TOPIC 5: JAVA SCRIPT AND ACTIVE SERVER PAGES

T5.1) Describing Java Script and ASP

JavaScript is a programming language commonly used in web development. It was originally developed by Netscape as a means to add dynamic and interactive elements to websites.

JavaScript is a client-side scripting language, which means the source code is processed by the client's web browser rather than on the web server. This means JavaScript functions can run after a webpage has loaded without communicating with the server. For example, a JavaScript function may check a web form before it is submitted to make sure all the required fields have been filled out. The JavaScript code can produce an error message before any information is actually transmitted to the server.

Like server-side scripting languages, such as PHP and ASP, JavaScript code can be inserted anywhere within the HTML of a webpage. However, only the output of server-side code is displayed in the HTML, while JavaScript code remains fully visible in the source of the webpage. It can also be referenced in a separate .JS file, which may also be viewed in a browser.

Below is an example of a basic JavaScript function that adds two numbers. The function is called with the parameters 7 and 11. If the code below were included in the HTML of a webpage, it would display the text "18" in an alert box.

```
<script>
  function sum(a,b)
  {
    return a + b;
  }
  var total = sum(7,11);
  alert(total);
</script>
```

JavaScript functions can be called within <script> tags or when specific events take place. Examples include onClick, onMouseDown, onMouseUp, onKeyDown, onKeyUp, onFocus, onBlur, onSubmit, and many others. While standard JavaScript is still used for performing basic client-side functions, many web developers now prefer to use JavaScript libraries like jQuery to add more advanced dynamic elements to websites.

ASP has two different meanings in the IT world: 1) Application Service Provider, and 2) Active Server Page.

1) Application Service Provider
An Application Service Provider is a company or organization that provides software applications to customers over the Internet. These Internet-based applications are also known

as "software as a service" (SaaS) and are often made available on a subscription basis. This means ASP clients often pay a monthly fee to use the software, rather than purchasing a traditional software license. Some SaaS applications can be accessed via a web browser, while others operate over a proprietary secure port.

2) Active Server Page
An Active Server Page, commonly called an "ASP page," is a webpage that may contain scripts as well as standard HTML. The scripts are processed by an ASP interpreter on the web server each time the page is accessed by a visitor. Since the content of an ASP page can be generated on-the-fly, ASP pages are commonly used for creating dynamic websites.

ASP is similar to other scripting platforms, like PHP and JSP, but supports multiple programming languages. While the default ASP language is VBScript, ASP pages can include other programming languages as well, such as C# and JavaScript. However, alternative languages must be defined before the script code using the following declaration:

<%@ Page Language="C#"%>

ASP pages are part of the ASP.NET web application framework developed by Microsoft. Therefore, ASP pages are most often found on Windows-based web servers that run Microsoft Internet Information Services, or IIS. You can tell if you are accessing an ASP page in your browser if the URL has an ".asp" or ".aspx" suffix.

PHP Stands for "Hypertext Preprocessor." (It is a recursive acronym, if you can understand what that means.) PHP is an HTML-embedded Web scripting language. This means PHP code can be inserted into the HTML of a Web page. When a PHP page is accessed, the PHP code is read or "parsed" by the server the page resides on. The output from the PHP functions on the page are typically returned as HTML code, which can be read by the browser. Because the PHP code is transformed into HTML before the page is loaded, users cannot view the PHP code on a page. This make PHP pages secure enough to access databases and other secure information.

T5.2) Data Input procedures

Input & data entry design

To develop user-friendly interface and process for getting quality data into the information system in a timely and accurate fashion

Documenting Data Entry Procedures

Objectives: generally, most work done on improving the standards of data entry covers the entry of data for operational, financial or administrative applications. These are either

developed for the Department or acquired as Commercial products. This Activity presents some Good Practices for ensuring that all data entry procedures are properly documented.

Scope of usage: generally, this would apply to data entry for operational, administrative and financial application software. However, in some cases, there are data entry procedures that are covered by Office Technology products.

Risks: not having proper data entry procedures would lead to the following:

- Improper entry of data causing errors, rework and possibly damages with citizens and other users of the data.
- Erroneous or slow entry of data leading to loss of time and re-entry
- More rigorous training of new staff

Standard Operating Procedure:

1) For each data entry form in each software application, a detailed Data Entry Procedure is to be prepared. Generally, the software would have such procedures if it is well documented. In such a case, the Department should check if such procedures are sufficient, otherwise, they would be used as the basis for a redeveloped Data Entry Procedure.

   The Data Entry procedure would be used in the following cases:

   - ✓ Training
   - ✓ Day to day entry tasks
   - ✓ Validation and consistency checking
   - ✓ Auditing purposes
   - ✓ Refreshing the user

   A Sample Data Entry Procedure is presented in the **Appendix of Supplementary Material**.

2) On preparing a Data Entry Procedure, it should be tested to ensure that all data elements are clearly explained and that the procedure is easy to understand and use.

3) Whenever the Department is developing its own systems or requesting such systems to be developed by third parties, it is essential to follow some Data Entry standards or Good Practices. These are presented in the **Appendix of Supplementary Material**.

4) **Error analysis** should be carried out. When errors are reported, they should be registered and totaled. If specific forms are found to generate more errors than others, then there is the possibility that the form is not being properly understood or that it may have programming problems.

5) **Use Audit Trails** whenever possible. Audit Trails are rigorous ways of verifying proper data entry. Throughout the years, the term got distorted to mean the following:

a list of transactions as entered. This is not correct. Audit trails technically mean the following. A batch of vouchers is entered. One of its fields is manually accumulated, say the amount on the voucher. The batch is controlled as one single group of vouchers by the system. The batch or Audit Trail is entered into the system. The batch is not updated or posted to the database unless the computed total equals the entered total. Along with this checking, the system must be able to produce a validation list of all vouchers before committing the entry.

6) **Data Integrity:** various systems are designed without proper data integrity. For example, there are many Inventory Management systems that allow the quantity to go negative. There are cases where the accounting vouchers posted from an operational module do not correspond to the totals recognized by the accounting system. For example, purchases in the Inventory Management system do not correspond to the total inventory in the Accounting System. Such areas should be investigated and checked on a regular basis to ensure that there is Data Integrity.

Input design

Objectives	Guidelines
• Effective • Accurate • Ease to use • Consistent • Attractive • Simple	• Input forms should be easy to fill out, purposeful, facilitate accurate completion, and attractive • Screens should be simple, consistent, facilitate navigation, and attractive • Web-forms should use a variety of input methods, provide clear instructions, demonstrate a logical entry sequence, provide a feedback screen, separate a lengthy form into simpler forms on separate pages

Web-form input methods

1. Text-box
2. Check-box
3. Option button
4. Drop-down list
5. Sliders
6. Image maps
7. Message box
8. Command button
9. Tab control dialog box

Data entry design

Objectives	Guidelines
• Effective coding • Efficient data capture • Effective data capture • Effective input validation	Reduce data input volume • Keep codes concise, stable, unique, sortable, simple, uniform, modifiable, and meaningful (Figure 19.11) • Input necessary data only • Do not input data that can be computed/retrieved from the system • Provide default values when appropriate • Allow look up of value Reduce data input error • Validate input transactions to avoid submitting wrong or unauthorized data and to prevent unauthorized action • Validate input data to prevent missing data, incorrect field length, type, range, value, cross-reference, un-match data.

Types of codes

Purpose	Codes	Example
Tracking information	• Sequence codes	• Order number at Atlanta Bread
	• Alphabetic derivation codes	• U-connect account name
Classifying information	• Classification codes	• Letter grade
	• Block sequence codes	• IP address
Concealing information	• Cipher codes	• Morse code
Revealing information	• Significant-digit subset codes	• Course number system at UK
	• Mnemonic codes	• Acronyms
Requesting action	• Function codes	• Menu choices

Date-entry methods

1. Keyboards
2. Optical character recognition
3. Magnetic ink character recognition
4. Mark-sense forms
5. Bar codes
6. Intelligent terminals

Validation tests

1. Data type/class test – data are of proper type

2. Combination test – data from two or more fields are consistent or reasonable when considered together

3. Expectancy test – data are anticipated, e.g., in some predetermined sequence

4. Existence test – data must be input, e.g., a required field

5. Range test – data are within proper range of values

6. Domain test – data are of proper situation or come from set of standard values

7. Size test – data are of proper length

8. Validity test – data must have certain values, e.g., referential integrity

9. Batch control test – use record count/hash totals to verify batch input

10. Check digit test – add an extra digit to a numeric field to verify its accuracy (Figure 19.19)

User interface design

Objectives	Guidelines
• Task matching • Efficient • Feedback provision • Productivity improvement	Minimize user frustration • Communicate • Minimal user action • Standard operation and consistency • Context-sensitive help Provide feedback on • Acknowledgement • Error/warning • Status • Reassurance • Availability of further assistance

Types of user interface

1. Natural language, e.g., www.askjeeves.com (search engine)
2. Question-and-answer, e.g., dialog box
3. Menu, e.g., Menu bar in Access
4. Form-fill, e.g., registration form as www.yahoo.com members
5. Command-language, e.g., DOS operating systems
6. Graphical, e.g., Icon bar in Access
7. Touch-screen, e.g., ATM machine

Database design

Objectives	Guidelines
• Purposeful information retrieval • Efficient data storage • Data availability • Efficient data update and retrieval • Data integrity	• Use ER diagram for data modeling • All tables are normalized

Data integrity

- Entity integrity: the primary key of a table cannot have a null value
- Referential integrity: all foreign keys in a child table must have a matching record in the parent table
- Domain integrity: table entries must be of the same type, limit, range and other validation checks.

Basic constructs of the E-R model

Concept	Definition	Examples
Entity	A person, place, object, event or concept	Employee, department, building, sale, account
Relationship	An entity that serves to interconnect two or more entity types	Assignment (Employee-Department)
Attribute	A property or characteristic of an entity/relationship type	Employee_name, department_location, sale_date

Types of relationships

1. one-to-one, e.g, STUDENT-Assign-PARKING
2. one-to-many, e.g., DEPARTMENT-Offer-COURSE
3. many-to-many, e.g., STUDENT-register-COURSE

E-R models → relational models

Elements	Relational model
Entity	A table
Attribute	A column of the table
One to one relationship	• One table is created for each entity • The key of either one of the tables is placed as the

	foreign key in the other table
One to many relationship	• One table is created for each entity • The key of the table on the "one" side of therelationship (parent) is placed as the foreign key in the table representing the "many" side of the relationship (child)
Many to many relationship	• One table is created for each entity • One table is created for the relationship itself

T5.3) Data output procedures

Output design

To deliver the right information to the right people in the right format at the right time

Output design

Objectives	Guidelines
• Purposeful • Meaningful • Adequate • Appropriate distribution • Timely • Appropriate medium	• Consider usage factors before choosing an output technology/medium • Avoid output bias • Consider functional and stylistic attributes in designing printed reports • Keep screen output simple, consistent, easy to navigate, and attractive

Types of output technology/medium

1. Print
2. Screen
3. Audio
4. Microform
5. CD-ROM or DVD
6. Electronic
7. Web-based documents

Usage factors in making output technology decisions

1. Users
2. Number of users
3. Physical destination
4. Purpose
5. Speed
6. Frequency
7. Longevity
8. Legal requirements
9. Costs
10. Environmental requirements

Sources of output bias

1. Sorting
2. Setting limits
3. Misleading graphics

T5.4) Implement Java Script and ASP

Implementing JavaScript

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript - Syntax

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within your web page, but it is normally recommended that you should keep it within the **<head>** tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
   JavaScript code
</script>
```

The script tag takes two important attributes −

- **Language** − This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type** − This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like −

```
<script language="javascript" type="text/javascript">
   JavaScript code
</script>
```

Your First JavaScript Script

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

```
<html>
   <body>
      <script language="javascript" type="text/javascript">
         <!--
            document.write("Hello World!")
         //-->
      </script>
   </body>
</html>
```

This code will produce the following result −

```
Hello World!
```

Whitespace and Line Breaks: JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional: Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="javascript" type="text/javascript">
   <!--
      var1 = 10
      var2 = 20
   //-->
</script>
```

But when formatted in a single line as follows, you must use semicolons −

```
<script language="javascript" type="text/javascript">
   <!--
      var1 = 10; var2 = 20;
   //-->
</script>
```

Note – It is a good programming practice to use semicolons.

Case Sensitivity: JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

NOTE – Care should be taken while writing variable and function names in JavaScript.

Comments in JavaScript: JavaScript supports both C-style and C++-style comments, Thus –

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Example

The following example shows how to use comments in JavaScript.

```
<script language="javascript" type="text/javascript">
   <!--

      // This is a comment. It is similar to comments in C++

      /*
      * This is a multiline comment in JavaScript
      * It is very similar to comments in C Programming
      */

   //-->
</script>
```

JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows –

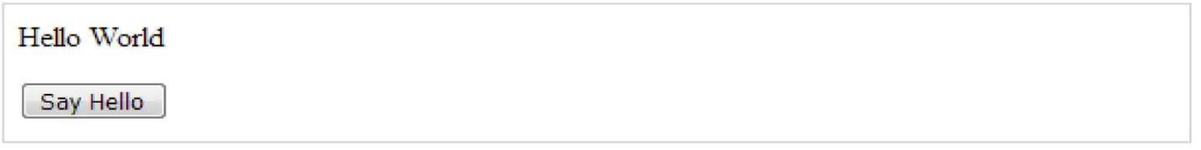
```
<html>
   <head>
      <script type="text/javascript">
         <!--
            function sayHello() {
               alert("Hello World")
            }
         //-->
      </script>
   </head>
```

```
   <body>
      <script type="text/javascript">
         <!--
            document.write("Hello World")
         //-->
      </script>

      <input type="button" onclick="sayHello()" value="Say Hello" />

   </body>
</html>
```

This code will produce the following result –



JavaScript in External File

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using **script** tag and its **src** attribute.

```
<html>

   <head>
      <script type="text/javascript" src="filename.js" ></script>
   </head>

   <body>
      .......
   </body>
</html>
```

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in **filename.js** file and then you can use **sayHello** function in your HTML file after including the filename.js file.

```
function sayHello() {
   alert("Hello World")
}
```

JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types −

- **Numbers,** eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined,** each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

Note − Java does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
<script type="text/javascript">
   <!--
      var money;
      var name;
   //-->
</script>
```

You can also declare multiple variables with the same **var** keyword as follows −

```
<script type="text/javascript">
   <!--
      var money, name;
   //-->
</script>
```

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type="text/javascript">
   <!--
      var name = "Ali";
      var money;
      money = 2000.50;
   //-->
</script>
```

Note – Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

Implementing ASP

ASP stands for **A**ctive **S**erver **P**ages

ASP is a development framework for building web pages.

ASP supports many different development models:

- Classic ASP
- ASP.NET Web Forms
- ASP.NET MVC
- ASP.NET Web Pages
- ASP.NET API
- ASP.NET Core

The ASP Technology

ASP and ASP.NET are server side technologies.

Both technologies enable computer code to be executed by an Internet server.

When a browser requests an ASP or ASP.NET file, the ASP engine reads the file, executes any code in the file, and returns the result to the browser.

An Active Server Page (ASP) is an HTML page that includes one or more scripts (small embedded programs) that are processed on a Microsoft Web server before the page is sent to the user. An ASP is somewhat similar to a server-side include or a common gateway interface (CGI) application in that all involve programs that run on the server, usually tailoring a page for the user. Typically, the script in the Web page at the server uses input received as the result of the user's request for the page to access data from a database and then builds or customizes the page on the fly before sending it to the requestor.