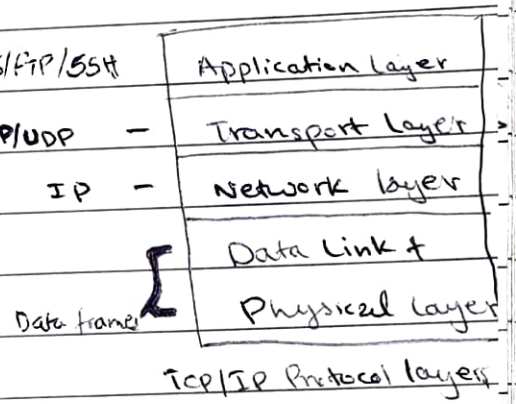
CS481 : Web Technology

16/08/2022

* Protocols: //wikipedia-
  - → IP
  - → TCP
  - → UDP
  - → HTTP
  - → HTTPS
  - → SSH
  - → Certificates.

* PHP:
* ISP / Serlets?
* Javascript : nodejs

- Data frames
- TCP aka Stream Service



TCP/IP Protocol layers

* Web Security
  - Public Key Cryptography.
  - Trusted Third Party
  - Clarification Authority

- LAMP : linux Apache (HTTP server) MySQL (RDBMS) PHP (Portable Hypertext Preprocessor (SSI = server side Include))
- XAMPP : Software for PHP (windows)

CGI : Common Gateway Interface

__/__/____

PHP Basics :

Variables : Preceded by $ sign.

```
$var = 10;
```

Comments : // , # → single line comments

/* */ → Block comments.

```
$x = "message";
echo $x;
$x = 10;
echo "Value of x is". $x
echo "<h1> Value of x is ". $x ."</h1>";
```

Data Types :

Simple : integer, float, string, boolean

Compound : Arrays, Objects,

Special : Resource, Null

To know type of a variable; gettype(value)

is_int(value), is_float(value) ...

Type Casting :

Operators and Expressions :

Arithmetic → +, -, *, /, %.

Assignment → =

Bitwise → &, |, ^ (XOR), ~ (Not), <<, >>

Comparision → ==, !=, <>, ===, !==, <, >, <=, >=

String → . , .= Concatination.

Logical → &&, ||, !, and, or, xor

Compound → +=, -=, *=, /=

28/08/2022

Constants:

```
define ("pi", 3.1415)
$s = 2 * pi * $r;
```

*

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

dtd → document type descriptor

Seq	Value
f_1	0
f_2	1
f_2	1
f_4	2
f_5	3

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
    <title> Fibonacci Series </title>
    <link rel="stylesheet" type="text/css" href="common.css"/>
    <style type="text/css">
        th {text-align: left; background-color: #999;}
        th, td {padding: 0.4em}
        tr.alt td {background: #ddd}
    </style>
</head>
<body>
    <h2> Fibonacci Series</h2>
    <table cellspacing="0" style="width: 20em;
        border: 1px solid #bbb; }
        <tr>
            <th> Sequence </th>
            <th> Value </th>
        </tr>
        <tr><td>f<sub>0</sub></td>
        </tr><td> 0 </td>
        <tr class="alt">
            <td>f<sub>1</sub></td>
            <td> 1 </td>
        </tr>
</body>
</html>
```

__/__/____

```
<?php
    $iterations = 10
    $num1 = 0;
    $num2 = 1;
    for ($i=2; $i<=$iterations; $i++){
            $sum = $num1 + $num2;
            $num1 = $num2;
            $num2 = $sum;
    }
?>
<tr <?php if($i%2!=0) echo "class='alt'" ?>>
<td>f<sub><?php echo $i ?></sub></td>
<td><?php echo $sum ?></td>
</tr>
<?php } ?>
</table>
</body>
```

30/08/2022

* PHP Strings:
  - strlen() : finding string length.
  - indexing, substr() : Accessing characters / substrings.
  - strstr() : searching for substring.
  - strpos(), strrpos() : Locating text
  - substr_count() : finding number of occurances.
  - strpbrk() : Searching for a set of characters.
  - Replacing text:
    - str_replace() : Replaces all occurances.
    - ~~substr_str~~ substr_replace() : Replaces a specified position.
    - strtr() : Replaces characters.

__/__/____

psneog.wordpress.com.

```
$var = 10;
echo 'Value of variable = $var';
```

Displaying a paragraph:

```
$poem << POEM-DELIM        (delimiter (user-defined name))

    poem poem poem poem ....

    ---  -  -  -  -  -  -

POEM-DELIM;

echo $poem;
```

```
$msg = "Hello world";
$len = strlen ($msg);
echo $len;

$chr = $msg[3];
$excl = '!';
echo $msg. $exel;                    // . joins the strings.

$sub = substr($msg, 0, 5);           // 'Hello'
$sub = substr($msg, 6);              // 'world'
$sub = strstr($msg,'wor');           // 'world'
$sub = strstr($msg,'xyz');           // false
$pos = strpos($msg,'wor');           // 7
$pos = strpos($msg,'ld',7);          // 8
$pos = strripos($msg,                // Reverse order searching.

$msg = "It was the best of times, it was the worst of times,"

$count = substr_count('times')       // 2
$sub = strpbrk($msg,'abc')           //
```

* Case Conversion:

strtoupper(); strtolower(); ucfirst(); lcfirst(); ucword(); lcwords();

- strstr(); strpos(); str_replace(); → Case sensitive
- stristr(); stripos(); str_ireplace(); → Case insensitive.

- C style formatted I/O: printf(), sprintf(), fprintf();
- Trimming: trim(), ltrim(), rtrim();
- Padding: str_pad();
- Text wrapping → word_wrap();
- Formatting numbers → number_format()

\t \n %d %f

05/09/2022

* PHP Arrays:

Two kinds:

(i) Indexed Arrays

(ii) Associative Arrays (Hash/Maps)

- Create Array: array()
- Create Indexed Array:

```
$string_array = array("Harry", "Hermione", "Ron");
$int_array = array (10, 20, 30);
echo $string_array[1];          // Hermione
echo $int_array[2];             // 30
$i = 0;
echo $int_array[$i];            // 10
```

- Associative Arrays:

```
                                  key          value
$author_book = array("Rushdie" => "Satanic Verses",
                     "Valmiki" => "Ramayana",
                     "GRR Martin" => "A Game of Thrones");
echo $author_book[1];           // Ramayana
echo $author_book["Valmiki"];   // Ramayana
$author_book["GRR Martin"] = "A Song of Fire (Ice) and Ice (Fire)";
```

- Outputting (Displaying) an entire Array: print_r()

```
$ print_r ($author_book);
```

```
$array = array(10,20,30);
$array[3] = 40;  $array[] = 50;
echo print_r($array);           // 10, 20, 30, 40, 50
```

```
$array[] = 10;
$array[] = 20;
$array[] = 30;
$array[] = 40;
$array[] = 50;
```

__/__/____

```
$book = array();
$book("Title") = "A Wrath of Grapes";
$book("Author") = "J. Stainback";
$bak("Year") = "1939";
print_r($book);          // Title ⇒ A wrath of Grapes
                         Author ⇒ J. Stainback
                         Year ⇒ 1939
```

- Extracting a Range of Elements : array_slice()

  Positions, NOT indexes

```
$book_slice = array_slice($book, 2, 3)
print_r($book_slice);    // Author ⇒ J. Stainback
                         Year ⇒ 1939
```

- Counting Elements of an Array: count()
- Stepping through an Array: current(); key(); next(); prev(); end(); reset(); each();

```
$book = array("Sobha Dey" ⇒ "Swabhiman",
              "Rowlings" ⇒ "Harry Potter",
              "Dickens" ⇒ "Great Expectations",
              "Asimor" ⇒ "I, Robot");
echo current($book);     // "Swabhiman"
echo key($book);         // "Sobha Dey"
next($book);
echo current($book);     // "Harry Potter"
$elem = each($book);     // "Rowlings" ⇒ "Harry Potter"
echo $elem[0]; echo $elem["key"];   // Rowlings
echo $elem[1]; echo $elem["value"]; // Harry Potter
```

07/09/2022

- counter, key(), next(), prev(), end(), reset(), each()

```
$movies = array("Marlen Brando" ⇒ "The Godfather",
                "John Trevolta" ⇒ "face off",
                "Harrison Ford" ⇒ "Indiana Jones");
$elem = current($movies);     // "The Godfather"
$k = key($movies);            // "Marlen Brando"
```

To iterate through the array:

1-
```
while (current ($movies)) {
        $k = key ($movies);
        $u = current ($movies);
        echo "Key = ".k."Value = ".$u;
        next ($movies);
}
```

2.
```
$len = count ($movies);
for ($i=0; $i<$len; $i++) {
        $k = key ($movies[$i]);
        $u = current ($movies[$i]);
        echo "key = ".$k."Value = ".$u;
}
```

-
```
$elements = each ($movies);
echo $element[0];          // Marlon Brando
echo $element["key"];      // Marlon Brando
echo $element[1];          // The Godfather.
```

3. foreach:

Indexed Array	Associative Array.
`foreach ($array as $value){`	`foreach ($array as $key => $value){`
`// use $value here`	`// use $value/$key here`
`}`	`}`

```
foreach ($movies as $key => $value) {
        echo "key = $key, value = $value",
        if ($key = "John Trevolta") {
                $value = "Pulp Fiction";
        }
print_r ($movies);
```

__/__/____

- **Sorting:**
  - (i) sort() / rsort() : Sort Ascending / descending; (Indexed Array)
  - (ii) asort() / arsort() : Sort ASC/DESC (Assoc. Array → by value)
  - (iii) ksort() / krsort() : Sort ASC/DESC (Assoc. Array → by key)
  - (iv) array_multisort() : Multidimensional Arrays.

- **Adding/Removing Array Elements:**
  - (i) array_unshift() : Adds elements at the beginning
  - (ii) array_shift() : Removes elements from the beginning
  - (iii) array_pop() : Removes elements from the end.
  - (iv) array_push() : Adds elements at the end.
  - (v) array_splice() : Add/Remove elements at/from arbitary position

```
array_splice ( $array , m , n , $array_to_add);
```

- `$array` ↑ original array
- `m` ↑ position to add
- `n` ↑ No. of elements to be removed from that position
- `$array_to_add` ↑ new array segment to be added.

```
array_unshift ($movies, "Vlivian Lay", "Gone with the winds");
print_r ($movies);
array_shift ($movies);
print_r ($movies);
```

- Merging Arrays : array_merge()
- Converting/Revering Arrays to/from strings: explode(); implode(); // Self Study.
- Converting Arrays to list of variables : list()

Array Assignment

08/09/2022

1912160
Subhajit Ghimire

- array_splice() :

```
<?php
    $books1 = array ( "Tolkien" => "LOTR",
                      "Paolini" => "Eragon",
                      "Rowling" => "Harry Potter",
                      "Riordan" => "Percy Jackson");
    $books2 = array ( "Dan Brown" => "Da Vinci Code",
                      "Mario Puzo" => "The Godfather");
    $books_spliced = array_splice ($books1, 0, 2, $books2);
    print_r ( $books_spliced);
    // Array ( [Tolkien] => LOTR  [Paolini] => Eragon )
    echo "<br>";
    print_r ( $books1 );
    // Array( [0] => Da Vinci Code  [1] => The Godfather
              [Rowling] => Harry Potter [Riordan] => Percy Jackson
?>
```

- implode() :

```
<? php
    $subjects = array ('Math', 'Science', 'English');
    echo implode(" ", $subjects)."<br>";
    // Math Science English
    echo implode ("+", $subjects);
    // Math + Science + English
?>
```

- explode() :

```
<? php
    $subject = "Math, Science, English";
    print_r (explode (',', $subject , 1));
    // Array ( [0] => "Math, Science, English)
```

__/__/____

```
echo "<br>";
print_r (explode (',', $subject, 2));
// Array ([0] => Math [1] => Science, English);
echo "<br>";
print_r (explode (',', $subject, 10));
// Array ([0] => Math, [1] => Science [2] => English)
?>
```

list().

```
<? php
$movies = array ("Avengers", "Batman", "Top Gun",
                 "Minions", "Uncharted");
list ($m1, $m2, , $m3) = $movies;
echo "Movies $m1 and $m2 are superhero <br";
// Movies Avgengers and Batman are superhero.
echo "Movie $m3 is animated.";
// Movie Minion is animated.
```

13/09/2022

* Class :

```
// car.php
class Car {
    public $color ; public $make; private $speed;
    public function getSpeed () { return $speed; }
    public function accelerate () {
        if ($this->speed >=100) return false;
        $this->speed += 10; return true;
    }
    public function brake () {
        if ($this->speed <=0 ) return false;
        $this->speed -=10; return true;
    }
}
```

```
<html>
  <head> <title> Car demo </title> </head>
  <body>
    <h1> Car demo </h1>
    <?php require_once "car.php";
        $myCar = new Car();
        $myCar->make = "Maruti";
        $myCar->color = "Red";
        echo "I am driving a $myCar->color $myCar->make";
        echo "Stepping on the gas ... <br/> ";
        while ( $myCar->accelerate() ) {
            echo "New Speed = $myCar->getSpeed() ";
        }
    ?>
  </body>
</html>
```

- Static Members :

Hints :

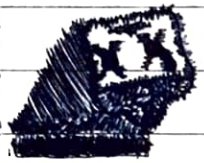
```
function paint ( $myCar , $myColor ) {
    $myCar->color = $myColor;
}
$c = new Color();
paint ( $c, "White");
```

__/__/____

```
$miew = new Car();
paint($miew, "Blue");
// will give error. How to resolve? Redeclare the
```

Previous function paint as follows:

```
function paint(Car $myCar, $myColor){ ... }
```



Inheritance:

```
class Fruit {
        public function eat() {
                echo "I am eating the fruit";
        }
}
class Banana Extends Fruit {
        echo "I am peeling a banana";
} //parent::eat();
```

// final class Fruit { }
↑ no more inheritance allowed.

```
$f = new Fruit();
$f->eat();
$b = new Banana();
$b->eat();
```

- Constructors and Destructors:

```
class MyCar extends Car {
        function __construct($make, $color){
                echo "Constructor for MyClass";
                $this->color = $color;
                $this->make = $make;
        }
}
$car = new MyCar("Hundai","Silver");
get_class($car);  // gives the class of object $car;
```

14/09/2022

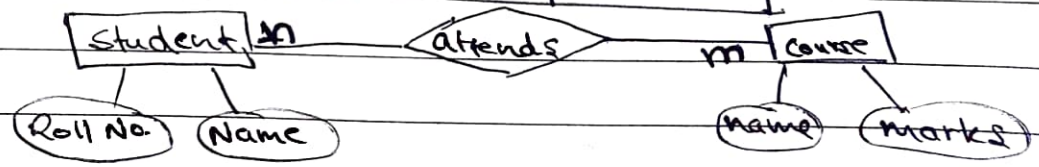
- PHP + MySQL :

LAMP
Light Apache MySQL PHP

- Relational DBMS:

Roll No.	Name	Course	Marks (100)
1.	Harry Potter	Divination	15
1.	Harry Potter	Transfiguration	80
1.	Harry Potter	DADA	100
2.	Ronald Weasley	Divination	10
2.	Ronald Weasley	Transfiguration	70
2.	Ronald Weasley	DADA	75
3.	Hermione Granger	Arithmancy	100
3.	Hermione Granger	Transfiguration	100
3.	Hermione Granger	DADA	90

Redundancy
Ambiguity.



E-R Diagram.

__/__/____

```
$mysql -u root -p
> Create database STUDENT-DB;
> use STUDENT-DB.

CREATE TABLE Student (rollno NUMBER(10) NOT NULL UNIQUE,
                      name VARCHAR(100) NOT NULL,
                      PRIMARY KEY (roll no),
                      REFERENCES course (course code)
                     );
CREATE TABLE course ( Course_code VARCHAR(10) NOT NULL UNIQUE
                      PRIMARY KEY,
                      Course_name VARCHAR(50) NOT NULL
                    );
```

DROP TABLE : Delete table entirely.
ALTER TABLE : Modify columns of table.
DELETE TABLE : Deletes All Row contents from the table ; does not delete table through.
SHOW DATABASE ; SHOW TABLES;

- Add data to a table.

```
INSERT INTO course (course_code, course_name) VALUES
('M001', 'DIVINATION');
INSERT INTO course (- - - - -) VALUES ('M002', 'TRANSFIGURATION')
```

- Delete data from a table:

```
DELETE FROM course WHERE course_code = 'M001'; //CASCADE
```

19/09/2022

Student: | Roll No. | Name | DeptID |

Department: | DeptID | DeptName |

To see list of students belonging to Dept CSE

```
SELECT RollNo, Name, DeptName FROM STUDENT, DEPARTMENT
WHERE Student.DeptID = Department.DeptID AND
DeptName = "CSE";
```

Grouped by DeptName

```
SELECT RollNo, Name, DeptName FROM Student, Department where Student.
deptID = Department.deptID GROUP BY DeptName;
```

```
$mysql -u root -p
> CREATE DATABASE MyDb;
> USE MyDb;
> CREATE TABLE fruit ( id SMALLINT UNSIGNED NOT NULL AUTO INCREMENT,
                       name VARCHAR(20) NOT NULL,
                       color VARCHAR(30) NOT NULL,
                       PRIMARY KEY (id) );
```

__/__/____

```
>INSERT INTO Fruit (name, color) VALUES ("Banana", "Yellow");
> INSERT INTO . . . . . ("Orange", "Orange");
> . . . . . ("Plum", "Purple");
```

```
<HTML> <HEAD> <TITLE> Fruits </TITLE> </HEAD>
    <BODY> <?php
        $dsn = "mysql:dbname = MyDb";   // Data Source Name
        $user = "root";
        $password = "";
        $conn = new PDO ($dsn, $user, $password); // create a PHP data obj
        if ($conn == null) { die ($!); }    //exit
        $sql = "SELECT * FROM Fruit";
        $rows = $conn -> query ($sql);   //associative array
        if ('$rows' == null') { die ($!); }
        foreach ($rows as $rows) {
            echo "A. ".$row["name"]."is ".row["color"];
            echo "<br>";
        }
        $conn = null;          //disconnect
    ?> </body>
</HTML>
```

25/10/2022

* XML : eXtensible Markup Language.

Basic Syntax/Format / Layout & Example

Prolog { `<?xml version = "1.0"?>`

Body {

```
<email>
    <to> abc@nits.ac.in </to>
    <from> psneog@gmail.com </from>
    <subject> Happy Diwali </subject>
    <body> Happy Diwali to all </body>
</email>
```

User Defined.

- Two Parts in an XML Document:
    1. Prolog
    2. Body

__/__/____

- Prolog Contains:
  - XML Declaration
  - Operational Processing Instructions (PI)
  - comments
  - Document type declaration (DTD)

XML Declaration:

<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>

Operational Processing Instruction:

<?xml stylesheet href = "simple.xsl" type = "text/xsl" ?>

Comments

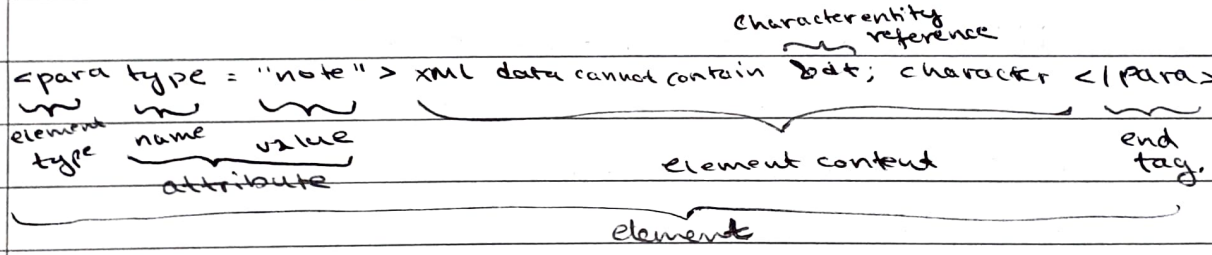
<!-- Comment text here -->

Master Grim DTD

- name of the root element
- reference to an external DTD
- element declarations
- entity Declarations

- Elements (example)

<para type = "note">XML data cannot contain &dt; character </para>



- Naming Rules.
  - can only contain letters, symbols and some other special chars like "_" underscore.
  - can't start with a number or punctuation marks
  - must not contain the string "xml".
  - can't contain white space.

- Well format XML:
  - one and only one root element.
  - all tags must be closed.
  - all tags must be properly nested.
  - tags are case sensitive.
  - attribute values must be quoted.
  - reserved characters: space; < "<", " etc.

SAX parsers: abgd

26 / 10 / 2020

oceanofpdf.com

- Converting XML to HTML:

```
<?xml version = "1.0" ?>
<?xml stylesheet type = "text/xsl" href = "books.xsl" ?>
<bookstore>
    <book category = "literature">
        <title lang = "bengali">Geetanjali </title>
        <author> Rabindranath Tagore </author>
        <year> 1930 </year>
    </book>
    <book category = "fiction">
        <title lang = "en"> Secret of Nagas </title>
        <author> Amish Tripathi </author>
        <year> 2004 </year>
    </book>
</bookstore>
```

- bookstore : book → title, author, year.

My Book Collections.

Title	Author	Year

- book.xsl

```
<?xml version = "1.0" ?>
<xsl:stylesheet version = "1.0" xmlns:xsl = "http://www.....">
```

namespace specification (user defined) — PI — namespace binding — Symbolic URL

```
<xsl:output method = "html" version = "1.0" encoding = "UTF-8"
    indent = "yes" />
<xsl:template match = "/" >
<html>
    <body>
```

31/10/2022

```
<h1> My Book collection </h1>
<table>
    <tr>
        <th> Title </th>
        <th> Author </th>
        <th> Year </th>
    </tr>
    <xsl: for-each select = "bookstore/book" >
    <tr>
        < xsl: for-each select = "./* " >
            <td> <xsl: value-of select = "." /> </td>

        </xsl: for-each >
    </tr>
    </xsl: for-each >
</table>
</body>
</html>
</ xsl: template >
</xsl: stylesheet>
```

* DTD (Document Type Descriptor).
  - defines XML Document Syntax.
  - Internal and External.

Internal

- Internal:

```
<? xml version = "1.0" ? >
<! DOCTYPE greeting [
        <! ELEMENT greeting (#PCDATA) >
] >
<greeting > Hello World! </greeting >
```

• DTD Syntax:

External

- External:

FILENAME: external.dtd

```
<! DOCTYPE greeting [
        <! ELEMENT greeting (#PCDATA)>
] >
```

- External DTD Syntax: (Processing Instruction, PI)

```
<! DOCTYPE root-element
        [ SYSTEM|PUBLIC FPI ] 'URI' >
```

- FILENAME: original.xml

```
<? xml version = "1.0" ? >
<! DOCTYPE greeting SYSTEM 'external.dtd' >
<greeting> Hello </ greeting >
```

root element

```
<?xml version = '1.0' ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD 4.0 HTML Transitional//EN"
        'http://www.w3c.com/TR/REC-html40/loose.dtd' >
<HTML>
   <HEAD> <TITLE> XHTML DOCUMENT </TITLE></HEAD>
   <BODY>
        Hello World! How are you?! Don't Care & Didn't Ask & You're Human!
   </BODY>
</HTML>
```

- Element Type Declaration.

```
<! ELEMENT document-name type >
                  OR
<! ELEMENT document-name (content) >
```

example: `<! ELEMENT greeting (#PC DATA) >`

Parsed Character data: cannot use '<' '>' Alt. use '<' '>'

- Standard Empty Elements:

```
<br/>   :  <br></br>
<hr/>   :  <hr></hr>
```

. `<! ELEMENT element-name EMPTY >`

- Unrestricted type:

`<! ELEMENT photo ANY >`

- Simple Elements: (#PCDATA or #CDATA)

- Compound Elements:

```
<! DOCTYPE employee [
        <!ELEMENT employee(name)>
        <! ELEMENT name (#PCDATA)>
]>
```

≡ equivalent to.

```
<employee>
   <name>
   </name>
</employee>
```

```
<!DOCTYPE organisation [
<!ELEMENT organisation (employee+)>
   <! ELEMENT name (#PCDATA)>
]>
```

occurance indicator one or more.

```
<organisation>
   <employee>...</employee>
   <employee>...</employee>
   <employee>...</employee>
</organisation>
```

```
<organisation>
   <employee>
      <name></name>
   </employee>
   <employee>
      <name></name>
   </employee>
</organisation>
```

* occurance indicators:

let, name of element be e.

- None (only once) : e
- * (zero or more) : e*
- + (one or more) : e+
- ? (zero or once) : e?

01/11/2022

-Compound Element : Declaring multiple children:

```
<!ELEMENT name (fname, mname, lname)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT mname (#PCDATA)>
<!ELEMENT lname (#PCDATA)>
```

```
<name>
  <fname></fname>
  <mname></mname>
  <lname></lname>
</name>
```

```
<!DOCTYPE bookstore [
    <!ELEMENT bookstore (book*)>
    <!ELEMENT book (#PCDATA)>
]>
```

```
<bookstore>
  <book>
```

```
<!DOCTYPE company [
    <!ELEMENT company (employee*, department*)>
    <!ELEMENT employee (#PCDATA)>
    <!ELEMENT department (#PCDATA)>
]>
```

```
company
  employee
  employee
  :
  department
  department
  :
```

- a, b : a followed by b
- a|b : a or b
- (a, b)* : (a followed by b) → zero times or more.
- , : Sequence | : () : grouping
- ? Optional middlename

```
<!ELEMENT employee (fname, mname?, lname)>
<!ELEMENT book (title, author+, chapters*)>
<!ELEMENT order (item+)
        <!ELEMENT item (#PCDATA)|(description)*>
        <!ELEMENT description (#PCDATA)>
<item> xxx </item>
<item>
    <description> yyy </description>
    <description> zzz </description>
</item>
```

__/__/____

Attribute Declaration:

Syntax:

<!ATTLIST element_name attribute_name attribute_type default_value>

Example

<!ELEMENT line EMPTY>
<!ATTLIST line width CDATA '100'>

<line />
<line width='200' />

default_value → Default, #REQUIRED, #IMPLIED, #FIXED value
(#IMPLIED: Optional)

attribute_type → String: CDATA
Tokenised Types: ID, IDREF/IDREFS,

<!ATTLIST question qno ID #REQUIRED>
<!ATTLIST employee eid ID #REQUIRED>

- only one ID per element
- must have either #IMPLIED or #REQUIRED
- first character must be letter, '_', or ':'

<!ATTLIST question qno ID #REQUIRED>
<!ATTLIST answer aqno IDREF #REQUIRED>

<question qno='Q1'>
concat is full form of DTD?
</question>
<answer aqno='Q1'>
Document Type Descriptor
</answer>

<!ATTLIST student roll ID #REQUIRED>
<!ATTLIST subject sid ID #REQUIRED>
<!ATTLIST marks ref IDREF #REQUIRED>

<student id=01>
Harry potter
</student>
<subject sid='10'>
DADA
</subject>
<marks ref='01 10'>
100
</marks>

08/11/2022:

Some Java Programs: (2 page of code).

Attribute Declaration: 02/11/2022

Tokenised Types:

- ID, IDREF/IDREFS, NMTOKEN/NMTOKENS
  ENTITY/ENTITIES

NMTOKEN

```
<!ATTLIST item serialno NMTOKEN #REQUIRED>
<!ATTLIST employee profile NMTOKENS #REQUIRED>
```

ENTITY

```
<!ATTLIST photo src ENTITY #REQUIRED>
e.g.: <photo src = "logo[redacted]">
```

- ```
  <!NOTATION jpg SYSTEM 'image/gif'>
  <!ENTITY logo 'ju.jpg' NDATA gif>
  ```

- Enumeration Types.

e.g.: `<employee gender='male'>`

```
<!ATTLIST employee gender (male|female) #REQUIRED>
```

- Multiple Attributes

```
<!ATTLIST employee
          name CDATA #REQUIRED
          gender (male|female) #IMPLIED
>
```

- Notation Type:

- ENTITY
  - Variables.
  - General and Parameterentities.
  - Internal and External entities.
  - Parsed? and Imposed entities.
    Posed? Unposed?

- General entity declaration:

```
<!ENTITY entity-name value>
```

eg: `<!ENTITY UKR 'Uttan K Ray'>`

```
<author>&UKR;</author>
<book author='&UKR'>Web Technology</book>
```

- External entities.

```
<!ENTITY entity-name PUBLIC|SYSTEM URI>
```

- Unposed entities (external)

```
<!ENTITY entity-name PUBLIC|SYSTEM NDATA notation-name>
<!NOTATION jpeg SYSTEM 'image/jpg'>
```

[redacted]
```

<! ENTITY [redacted] Logo PUBLIC 'http://www.vit.ac.in/images/logo.jpg' NDATA jpeg >

- Parameter entities:

  <!ENTITY % entity_name entity-def >

  e.g.: <!ENTITY % name "fname, name, lname" >

* DTD Validation :

  -xmlint

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
public class DTDValidator {
    Public static void main (String args[]){
        try {
            DocumentBuilderFactory f.DocumentBuilderFactory.
            f.setValidating(true);          newInstance();
            Document d = f.newDocumentBuilder().parse(
                                            args[0]);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

DTDValidator.java

DTDValidator.class

java DTDValidator filename.

09/11/2022

* document.xml → transform.xsl (xml stylesheet) → output.xml

```
<employee>
    <name> P S Neog </name>
    <Sex> Male </Sex>
    <Status> Married </status>
    <Salary basic = '10000' DA = '10000'></Salary>
</employee>
```

⇓

```
<employee profile = 'male married'>
    <name> P S Neog </name>
    <Salary>
        <basic> 100 000 </Salary>
        <DA> 10 000 </DA>
    </Salary>
</employee>
```

(Yesterday, we used JAVA for this transformation which was complicated, today we'll be using XSL??)

```
<?xml version = '1.0' ?>
<xsl:transform version = '1.0' xmlns:xsl = 'http://w3c.org/1999/xsl/Transform' >
<xsl: template match = '/employee' >
<xsl: element name = 'employee' >
<xsl: attribute name = 'profile' select = 'concat (status, " ", sex )'/>
    <name> <xsl: value-of select = 'name' /> </name>
    <salary>
        <basic > <xsl: value-of select = 'salary/@basic' /> </basic>
        <DA > <xsl: value-of select = 'salary/@DA' /> </DA>
    </salary>
</xsl:element>
</xsl: template>
```

__/__/____

* Tastes performed by XSLT:
  - Constant text generation.
  - Reformating of info
  - Information supression.
  - Add new information.
  -

* Style Structure Schema:

http://www.w3c.org/2005/02/schema/for-xslt20.xsd

Structure PI (Processing Instructions)

```
<?xml version='1.0'?>
<xsl:transform version='2.0' xmlns:xsl='http://
                                  www3c.org/1999/XSL/Transform>
    :
<xsl:transform>
```

(xsl:transform = Stylesheet)

* XSLT Elements:

analyze-string	namespace-alias	result-document
call-template	perform-sort	text
copy-of	strip-space	with-param
for-each-group	value-of	apply-templates
include	attribute-set	choose
otherwise	copy	element
processing-instruction	for-each	if
template	import-schema	message
when	number	param
apply-imports	preserve-space	sort
character-map	stylesheet	transform
decimal-format	variable	attribute
function		comment
key		fallback
output		import

15/11/2022

* Template Rules

```
<xsl:template match = 'XPath expression'>
    Instructions
    ...
</xsl:template>
```

Example:

```
<xsl:template match = '/' >
    Hello World!
</xsl:template>
```

Named Templates.

```
<xsl:output method = "html" />
<xsl:template match = '/' >
    <xsl:call-template name = "header" />
</xsl:template>
<xsl:template name = 'header' />
    <title> XSLT </title>
</xsl:template>
```

Web Technology

WEB TECHNOLOGY

* Repetition: (for-each)

```
<store>
  <HDD type = 'SATA'>
    <make> Samsung </make>
    <capacity> 80 </capacity>
    <speed> 7200 </speed>
    <price> 1800 </price>
  </HDD>
  <HDD type = 'SATA'>
    <make> Seagate </make>
    <capacity> 160 </capacity>
    <speed> 7200 </speed>
    <price> 2200 </price>
  </HDD>
  ...
</store>
```

Available HDDs

make	Quantity	Speed	Price
Samsung	80	7200	1800
Seagate	160	7200	2200
...	...	...	...

Desired Table

HTML code

HTML code

WEB.TECH

```
<HTML>
  <BODY>
    <TABLE border="1">
      <caption> Available HDDs </caption>
      <tr>
        <th> Make </th>
        <th> Capacity </th>
        <th> Speed </th>
        <th> Price </th>
      </tr>
      <tr>
        <td> Samsung </td>
        <td> 80 </td>
        <td> 7200 </td>
        <td> 1800 </td>
      </tr>
      <tr>
        <td> Seagate </td>
        <td> 160 </td>
        <td> 7200 </td>
        <td> 2200 </td>
      </tr>
      :
    </TABLE>
  </BODY>
</HTML>
```

- XML Stylesheet for above code ↑

```
<?xml version ... >
<xsl:transform ... >
<xsl:output method = 'html' />
<xsl:template match = '/'>
  <HTML>
  <BODY>
  <TABLE border = '1' >
    <caption >Available HDD </caption>
    <tr>
      <th > Make </th>
      <th> Capacity </th>
      <th> Speed </th>
      <th> Price </th>
    </tr>
    <xsl:for-each select = './/HDD' >
      <tr>
        <td> <xsl:value-of select = 'make' /></td>
        <td> <xsl:value-of select = 'capacity' /></td>
        <td> <xsl:value-of select = 'Speed' /></td>
        <td> <xsl:value-of select = 'Price' /></td>
      </tr>
    </xsl:for-each>
  </TABLE>
  </BODY>
</HTML> </xsl:template>
```

```
<xsl:for-each select = './/HDD' >
  <tr>
    <xsl:for-each select = '.' >
      <td> <xsl:value-of select = '.' /></td>
    </xsl:for-each>
  </tr>
</xsl:for-each>
```

__/__/____

* Conditionals: (if)

```
<xsl:if test='condition'>
    ... do --- something ---
</xsl:if>
```

Example
bookstore.

```
<book title='X'>
:
<book title='Y'>
;
```

```
<template match='/book'>
    <xsl:if test='@title = "Computer Networks"'>
        <xsl:value-of select='@title'/>
        <xsl:value-of select='author'/>
    </xsl:if>
</xsl:template>
```

16/11/2022

* Using Choose: <xsl:choose>

```
<xsl:when test='Expression1'> ... </xsl:when>
<xsl:when test='Expression2'> ... </xsl:when>
:
<xsl:otherwise> ... </xsl:otherwise>
</xsl:choose>
```

Example

```
<xsl:template match='/'>
    <xsl:for-each select='all student'>
        <xsl:value-of select='@roll'/>
            <xsl:text/> &#x20;&#x20;&#x20;&#x20; </xsl:text>
            <xsl:choose>
                <xsl:when test='marks &gt; 80
                and marks &lt;= 100'> A </xsl:when>
                <xsl:when test='marks &gt; 50 and
                marks &lt;= 80'> B </xsl:when>
                <xsl:otherwise> F </xsl:otherwise>
            </xsl:choose>
    </xsl:for-each>
</xsl:template>
```

```
<result>
    <student roll='01'>
        <marks> 80 </marks>
    </student>
    <student roll='02'>
        <marks> 92 </marks>
    </student>
    :
</result>
```

* Variables and Parameters:

```
<xsl:template name='add'>
    <xsl:param name='a' />
    <xsl:param name='b' />
    <xsl:value-of select='$a + $b' />
</xsl:template>
```

To call this →

__/__/____

```
<xsl:call-template name='add'>
    <xsl:with-param name='a' select='10'/>
    <xsl:with-param name='b' select='20'/>
</xsl:call-template>
```

Variable

```
<xsl:variable name='a'>20</xsl:variable>
<xsl:variable name='b'>30</xsl:variable>
```

To use this:

```
<xsl:value-of select='$a+$b'/>
```

Attribute Value Template

```
<article title='{@title}'/>
```

```
<book title='...'>
<xsl:template select='book'>
    <article title='{@title}'>
</xsl:template>
```

Creating element node:

```
<xsl:element name='greeting'>
    Hello world!
</xsl:element>
```

Output

```
<greeting>Hello World!</greeting>
```

```
<employee>
    <name>PS Neog</name>
    <sex>male</sex>
    <status>married</status>
    <salary basic='15000' DA='1000'/>
</employee>
```

```
<employee>
    <name>PS Neog</name>
    <sex>male</sex>
    <status>married</status>
    <salary>
        <basic>15000</basic>
        <DA>1000</DA>
    </salary>
</employee>
```

Sammi Sammiegzat Sammiegzat Sammiegzat Sammiegzat Sammiegzat

__/__/____

```
<xsl:template match='*'>
    <xsl:element name='{name()}'>
        <xsl:for-each select='@*'>
            <xsl:element name='{name()}'>
                <xsl:value-of select='.'/>
            </xsl:element>
        </xsl:for-each>
    </xsl:element>
</xsl:template>
```

→ name of the current element

* Creating Attribute Nodes:

```
<xsl:element name='greeting'>
    <xsl:attribute name='lang'>en</xsl:attribute>
        Hello world!
    </xsl:element>
<greeting lang='en'>Hello world!</greeting>
```