PERL

Day 2

Agenda

- given-when Statement
- Loops
- Subroutine
- File Handling
- Throw an Exception
- File and Directory Manipulation

given-when Statement

given-when Statement

- given-when in Perl is similar to the switch-case of C/C++, Python or PHP.
- Just like the switch statement, it also substitutes multiple if-statements with different cases.
- Syntax:

```
given(expression)
{
    when(value1) {Statement;}
    when(value2) {Statement;}

    default {# Code if no other case matches}
}
```

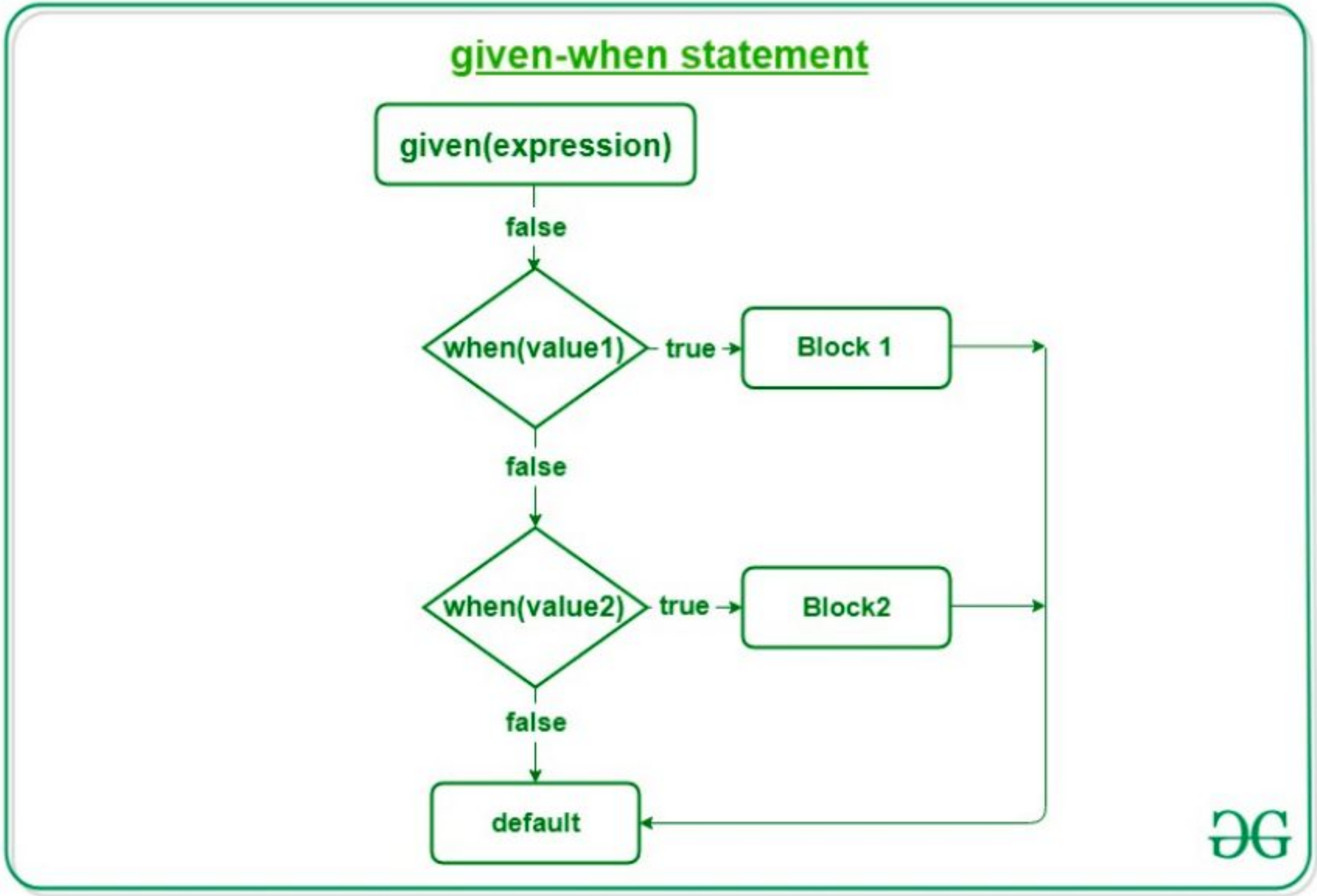
J-01-given-when-1.pl
J-02-given-when-2.pl

given-when Statement (Cont'd)

- given-when statements also use two other keywords along with it, namely **break** and **continue**.
- **break:**
  break keyword is used to break out of a when block. In Perl, there is no need to explicitly write the break after every when block. It is already defined implicitly.
- **continue:**
  continue, on the other hand, moves to the next when block if first when block is correct.

given-when Statement (Cont’d)

- In a given-when statement, a conditional statement must not repeat in multiple when statements, this is because Perl checks for the first occurrence of that condition only and the next repeating statements will be ignored.
- Also, a default statement must be placed after all the when statements because the compiler checks for condition matching with every when statement in order, and if we will place default in between, then it will take a break over there and will print the default statement.


given-when statement
given(expression)
false
when(value1)
true
Block 1
false
when(value2)
true
Block2
false
default

Loops

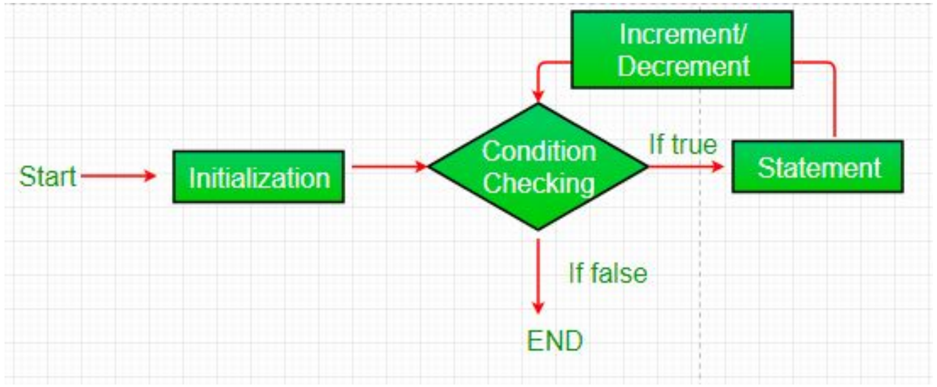
Loops

- Looping in programming languages is a feature which facilitates the execution of a set of instructions or functions repeatedly while some condition evaluates to true.
- Perl provides the different types of loop to handle the condition based situation in the program. The loops in Perl are :
  - for loop
  - foreach loop
  - while loop
  - do .. while loop
  - until loop
  - Nested loops

for Loop

- for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.
- Syntax:

```
for (init statement; condition; increment/decrement ) {
    # Code to be Executed
}
```



K-01-for.pl

for Loop (Cont’d)

A for loop works on a predefined flow of control. The flow of control can be determined by the following :

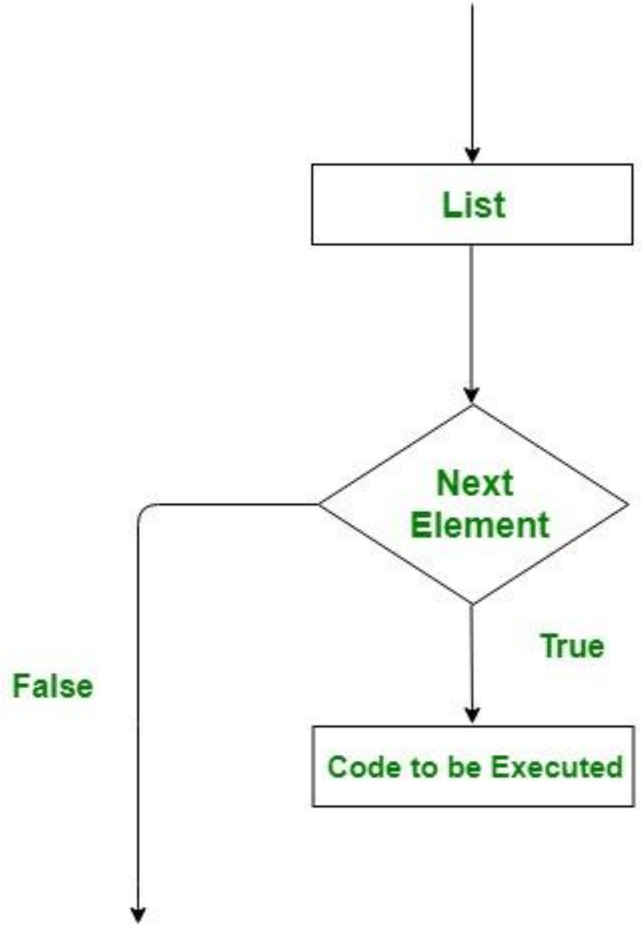
- **init statement:** This is the first statement which is executed. In this step, we initialize a variable which controls the loop.
- **condition:** In this step, the given condition is evaluated and the for loop runs if it is True. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.
- **Statement execution:** Once the condition is evaluated to true, the statements in the loop body are executed.
- **increment/decrement:** The loop control variable is changed here (incremented or decremented) for updating the variable for next iteration.
- **Loop termination:** When the condition becomes false, the loop terminates marking the end of its life cycle.

foreach Loop

- A foreach loop is used to iterate over a list and the variable holds the value of the elements of the list one at a time.
- A foreach loop is used to iterate over a list and the variable holds the value of the elements of the list one at a time.
- The process of iteration of each element is done automatically by the loop.
- Syntax:

```
foreach variable
{
    # Code to be Executed
}
```

K-02-foreach.pl

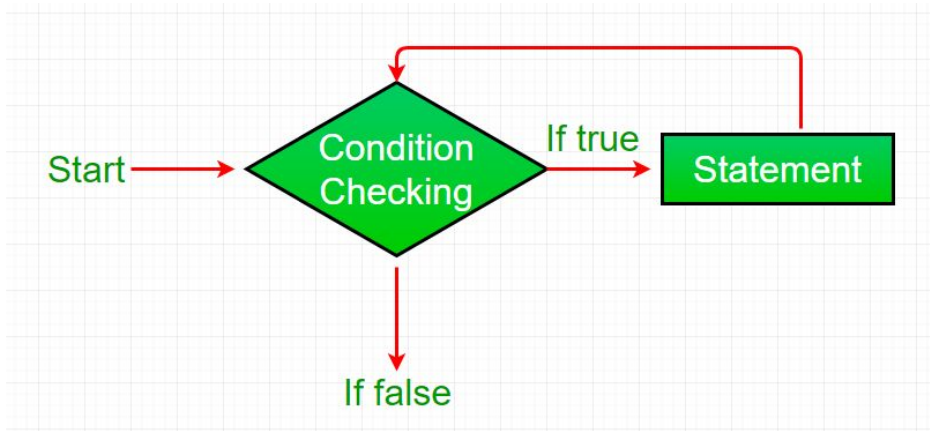

List
Next Element
True
False
Code to be Executed

while Loop

- A while loop generally takes an expression in parenthesis. If the expression is True then the code within the body of while loop is executed.
- A while loop is used when we don't know the number of times we want the loop to be executed however we know the termination condition of the loop.
- Syntax:

```
while (condition)
{
    # Code to be executed
}
```

K-03-while.pl

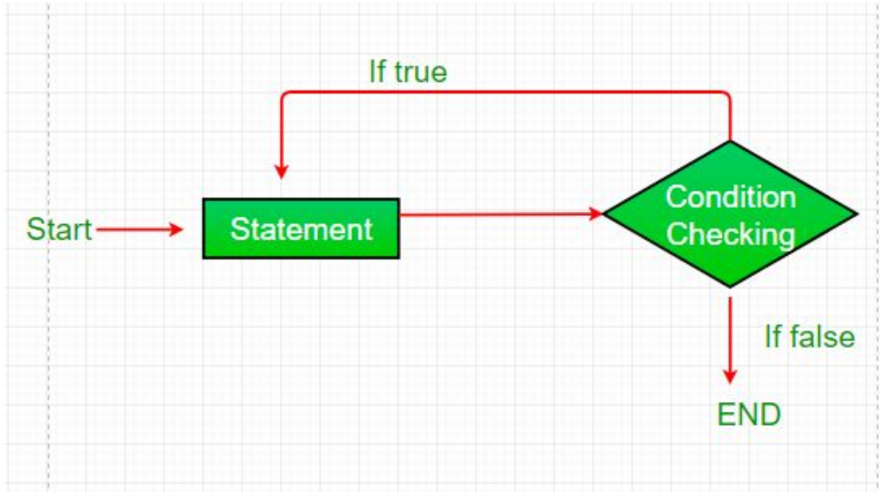


do.... while loop

- A do..while loop is almost same as a while loop. The only difference is that do..while loop runs at least one time. The condition is checked after the first execution.
- A do..while loop is used when we want the loop to run at least one time.
- Syntax:

```
do {
    # statments to be Executed
} while(condition);
```

K-04-do-while.pl

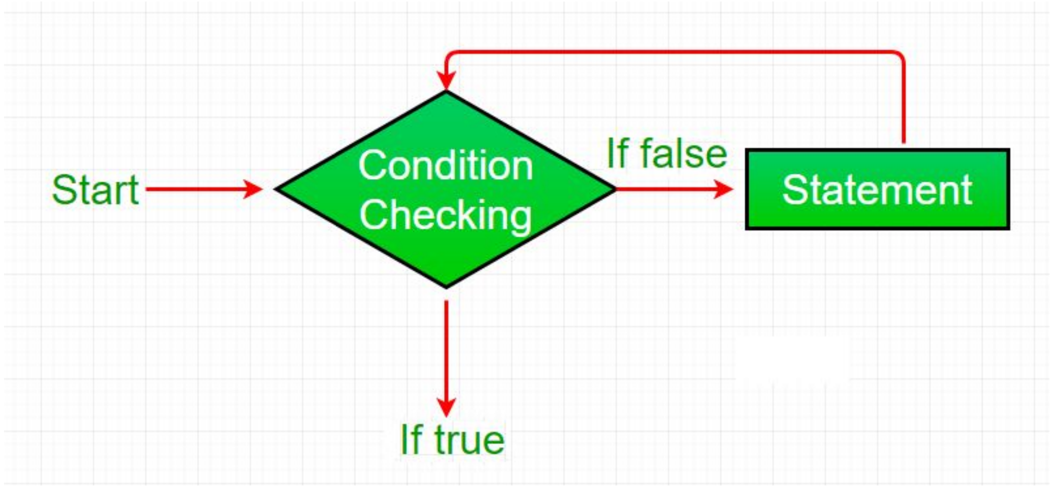


until loop

- until loop is the opposite of while loop.
- It takes a condition in the parenthesis and it only runs until the condition is false.
- Basically, it repeats an instruction or set of instruction until the condition is FALSE.
- Syntax:

```
until (condition)
{
   # Statements to be executed
}
```

K-05-until.pl



Subroutine

Functions or Subroutines

- A function/Subroutine is a block of code written in a program to perform some specific task.
- **Built-in Functions:**
  Perl provides us with a huge collection of built-in library functions. These functions are already coded and stored in the form of functions.
- **User Defined Functions:**
  Perl allows us to create our own customized functions called the user-defined functions or Subroutines.

Subroutine

- A Perl function or subroutine is a group of statements that together perform a specific task.
- So the user puts the section of code in function or subroutine so that there will be no need to write code again and again.
- The word **subroutines** is used most in Perl programming because it is created using keyword **sub**.
- The general form of defining the subroutine in Perl is as follows:

```
sub subroutine_name
{
    # body of method or subroutine
}
```

L-01-subroutine.pl

Subroutine (Cont’d)

- Calling Subroutines:

  subroutine_name(aruguments_list);
- Passing parameters to subroutines:

  - This is used to pass the values as arguments.This is done using special list array variables ‘**$_**’.

  - This will assigned to the functions as $_[0], $_[1] and so on.

L-02-pass-arguments.pl

Subroutine (Cont'd)

- Passing Lists to Subroutines:

  - As we know that @_ is a special array variable inside a function or subroutine, so it is used to pass the lists to the subroutine.

  - In order to pass a list along with other scalar arguments, it is necessary to make the list as the last argument.

- Different number of parameters in subroutine call:

  - Perl does not provide us any built-in facilities to declare the parameters of a subroutine, which makes it very easy to pass any number of parameters to a function.

L-03-passing-list.pl
L-04-different-parameters-numbers.pl

Returning Value

- Returning Value from a Subroutine:

  - Subroutine can also return a value like in other programming languages such as C, C++, Java etc.

  - If the user will not return a value from subroutine manually, then the subroutine will return a value automatically. In this, the automatically returned value will be the last calculation executed in the subroutine.

  - The return value may be scalar, array or a hash.

L-05-return-from-subroutine.pl

Pass By Reference

- When a variable is passed by reference function operates on original data in the function. Passing by reference allows the function to change the original value of a variable.
- When the values of the elements in the argument arrays @_ are changed, the values of the corresponding arguments will also change. This is what passing parameters by reference does.

L-06-pass-by-reference-1.pl
L-07-pass-by-reference-2.pl

File Handling

File Handling

- In Perl, a FileHandle associates a name to an external file, that can be used until the end of the program or until the FileHandle is closed.
- In short, a FileHandle is like a connection that can be used to modify the contents of an external file and a name is given to the connection (the FileHandle) for faster access and ease.

N-01-read-only.pl
N-02-write-only-create-if-need.pl
N-02-append-create-if-need.pl
N-04-read-write-not-create.pl
N-05-read-write-create-if-need.pl
N-06-read-append-create-if-need.pl

Open File

- Syntax:

  open(FileHandle, Mode, FileName);
- Parameters:

  - FileHandle: The reference to the file, that can be used within the program or until its closure.

  - Mode: Mode in which a file is to be opened.

  - FileName: The name of the file to be opened.

Close File

- Syntax:

  close(FileHandle);
- Parameters:

  - FileHandle: The FileHandle to be closed.

Reading from a File using FileHandle

- Reading from a FileHandle can be done through the print function:
- Syntax:

```
print(<FileHandle>);
```

- Parameters:

  - FileHandle: FileHandle opened in read mode or a similar mode.

Writing to a File using FileHandle

- Writing to a File can also be done through the print function:
- Syntax:

    print FileHandle String
- Parameters:

    - FileHandle: FileHandle opened in write mode or a similar mode.

    - String: The String to be inserted in the file.

Different Modes in File Handling

MODE	MODE	EXPLANATION
r	“<“	Read Only Mode
w	“>”	Creates file (if necessary), Clears the contents of the File and Writes to it
a	“>>”	Creates file (if necessary), Appends to the File
r+	“+<“	Reads and Writes but does NOT Create
w+	“+>”	Creates file (if necessary), Clears, Reads and Writes
a+	“+>>”	Creates file (if necessary), Reads and Appends

File Test Operators

- File Test Operators in Perl are the logical operators which return True or False values.
- There are many operators in Perl that you can use to test various different aspects of a file.
- For example, to check for the existence of a file -e operator is used. Or, it can be checked if a file can be written to before performing the append operation.
- Filename or filehandle is passed as an argument to this file test operator -e.

N-07-check-exist.pl

File Test Operators (Cont’d)

- Following is a list of most important File Test Operators:

OPERATOR	DESCRIPTION
-r	checks if the file is readable
-w	checks if the file is writable
-x	checks if the file is executable
-e	checks if the file exists
-z	checks if the file is empty
-d	checks if the file is a directory
-l	checks if the file is a symbolic link

File and Directory Manipulation

File and Directory Manipulation

- Perl provide a goodly number of functions for manipulating directories and files.
- Often using these functions eliminate the need to use the system function to invoke an external process; in addition, these functions are simply more convenient in many cases.

File and Directory Manipulation Functions

Name	Use	Arguments
chdir	Change current directory	scalar directory name
chmod	Change permissions on files	scalar(mode), list of filenames
chown	Change ownership of files	scalar(uid),scalar(gid),list of files
link	Create hard link to a file	scalar(old file),scalar(newfile)
mkdir	Create a new directory	scalar(filename), scalar(mode)
rename	Rename a file	scalar(oldname),scalar(newname)
rmdir	Remove a directory	scalar(filename)
umask	Display or set umask	scalar(umask)
unlink	Remove a file	list of filenames