning for the user and proceed with rest of the operation –

```
chdir('/etc') or warn "Can't change directory";
```

**B)The die Function:-**

The die function works like warn function.

The die function is used to signal the occurrences of fatal errors in the sense that the program in question should not be allowed to continue.

**OR**

The die function works like warn function.

This function has the effect of immediately terminating execution if an error occurs. You should use this function in case it is useless to proceed if there is an error in the program

```
chdir('/etc') or die "Can't change directory";
```

**C)The Unless Function:-**

The statements in the code block will only be executed if the expression returns false.

```
die "Error: Can't change directory!: $!" unless(chdir("/etc"));
```

Here we die only if the chdir operation fails, and it reads nicely. The unless statement is best used when you want to raise an error or alternative only if the expression fails.

## Q3.Write a short note on Carp Function and Cluck Function with example.

**Ans:**

**A)Carp Function:-**

The carp function is the basic equivalent of warn and prints the message without actually exiting the script and printing the script name.

**Ex.**

```
package test;
use Carp;
Sub functionName{
Carp "Error in Module!"
}
1;
use test;
functionName();
```

It will produce the following result –

Error in module! at test.pl line 2

**B)Cluck Function:-**

The cluck function is a sort of supercharged carp, it follows the same basic principle but also prints a stack trace of all the modules that led to the function being called, including the information on the original script.

**Ex.**

```
package test;
use Carp qw(cluck);
```

```
sub functionName{
cluck "Error in module!";
}
1;
use T;
functionName();
```

It will produce the following result –

```
Error in module! at test.pm line 5
    test::functionName() called at test.pl line 11
```

## Q4. Write a short note on Croak Function and Confess Function with example.

**Ans:**

**A)Croak Function:-**

The croak function is equivalent to die, except that it reports the caller one level up. Like die, this function also exits the script after reporting the error to STDERR −

**Ex.**

```
package test;
use carp;
sub functionName{
croak "Error in module!";
}1;
use test;
functionName();
```

It will produce the following result –

```
Error in module! at test.pl line 2
```

**B)Confess Function:-**

The confess function is like cluck; it calls die and then prints a stack trace all the way up to the origination script.

**Ex.**

```
package test;
use Carp;
```

```
sub functionName{
confess "Error in module!";
}1;
use test;
functionName();
```

It will produce the following result –

```
Error in module! at test.pm line 5
    test::functionName() called at test.pl line 11
```

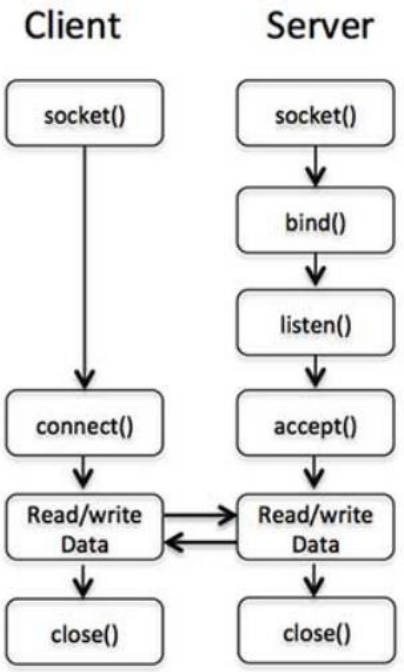
## Q5. What is socket programming also explain to create server and to create client?

**Ans:**

Socket is a Berkeley UNIX mechanism of creating a virtual duplex connection between different processes. This was later ported on to every known OS enabling communication between systems across geographical location running on different OS software. If not for the socket, most of the network communication between systems would never ever have happened.

Taking a closer look; a typical computer system on a network receives and sends information as desired by the various applications running on it. This information is routed to the system, since a unique IP address is designated to it. On the system, this information is given to the relevant applications, which listen on different ports. For example an internet browser listens on port 80 for information received from the web server. Also we can write our custom applications which may listen and send/receive information on a specific port number.

Let's sum up that a socket is an IP address and a port, enabling connection to send and receive data over a network. So that we need to understand client server architecture.




## To create a Server:-

- Create a socket using socket call.
- Bind the socket to a port address using bind call.
- Listen to the socket at the port address using listen call.
- Accept client connections using accept call.

## To create a Client:-

- Create a socket with socket call.
- Connect the socket to the server using connect call.