eXtensible Markup Language

- Jalpa Poriya

INTRODUCTION TO XML

XML stands for eXtensible Markup Language.

XML was designed to store and **transport data**.

XML was designed to be both **human-** and **machine-readable**

XML was designed to be self-descriptive

Rules for XML

Rule 1 - XML Documents Must Have a Root Element

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Rule 2 – programmer can set the XML Prolog. The XML prolog is optional. If it exists, it must come first in the document.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Rule 3 –

All XML Elements Must Have a Closing Tag In XML, it is illegal to omit the closing tag. All elements must have a closing tag

Rule 4 –

XML Tags are Case Sensitive The tag <Letter> is different from the tag <letter>.

Rule 5 –

XML Elements Must be Properly Nested all elements must be properly nested within each other

Rule 6 –

XML Attribute Values Must Always be Quoted

DOCUMENT STRUCTURE

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tonny</to>
  <from>John</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML - ATTRIBUTES

This chapter describes the **XML attributes**. Attributes are part of XML elements. An element can have multiple unique attributes.

Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a name-value pair.

SYNTAX

<element-name attribute1 attribute2 >
....content..
< /element-name>

EXAMPLE

<garden>

<plants category = "flowers" />

<plants category = “tree"> </plants>

</garden>

THE DIFFERENCE BETWEEN XML AND HTML

XML was designed to carry data - with focus on what data is

HTML was designed to display data - with focus on how data looks

XML tags are not predefined like HTML tags are

UNIT IV

XML DTD **and** Schemas

ELEMENT ATTRIBUTE RULES

An attribute name must not appear more than once in the same start-tag or empty-element tag.

An attribute must be declared in the Document Type Definition (DTD) using an Attribute-List Declaration.

Attribute values must not contain direct or indirect entity references to external entities.

The replacement text of any entity referred to directly or indirectly in an attribute value must not contain a less than sign (<)

SYNTAX

<element-name attribute1 attribute2 > ....content..

< /element-name>

EXAMPLE

<garden>

<plants category = "flowers" />

<plants category = “tree"> </plants>

</garden>

THE DIFFERENCE BETWEEN XML AND HTML

XML was designed to carry data - with focus on what data is

HTML was designed to display data - with focus on how data looks

XML tags are not predefined like HTML tags are

DOCUMENT TYPE DEFINITIONS

A DTD is a Document Type Definition.

A DTD defines the structure and the legal elements and attributes of an XML document.

SYNTAX

Basic syntax of a DTD is as follows −

```
<!DOCTYPE element DTD identifier
[ declaration1 declaration2 ........
]>
```

INTERNAL DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files.

This means, the declaration works independent of an external source.

SYNTAX

Following is the syntax of internal DTD −

```
<!DOCTYPE root-element
[element-declarations]
>
```

DTD Body – The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations.

Example:

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone_no (#PCDATA)>

AN INTERNAL DTD DECLARATION

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

XML - SCHEMAS

XML Schema is commonly known as **XML Schema Definition (XSD)**. It is used to describe and validate the structure and the content of XML data.

XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

DEFINITION TYPES

xs:integer

xs:boolean

xs:string

xs:date

Example : <xs:element name = "phone_number" type = "xs:int" />

Simple Type

```
<xs:element name = "phone_number" type = "xs:int" />
```

Complex Type

```
<xs:element name = "Address">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "name" type = "xs:string" />
      <xs:element name = "company" type = "xs:string" />
      <xs:element name = "phone" type = "xs:int" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML - NAMESPACES

A **Namespace** is a set of unique names. Namespace is a mechanisms by which element and attribute name can be assigned to a group.

NAMESPACE DECLARATION

A Namespace is declared using reserved attributes. Such an attribute name must either be **xmlns** or begin with **xmlns:** shown as below -

<element xmlns:name = "URL">

Syntax

The Namespace starts with the keyword **xmlns**.

The word **name** is the Namespace prefix.

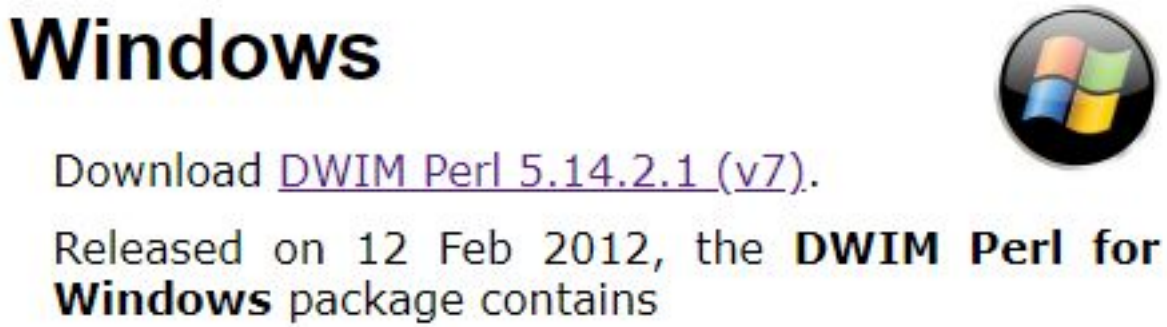
The **URL** is the Namespace identifier.

CGI and Perl

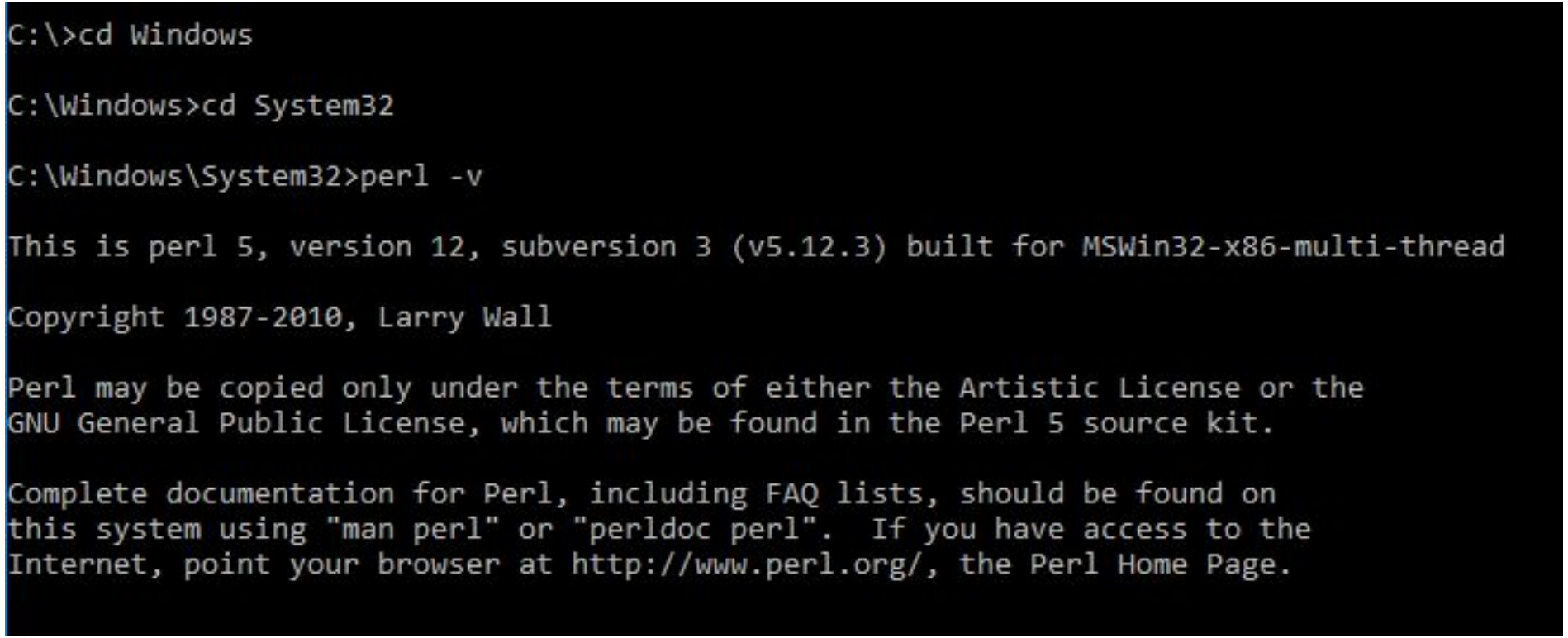
Unit 4

How to install

- http://padre.perlide.org/download.html



- After installing software open command prompt.



- Add plug-in in notepad++
- Open notepad++
- Select Run->Run
- cmd /k C:\Dwimperl\perl\bin\perl.exe "$(FULL_CURRENT_PATH)“
- Create shortcut for this
- Perl file:

```
sub Test{
print "Running";
}
Test();
```

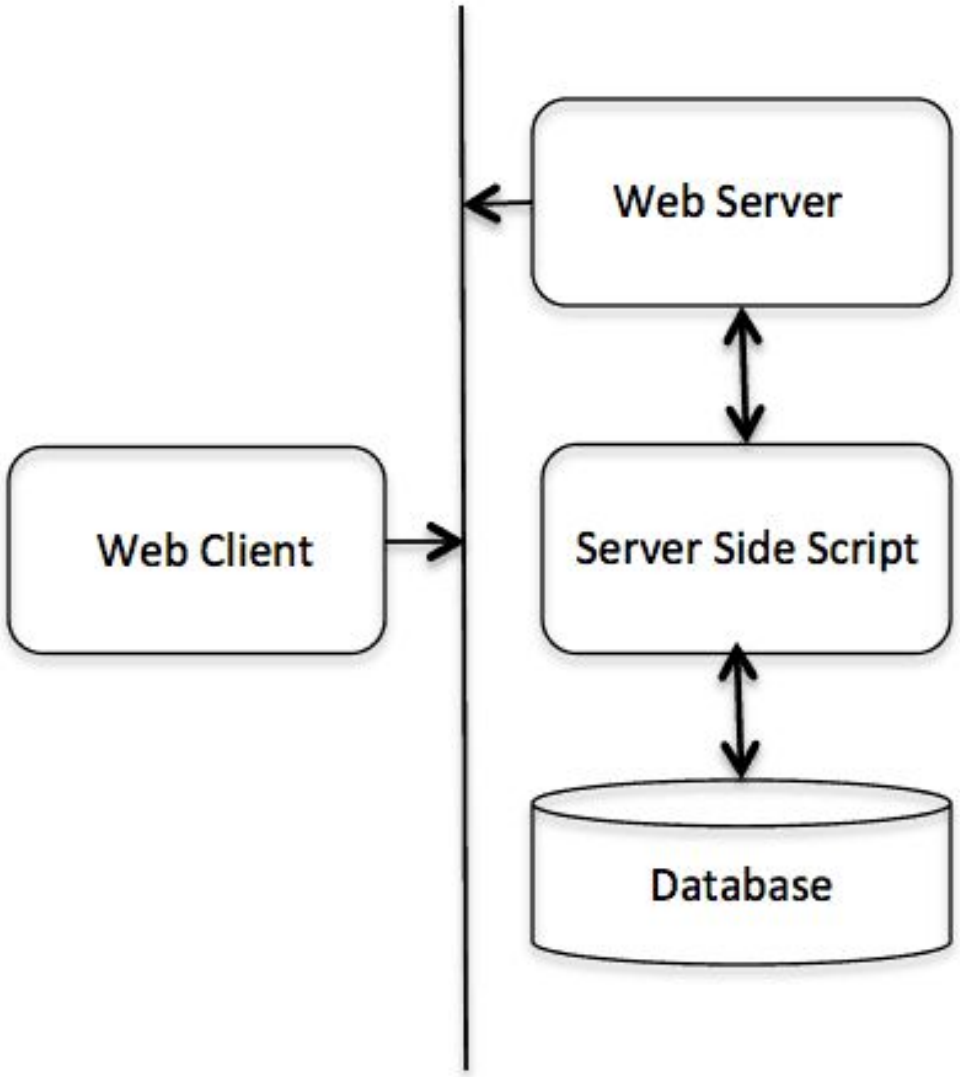
PERL and CGI (**CGI Concepts)**

- The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script.
- *“The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.”*

How it works?

- To understand the concept of CGI, lets see what happens when we click a hyper link to browse a particular web page or URL.
    1. Your browser contacts the HTTP web server and demand for the URL ie. filename.
    2. Web Server will parse the URL and will look for the filename in if it finds that file then sends back to the browser otherwise sends an error message indicating that you have requested a wrong file.
    3. Web browser takes response from web server and displays either the received file or error message.
- However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your browser to display.
- This function is called the Common Gateway Interface or CGI and the programs are called CGI scripts.

CGI Architecture Diagram



The PERL Language (Introduction)

- Perl is a stable, cross platform programming language.
- Though Perl is not officially an acronym but few people used it as **Practical Extraction and Report Language**.
- It is used for mission critical projects in the public and private sectors.

Perl Features

- Perl takes the best features from other languages, such as C, BASIC, among others.
- Perls database integration interface DBI supports third-party databases including Oracle, Sybase, MySQL and others.
- Perl works with HTML, XML, and other mark-up languages.

PERL Basics

- Perl - Syntax Overview
- Perl File Extension (.pl)
- Comments in Perl (#) Example : # This is a comment in perl
- Perl - Data Types

Perl - Syntax Overview

- A Perl program consists of a sequence of declarations and statements, which run from the top to the bottom.
- Loops, subroutines, and other control structures allow you to jump around within the code.
- Every simple statement must end with a semicolon (;).

Perl - Data Types

- Perl is a loosely typed language and there is no need to specify a type for your data while using in your program.
- Different types of variables :
  - Scalar variables
  - Array variables
  - Hash variables

- **Scalar :** Scalars are simple variables. They are preceded by a dollar sign ($).
- A scalar is either a number, a string, or a reference. A reference is actually an address of a variable.
- Example :
- $age = 25; # An integer assignment
- $name = "John Paul"; # A string
- $salary = 1445.50; # A floating point

- **Arrays :** Arrays are ordered lists of scalars that you access with a numeric index, which starts with 0.
- They are preceded by an "at" sign (@).
- Example:
- @ages = (25, 30, 40);
- @names = ("John Paul", "Lisa", "Kumar");

- **Hashes :** Hashes are unordered sets of key/value pairs that you access using the keys as subscripts. They are preceded by a percent sign (%).
- %data = ('John Paul', 45, 'Lisa', 30, 'Kumar', 40);

Example

```
print “Content-type:text/html \n\n”; #HTTP HEADER

$str1 = “Hello!!!”;
#String literal Demo with the use of single quote, double quote, $, \n and
<>
print “<H1>”;
print $str1;
Print “<br></h1>”;
$no1 = 10;
$no2 = 20;
$no3 = $no1 + $no2;
Print $no3;
```

Loops

- **while loop**
- Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

Syntax:

```
While(condition)
{
        Statements
        Increment or decrement operations;
}
```

- **until loop**
- Repeats a statement or group of statements until a given condition becomes true. It tests the condition before executing the loop body.

Syntax :

```
until(condition)
{
        statement(s);
}
```

- **for loop**
- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

```
for ( init; condition; increment )
{
    statement(s);
}
```

- **foreach loop**
- The foreach loop iterates over a normal list value and sets the variable VAR to be each element of the list in turn.
- Syntax:

```
foreach var (list)
{
...
}
```

- **do...while loop**
- Like a while statement, except that it tests the condition at the end of the loop body

```
do {
        statement(s);
}while( condition );
```

https://paiza.io/projects/ZweARBDPVWn6I0YbKFFjaw?language=perl

Example

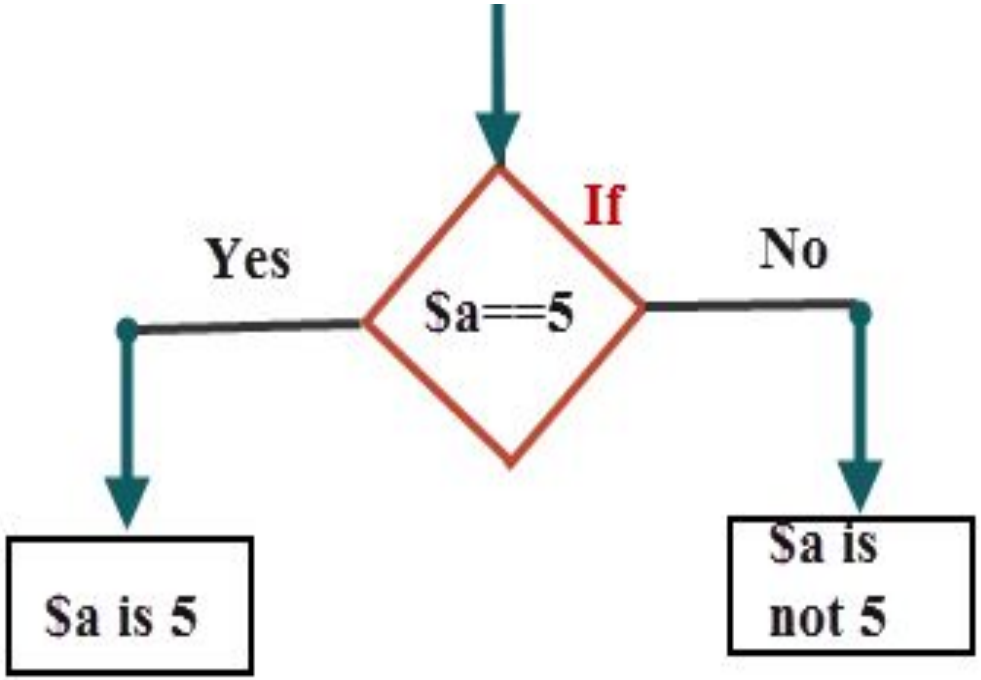
```
print "<h1>While Demo</h1>";
print"<table border=1>";
$count = 1;
$a = 5;
while ($count <= 10)
{
    $ans =  $a*$count;
    print "<tr><td> $a </td><td> * </td><td> $count
    </td><td> = </td><td> ".$ans;
    $count++;
```

Perl Conditional Statements

- We can use conditional Statements in Perl. So, what are conditional statements? Conditional statements are those, where you actually check for some circumstances to be satisfied in your code.
- Think about an example, you are buying some fruits, and you don't like the price to be more than 100 bucks. So, the rule here is 100 bucks.
- Perl supports two types of conditional statements; they are if and unless.

Perl If

- If code block will be executed, when the condition is true.



Array and Array functions

- **Arrays :** Arrays are ordered lists of scalars that you access with a numeric index, which starts with 0.
- They are preceded by an "at" sign (@).
- Example:
- @ages = (25, 30, 40);
- @names = ("John Paul", "Lisa", "Kumar");

Array function in perl

- **Push**: adds array element at the end of an existing array.

Example: Push(*array,value*)

- **Pop**: removes the last element from an array.

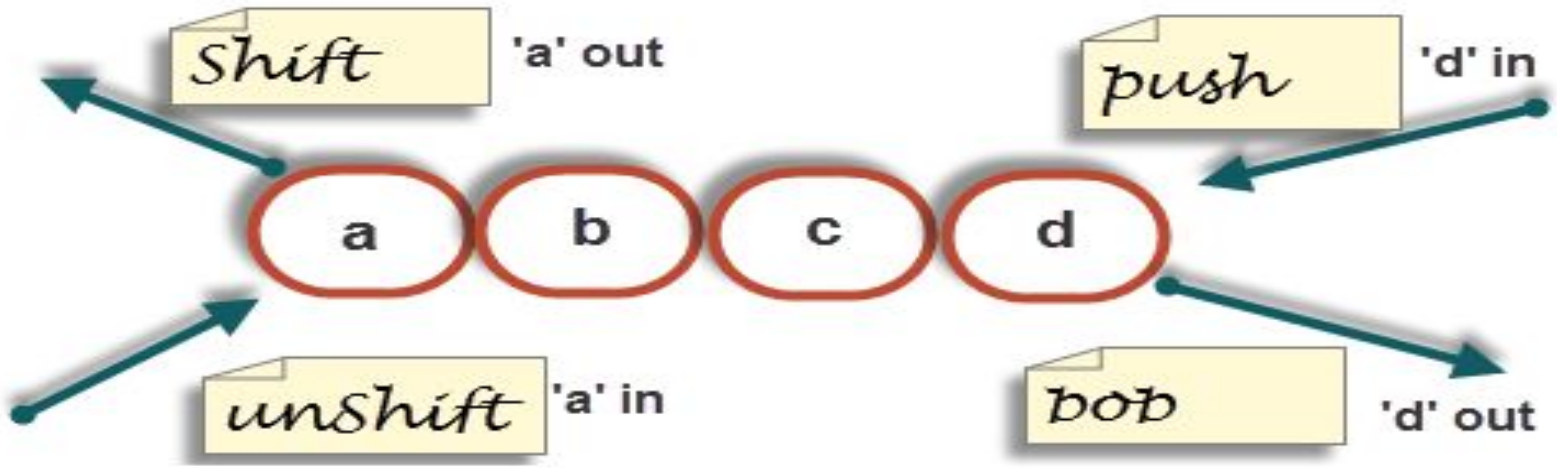
Example: Pop(*array*)

- **Unshift**: adds an element at the beginning of an array.

Example: Unshift(*array,value*)

- **Shift**: removes the first element from an array.

Example: Shift(*array)*



```
@days = ("Mon","Tue","Wed");
print "1st : @days\n";
push(@days, "Thu");    # adds one element at the end of an array
print "2nd when push : @days\n";
unshift(@days, "Fri");        # adds one element at the beginning of an array
print "3rd when unshift : @days\n";
pop(@days);
print "4th when pop : @days\n";   # remove one element from the last of an array.
shift(@days);   # remove one element from the beginning of an array.
print "5th when shift : @days\n";
```

String functions

Function	Description	Syntax	Example
lc	For converting the uppercase letters to lower case letters	lc(string)	$str = “INDUS”; lc($str)
uc	For converting the lowertcase letters to upper case letters	uc(string)	uc($str)
length	Returns the length of the string	length(string)	print length($str)

Function	Description	Syntax	Example
substr	finding the substring	substr(string, index)	$str = “indus university”; substr($str,6); output : university
join	returns the concatenated string	join(string1,string2)	$str1 = “Indus”; $str2 = “University “; join($str1,$str2)

- **Current Date and Time: localtime() function**
- Which returns values for the current date and time if given no arguments.

- Following is the 9-element list returned by the **localtime** function while using in list context –
- sec, # seconds of minutes from 0 to 59
- min, # minutes of hour from 0 to 59
- hour, # hours of day from 0 to 24
- mday, # day of month from 1 to 31
- mon, # month of year from 0 to 11
- year, # year since 1900
- wday, # days since sunday
- yday, # days since January 1st
- isdst # hours of daylight savings time

- Example:
- ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime();

- print “$mday $mon $year";

Formating Date and Time

- You can use localtime() function to get a list of 9-elements and later you can use the **printf()** function to format date and time based on your requirements as follows –
- Example:
- ($sec,$min,$hour) = localtime();
- printf("Time Format - HH:MM:SS\n");
  printf("%02d:%02d:%02d", $hour, $min, $sec);

Math functions

- **rand :** retval = rand (num);
- Example : print rand(10);
- Output : 5
- **Sqrt :** retval = sqrt (value);
- Example : print sqrt(5);
- Output : 2.23606797749979
- **abs :** retval = abs (value);
- Example : print abs(-10);
- Output : 10

- **Cosine** :
- retval = cos (value);
- Example : print cos($no);
- Output : 0.89200486978816
- **Sine** :
- Syntax: retval = sin (value);
- Example : $no = 101; print sin($no);
- Output : 0.452025787178351

File IO in PERL

Perl Create or Open File (Method)

- **open <FILE> , <Entity> , <FILENAME>**
- **Entities :**
- **< or r -** Read Only Access
- **> or w -** Creates, Writes, and Truncates (cut)
- **>> or a -** Writes, Appends, and Creates
- **+< or r+-** Reads and Writes
- **+> or w+** Reads, Writes, Creates, and Truncates
- **+>> or a+** Reads, Writes, Appends, and Creates

- **(<) Syntax**
- The < sign is used to open an already existing file. It opens the file in read mode.
- open FILE, "<", "fileName.txt" **or die** $!
- **(>) Syntax**
- The > sign is used to open and create the file if it doesn't exists. It opens the file in write mode.
- open FILE, ">", "fileName.txt" **or die** $!

- The "<" sign will empty the file before opening it. It will clear all your data of that file. To prevent this use (+) sign before ">" or "<" characters.
- **(+> , +<) Syntax**
- open FILE, "+<", "fileName.txt" **or die** $!
- open FILE, "+>", "fileName.txt" **or die** $!
- **(>>) Syntax**
- The >> sign is used to read and append the file content. It places the file pointer at the end of the file where you can append the information. Here also, to read from this file, you need to put (+) sign before ">>" sign.
- open FILE, “>>", "fileName.txt" **or die** $!
- open FILE, “+>>", "fileName.txt" **or die** $!

Close file function

- To close a filehandle, and therefore disassociate the filehandle from the corresponding file, you use the **close** function. This flushes the filehandle's buffers and closes the system's file descriptor.
- Syntax :
- close(DATA) || die "Couldn't close file properly";

File Writing

- open(fp, '>', 'report.txt');
- print fp "My first report generated by perl\n";
- close fp;
- print "done\n";

CREATING STRUCTURED PROGRAMS, OBJECTS, DATABASE CONNECTIVITY

SUBROUTINES

- A Perl subroutine or function is a group of statements that together performs a task.
- You can divide up your code into separate subroutines.
- How you divide up your code among different subroutines is up to you, but logically the division usually is so each function performs a specific task.

Define and Call a Subroutine

- The general form of a subroutine definition in Perl programming language is as follows −
- sub subroutine_name {
- body of the subroutine
- }

Example:

```
# Function definition
sub Hello {
   print "Hello, World!\n";
}

# Function call
Hello();
```

PASSING ARGUMENTS TO A SUBROUTINE

- You can pass various arguments to a subroutine like you do in any other programming language and they can be accessed inside the function using the special array @_.
- Thus the first argument to the function is in $_[0], the second is in $_[1], and so on.

```
# Function definition
sub Average {
   # get total number of arguments passed.
   $n = scalar(@_);
   $sum = 0;
   foreach $item (@_) {
      $sum += $item;
   }
   $average = $sum / $n;

   print "Average for the given numbers : $average\n";
}
# Function call
Average(10, 20, 30);
```

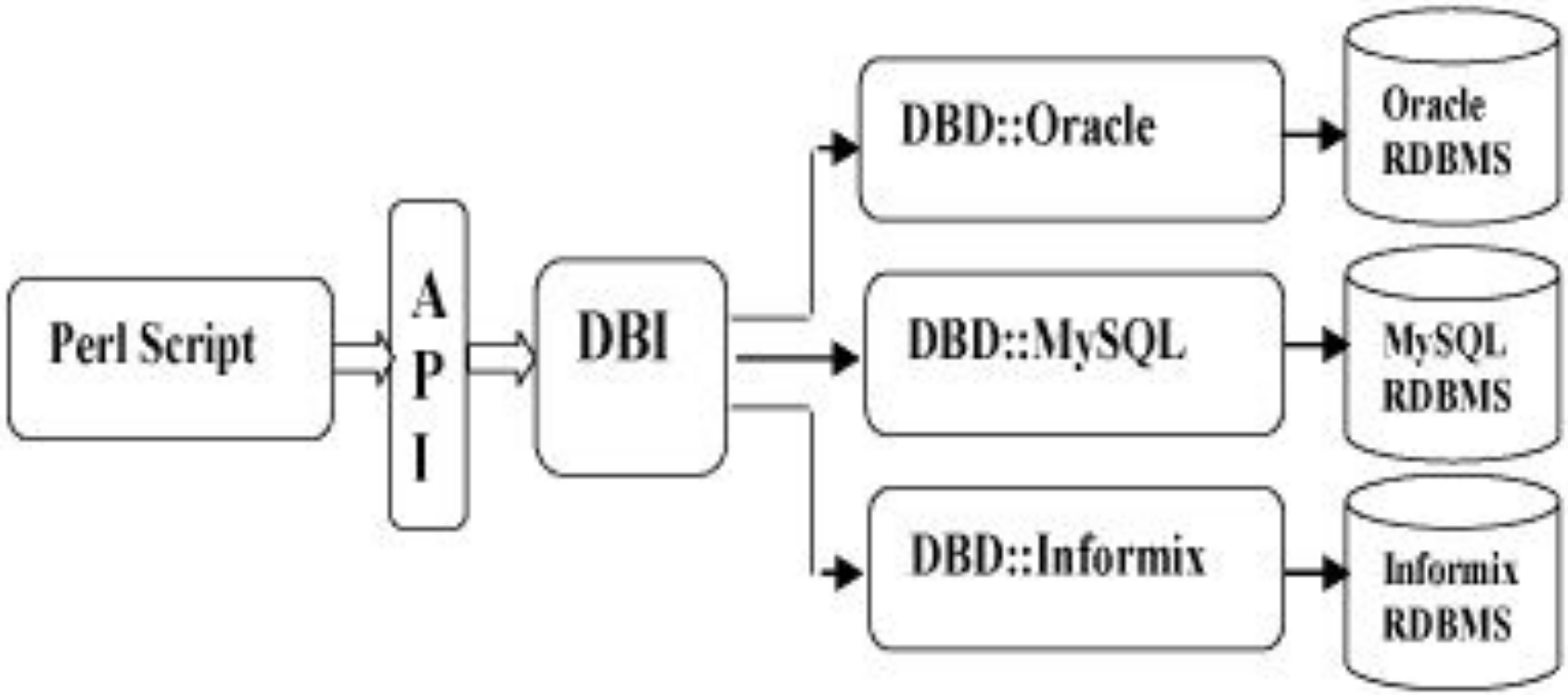
USING REFERENCES :

- Using the backslash operator is analogous to using the ampersand (&) operator in C to pass the address of an operator.
- Usually, you use the backslash operator to create a second, new reference to a variable.
- The following code shows how to create a reference to a scalar variable:
- $variable = 22;
- $pointer = \$variable;
- $ice = "jello" ;
- $iceptr = \$ice;

USING THE ARROW OPERATOR TO DEFERENCE

- The arrow operator (->) is actually a way to deference a reference to an array or a hash.
- Example:
- $aref->[42] # an array dereference
- $href->{"corned beef"} # a hash dereference
- $sref->(1,2,3) # a subroutine dereference

DATABASE CONNECTIVITY
ARCHITECTURE OF A DBI APPLICATION



METHODS

- connect(dsn, id, password)
- result = prepare(query)
- execute() //to execute insert, update, delete, select query
- fetchrow_array() //read select date
- finish() //query execution complete
- commit() //save process in database

METHODS FOR DATABASE CONNECTIVITY

```
$driver = "mysql";
$database = “EMP";
$dsn = "DBI:$driver:database=$database";
$userid = "testuser";
$password = "test123";
$dbh = DBI->connect($dsn, $userid, $password ) or die $DBI::errstr;
```

CRUD IN DATABASE (INSERT)

```
$st = $dbh->prepare("INSERT INTO TEST_TABLE
                    (FIRST_NAME, LAST_NAME, GENDER, AGE, INCOME )
                      values
                    ('MR ABC', 'AAA', 'M', 30, 13000)");
$st->execute() or die $DBI::errstr;
$st->finish();
$dbh->commit() or die $DBI::errstr;
```

CGI.PM

Introduction of CGI.pm

- The CGI.pm module has become the standard tool for creating CGI scripts in Perl.
- It provides a simple interface for most of the common CGI tasks.
- It helps to create HTML page dynamically.

Methods and dynamic web page

```
use CGI;
my $cgi = CGI->new;
print $cgi->header('text/html');
<html>
<body>
     <form method="post">
          Name: <input type="text" name="name"  /><br />
     <input type="submit" name="Submit!" value="Submit!" />
          </form>
 EndOfHTML
if ( my $name = $cgi->param('name') ) {
   print "Your name is $name.<br />";
}
print '</body></html>';
```