ICT286
Web and Mobile Computing

Topic 3
Introduction to JavaScript Core

Objectives

- Understand JavaScript's execution environment and the difference between web browser execution environment and Node.js execution environment.
- Be able to use console object to display output and error messages, and be able to obtain keyboard input using readline-sync in Node.
- Understand and be able to declare and use variables and understand that declarations are hoisted.
- Understand and be able to use the five primitive types and the associated operators, and understand how the type of a variable is determined, and how JavaScript implicitly convert values from one type to the other and know how to explicitly convert the the type of a variable.
- Understand and be able to define new objects using object literals and use these objects.
- Understand and be able to create new objects using `new` and existing object constructors, such as Date and Array.
- Understand and be able to declare and use functions.
- Understand and be able to use regular expressions.
- Be able to test and run JavaScript code using Node.js.

History of JavaScript

- Originally developed by Netscape and was called LiveScript
- Joint development with Sun Microsystems in 1995 and changed its name to JavaScript
- Become Standard 262 (ECMA-262) of the European Computer Manufacturers Association in 1997.
- The official name of the language is ECMAScript
- The latest edition is ECMA-262 10th edition published in June 2019.
- Supported by nearly all browsers
- It is now also used on the server side (eg, Node.js) and or desktop (eg, Electron)

JavaScript as Client-side Technology

- JavaScript is commonly used as a client-side technology
    - The JavaScript code is downloaded to the web client (ie, web browser) and executed by the client. This frees up the resources on the server, and also provides some dynamics to the web page.
    - The web client has full access to the original JavaScript source code - this means that users have the ability to read the original code.
    - (Client-side) JavaScript cannot do direct manipulation of resources on the server-side (eg. access to data in a central database).
- JavaScript is also increasingly used on the server-side, eg., Node.js provides a run-time environment for executing JavaScript code outside browsers.

JavaScript Execution Environment

- JavaScript was originally designed for the web browser environment. Now it is used in many other host environments, such as Node.JS for server, Electron for desktop.
- JavaScript defines a global object where the core language build-ins and host-specific properties and methods are defined.
- For different host environment, the name will be different.
- For example, in a web browser environment, this global object is named `Window`, and its object reference is `window`.

JavaScript Execution Environment

- If Node.JS is the host environment, the object reference of this global object is `global`.
- In this topic, we use JavaScript with Node.js as its host environment.
- JavaScript has introduced a standard reference for the global object: `globalThis`. However, this name is not yet supported in Node.js version (v10.16.2).
- The `global` object contains the properties and methods defined in JavaScript core, such as `NaN`, `parseInt`, `Object`, `Array`, `String`, `Math`, `RegExp`, `JSON`, as well as properties specific to Node.js.

JavaScript Execution Environment

- Properties defined in object `global` are automatically available everywhere in your scripts, with or without the object reference.
- For example, method `parseFloat` is a JavaScript builtin. Therefore it is a method of the object `global`. Normally we must access it with its object reference, eg:

```
console.log(global.parseFloat(“3.14xradius”);
```

- Because properties and methods in `global` is available everywhere, the method can be also be accessed without the object reference, as in

```
console.log(parseFloat(“3.14xradius”);
```

Console Output

- Both web browser’s `window` object and Node.js’ `global` object contain `console` object.
- The `console` object allows you to display program output, error messages and other information.
- Most web browsers support debugging console. Eg, in Google chrome browser, you can bring up the console by clicking menu: View -> Developer -> JavaScript Console.
- The browser’s console allows JavaScript program to display debugging messages.
- For Node.JS, this console is the terminal window in which you run your JavaScript program. You can use the console to display program output, error message and other information.

User Input

- In a web browser environment, user input can be obtained using the built-in methods `prompt` from the `window` object. We will discuss `window` object in the next topic.
- For Node.JS, JavaScript program may obtain user input from the terminal.
- There are a number of ways in Node.js to obtain user input. The simplest one is `readline-syn` module.
- To use `readline-sync` module, you must download and install it, which can be done with the following command from the terminal:

```
npm install readline-sync
```

Storing Data in Variables

- In all programs, behaviours are defined by storing data, and manipulating data, and outputting data. The data are usually stored in variables.
- A variable is basically a name for a memory address where a value is stored. You can change the value stored in that variable.
- You declare (create) a variable by using the *var* keyword.

Variable Names

- There are rules for the names used for variables in JavaScript:
    - a name must begin with a letter, an underscore (_), or a dollar sign followed by any sequence of letters, underscore, digits and dollar signs.
    - names cannot have spaces in them
    - names are case-sensitive; which means that customer and Customer are not the same variable
    - names cannot be reserved words in the JavaScript language; there is a list of all the reserved words in JavaScript in Table 4.1 of the text (page 143).
- You should use meaningful (descriptive) names for your variables.

Variable Declarations

- In JavaScript, there are two ways you can introduce new variables:
  - *Explicit declarations*: using var, with or without initialisation. Example:

    ```
    var x;
    var y = 10;
    ```

  - *Implicit declarations*: without using var, but always with an initial value. Example:

    ```
    radius = 5.4;
    circumference = radius * 2 * 3.14;
    ```

    When you initialised an undeclared variable, such as `radius` and `circumference` in the above example, JavaScript automatically declared it.
- It is recommended that you always declare variables explicitly.

Example Code with Variables

```
var x;

var y = 0;

var z = x + y;

x = z - 1;
y = 2 * 2;

var my_name = "John";

var my_other_name = "Jane";
```

Hoisting of Declarations

- An explicit declaration with initialisation such as

  ```
  var x = 10;
  ```

  can be seen as the combination of a declaration and an assignment:

  ```
  var x;
  x = 10;
  ```

- It is important to note that JavaScript declarations (but not the assignment), including variable declarations and function declarations, are “hoisted”. The declarations are treated by the JavaScript interpreter as if they are all moved to the top of the current scope.
- Example:

```
console.log(x); // print undefined
var x = 10;
console.log(x); // print 10
```

Primitive Data Types

- In JavaScript each primitive value has a data type. The following are JavaScript's five primitive data types:
  - Number: a number
  - String: a sequence of characters
  - Boolean: only two values `true` or `false`
  - Null: only one value `null`
  - Undefined:
- Unlike C/C++ or Java, JavaScript *variables* do not have a fixed type. Their types are determined at the runtime, based on the *values* the variables have at the time.

Variable Types

- In declaring a new variable, you do not need to explicitly say what data type the variable is - it is inferred from the value you give it during the runtime.
    - eg. 123 would be inferred as a numeric type (Number), while "one" would be a string type (String).
- Therefore the type of the variable can change during runtime.
    - In the following example, variable x was initially of type Number, but later it changes to String:

```
var x = 10;
x = “WWW”;
```

Use of Symbol Not Defined

- Any attempt to use a symbol that is neither explicitly defined, nor implicitly defined, would cause a runtime error: the symbol is not defined.
- Eg

```
console.log (x);
```

Executing the above code from Node.js would produce a run-time error:

```
ReferenceError: x is not defined
```

Undefined Variable

- If a variable is explicitly declared, but not initialised, its type is *undefined,* but its value is the string “undefined”
- Eg.

```
var x;
console.log(x+”string”);
console.log(typeof x);
```

Executing the above code from Node.js would produce the following output:

```
undefinedstring
undefined
```

Null Variable

- If a variable is assigned the value null, it means it is not pointing to any object.
- However the variable's type is Object.
- Eg.

```
var x = null;
console.log(x);
console.log(typeof x);
```

Executing the above code produces the following output:

```
null
object
```

Numbers and Strings

- Number values are represented internally as double-precision floating-point values
  - Number literals can be either integer or float,
  - Float values may have a decimal and/or an exponent
  - Examples:
    - Integers; 5, 100, 65;
    - Decimals: 12.6, 0.77, .78, 21.;
    - With exponent: 5e2, 5.678e3, 988.111e-2
- Strings are collections of characters
  - The characters in the string must be enclosed in quotes (" "); for example "Jane"
  - They can be a number, if no arithmetic is to be performed on it; for example "33"
  - If you need a quotation mark inside a string, use ‘, or else place ‘\’ before the quote; for example "Christine’s house" or "Lee said \"Come here\""

Special Literals

There are some special literals defined in JavaScript that you may find useful:

- NaN: not a number
  - var temp = 3*"cat";   will cause temp to contain NaN
- undefined: a variable that has been declared , but not initialised
- Infinity: results from division by zero (0)

Implicit Type Conversion

- JavaScript attempts to convert values in order to be able to perform operations
- "August" + 2019 causes the number to be converted to string and a concatenation to be performed (operator "+" is treated as string concatenation).
- Similarly, in "2019" + 2008 and 2019 + "2008", the operator "+" is treated as string concatenation operator, the numbers are converted into strings.
- However, in expression such as 7 * "3", "7" - 3, "7" – "3", "7" * "3", and "7" / "3", these operators are numerical operators, hence strings are converted into numbers.
- However strings such as "7x" cannot be converted into numbers. Eg, "7x" * 3 would result in NaN.

Implicit Type Conversion

- The value undefined is converted to NaN in a numeric context
- The value 0 is interpreted as a Boolean false in a Boolean context, all other numbers are interpreted as a true
- The empty string is interpreted as a Boolean false in a Boolean context, all other strings (including “0”!) as true
- The values undefined, null and NaN are all interpreted as false in a Boolean context

Explicit Type Conversion

- Explicit conversion of number string to number using object Number. Eg:

```
console.log( Number(“303”) +10  );
```

would print 313.

- parseInt and parseFloat convert the beginning of a string. Eg:

```
Console.log( parseInt(“50South Stree”) + 10 );

Console.log( parseFloat(“3.14=pi”) + 10 );
```

would print 60 and 13.14 respectively.

Comments

- In JavaScript code, when the string // is encountered, all characters after it in the same line will be ignored by the interpreter.
    - That is, it can be used for comments.
- Make use of comments in your code, so that you can understand it and someone else (having to fix or change your code) can too.

$$\text{var email; }\underbrace{\text{//the email address typed in by the user}}_{\text{As far as the program is concerned, this section might as well not exist. It is only useful for humans reading it.}}$$

Operators

JavaScript has a set of operators which can be used for

- testing, or
- manipulating variables

eg.

- >, <, <=, >=, ==, != : to test the relationship between two numbers
- &&, || !: (logical AND) and (logical OR) and (logical NOT)
- + : add two numbers or concatenate two strings
- -, *, / : subtract, multiply and divide two numbers
- ++, -- : increase or decrease by 1
- +=, -=, *=, /= : add/subtract/multiply/divide, followed by assign

Examples of use of Operators

```
i++; (same as i = i+1;)

if (i<=20)  ...

if ((i==20) && (Month!=12)) ...

total = (time * rate) + 100;
```

Control Statements

- JavaScript code exists in statements.
- A statement ends with a semi-colon.
- All simple statements, which are separated by semi-colons, are executed from left to right, and from top to bottom.

Statements

- We can perform more interesting behaviours by using more interesting *control statements*.

    - if…else
    - for
    - while

if…else

The *if*-statement is a *conditional*

- It allows us to define a decision - under this condition, do this; under that condition, do that; etc.

Format:

```
if (condition) {
     statements
} else {
     statements
}
```

Example of *if…else*

```
var day_of_week;

// code here to get day_of_week

if (day_of_week == "Monday") {
      console.log("Today is Monday");
} else {
      console.log("Today is not Monday");
}
```

for loops

- Loops are for executing a section of code repeatedly.
- The *for* loop controls how many times the code is repeated by having a control variable.
- General format:

```
for (counter=starting value ;
     counter <= ending value ;
          counter++)
{
     statements
}
```

Example of *for loop*

```
for (var count =0; count < 5; count++)
{
   // repeated statements go here
   console.log("The counter is " + count);
}
```

while loops

- Another more general form of loop is the *while* loop. This is used when we do not know how many times the statements inside the loop should be executed.
- Format

```
while (condition)
{
      statements
}
```

Example of while loop

```
var loop_count = 1;

while (loop_count < 4)
{
   if (loop_count == 3) {
      console.log("3 strikes – you’re out!");
      loop_count = loop_count + 1;
   } else {  // some other code in here
      loop_count = loop_count + 1;
   }
}
```

Break and Continue

- The *break* statement can be used to get out of the middle of a loop. Use *break* with care. Your loops should normally be **designed** so that you do not need to use a *break* command.
- *continue* is the complement of *break* and can be used to take execution to the beginning of the *for* or *while* loop ignoring any further commands in the block.

Compare the two loops on the next slides.

Break Example (without break)

```
// need to install readline-sync module in node.js:
// npm install readline-sync

var readline = require('readline-sync');
var answer;
var correct = false;

while (!correct)
{
   answer = readline.question("What is your name?");
   if (answer == "Hong")
        correct = true;
}
```

Break Example (with break)

```
// need to install readline-sync module in node.js:
// npm install readline-sync


var readline = require('readline-sync');
var answer = 0;
var i = 0;

for (i = 0; i < 10; i++)
{
   answer = readline.question("What's your lucky 
number?");
   if (answer == 7) {
      break;
   }
}
```

Switch

- Sometimes using if-else statements can be difficult because there are a lot of possibilities. In this case a switch statement may be more appropriate. The switch statement is like a case in other languages.
    - Make sure you understand the difference between a *switch* statement and multiple nested *if else* statements.

Example of Switch

```
var now = new Date();     // create a new date object;
var monthname;
var month=now.getMonth(); // a function returning
                          //the current month 0 ..11
switch (month) {
      case 0 :
                  monthname="January";
                  break;
      case 1 :
                  monthname="February";
                  break;
      case 2 :
                  monthname="March";
                  break;
      default :
                  monthname= "Invalid";
}

console.log("The current month is " + monthname);
```

Functions

- A function allows you to define a piece of code once and uses it many times with different arguments.
- A function can have a list of parameters.
- You can define local variables within a function whose scope is limited to the function.
- A function often returns a value.
- Example

```
function areaOfCircle (radius) {
    var PI = 3.1415;
    return PI*radius*rsadius;
}
console.log(”circle area is “ + areaOfCircle(5.4));
```

Functions

- As function declarations are hoisted, they can be invoked before or after they are declared.
- Example

```
console.log(”circle area is “ + areaOfCircle(5.4));
function areaOfCircle (radius) {
    var PI = 3.1415;
    return PI*radius*radius;
}
```

- In the above example, the function `areaOfCircle` was invoked in the `console.log` before it is declared.
- Due to “hoisting”, the function declaration is treated as if it is declared at the top of the scope regardless where it is actually declared.

Functions

- A function can also be declared and assigned to a variable.
- Example

```
var areaOfCircle = function(radius) {
    var PI = 3.1415;
    return PI*radius*rsadius;
};
```

  Notice the semicolon (in red color) at the end of the function variable declaration.

- However, such a function can only be invoked after the function is assigned to the variable.

```
var areaOfCircle = function(radius) { . . . };
console.log(”circle area is “ + areaOfCircle(5.4));
```

What is an Object?

- In the physical world, an object is a thing; for example a cat, a car or a person.
- An object in the physical world has both a set of properties and a set of behaviours.
- For example, a dog has a set of properties such as breed, fur colour, age, and weight.
- A dog can eat, can run, can bark, etc. These are the dog’s behaviours.
- In JavaScript, an object is a computer representation of a real object in the physical world.

JavaScript Object

- A JavaScript object models both the physical object’s properties, such as hair colour and age of a dog, and its behaviours, such as dog barking.
- The properties are represented by a set of data stored in the JavaScript object (you may think of them as a set of variables).
- The behaviours are represented by a set of methods (or functions) defined in the JavaScript object which can manipulate the data stored in the object.

Types of JavaScript Objects

- JavaScript core built-in objects:
  - to represent the type of data such as `Number`, `String`, `Boolean`, `Array`
  - for special tasks: `Date`, `Math`, `RegExp` (regular expressions)
- Standard objects provided by the web browser environment
  - to represent the objects associated with the web browser: `navigator` (for browser), `window` (for the window in which the HTML document is displayed), `history` (browsing history of the browser), and `location` (current URL of the window).
  - HTML DOM objects: the HTML page in a window is internally represented as a tree of objects, each representing an HTML element including its attributes. These objects are created by the browser after the HTML document is parsed. The top level DOM object is `document` object.
- The library of objects in a given JavaScript running environment, such Node.js
- The user can also create his or her own objects.
  - however in this unit, we will focus on the use of built-in and standard objects provided by the web browser environment.

Object Creation

- JavaScript objects can be created in two different ways:
  - Created using an object literal, such as

```
var myDog = {
     name: “alex”,
     breed: ”Labrador”,
     color: “black”,
     bark: function(){ console.log(“Woof woof woof!”) }
};
```

Each object consists of a list of unordered properties, and each property consists of a *name*:*value* pair.

  - Created using new and an object constructor, such as

```
var today = new Date();
```

Accessing Objects

- Properties and methods of an object are accessed via its object reference using the dot notation:

  Object_ref.property;

  Object_ref.method(…);

  - Examples:

```
var dogColor = myDog.color;
myDog.bark();


var year = today.getFullyear();
var hour = today.getHours();
```

An Example of Object Creation using Literals

```
var student = {
    name: "John",
    student_no: 123456,
    major: "computer science",
    info: function(){
        return this.name + " " + this.student_no
              + " " + this.major;
    }
}

console.log("student name is " + student.name);
console.log(student.info());
```

Object Creation With Constructors

- The `new` expression, which includes a call to the constructor of an existing object, is used to create an object

```
var today = new Date();
var list = new Array (1, 2, “three”, 4);
var obj = new Object();
```

- The constructor will create a new object (eg, set aside memory for it) and initialise it with the relevant properties and methods depending on which object constructor was used. Finally the constructor returns the memory address of the new object. This memory address is known as the new object’s “object reference”.

What Are in the New Object?

- The initial properties and methods in a new object depends on the constructor used to create the object.
  - For example, the object `today` would contain the current date and time and methods such as `getDate()` and `getHours()`.
  - while the object `obj` is largely empty, as it is from `Object`.

- The number of properties of an object may vary dynamically in JavaScript:
  - You can add properties to, and delete properties from, an object at run-time.

Object Modification

- Assuming the following new object `car`:

```
var car = { make: “Holden”, model: ”Commodore” };
```

- Alternatively you can create the same object with the Object constructor and then add two properties “make” and “model” to the object

```
var car = new Object();
car.make = “Holden";
car.model = “Commodore";
```

- The delete operator can be used to delete a property from an object

```
delete car.model;
```

Built-in Object Constructors

- JavaScript core consists of a number of builtin object constructors including:
    - `Object`
    - `Date`
    - `Array`
    - `String`
    - `RegExp`
    - `Math`
    - `JSON`
- Apart from `Object`, we will also cover `Date`, `Array`, `String` objects in this topic. Later topics will cover more.

The Date Objects

- You often need to create or manipulate dates. JavaScript's built-in object `Date` returns the current date and time.

```
    Var today = new Date();
```

- In the above example, the reserved word `new` creates a new object, while `Date()` is a constructor for the `Date` objects which, together with `new` creates a new `Date` object. The constructor returns the object reference of the new object and the object reference is stored in variable `today`.

Date Object Methods

- Methods that are associated with the `Date` object can be used to manipulate the date and/or parts of it. Some of the common ones are shown in the next slides.
- These methods can be invoked using the object reference and the dot symbol, eg:

```
var year = today.getFullYear();
```

Some Date Methods

Method	Description
`getDate()`	Returns a number from 1 to 31, representing the date of the month.
`getDay()`	Returns a number from 0 (Sunday) to 6 (Saturday) representing the day of the week.
`getFullYear()`	Returns the year as a four digit number.
`getHours()`	Returns a number from 0 to 23 representing hours since midnight.
`getMinutes()`	Returns a number from 0 to 59 representing the minutes for the time.

Some Date Methods

Method	Description
`setDate(`*`val`*`)`	Sets the day of the month to *val*.
`setHours(`*`h`*`,`*`m`*`,`*`s`*`,`*`ms`*`)`	Sets the hour; the first argument is the only one required.
`setMonth`*`(val)`*	Sets the month to *val*.
`toString()`	Returns a string representation of the date and time specific to the locale of the computer.

Example:

```
var today = new Date();
var today_str = today.toString();
console.log("Today is " + today_str);
```

Arrays in JavaScript

- `Array` is a special object in JavaScript.
- An array is a list of variables that are usually related in some way and can be referenced using an index. In JavaScript an array is an object, so to create an array we use the reserved word `new`:

```
arrayName = new Array( arrayLength );
```

Eg:

```
var list = new Array (10);
```

The variable list points to an array of length 10, but no array element is added to the array yet.

Arrays in JavaScript

- You can also assign a list of array elements when creating a new array, eg:

```
var my_list = new Array(1, 2, “three”, “four”);
```

or simply,

```
var my_list = [1, 2, “three”, “four”];
```

The variable my_list is an array of 4 elements:

my_list [0]:	1
my_list [1]:	2
my_list [2]:	“three”
my_list [3]:	“four”

Arrays in JavaScript

- An array can have elements of different kinds and can be grown by adding elements past the arraylength.
- Example:

```
TestArray = new Array(3);
TestArray[0] = "cat";
TestArray[2] = 1234;
TestArray[3] = "a new element";
```

- Note that in the above array, the second element is not defined. The last element has gone past the original definition.

Properties of Arrays

- The most important property of an array is the one we have already seen, the `length`, that contains the length of the array. You will see it commonly used in loops moving through an array:

```
for (var i=0; i<TestArray.length; i++) {
   . . . . . .
}
```

Array Methods

- `pop()`: removes an element from the end of the array, and returns the removed element.
- `push()`: adds one or more elements to the end of the array.
- `shift()`: removes the first element from the array and returns the removed element.
- `unshift()`: adds one or more elements to the beginning of the array.
- `splice()`: adds and/or removes a portion of the array.

Array Methods

- `sort()`: sorts the elements of the array alphabetically.
- `reverse()`: reverses the order of elements in the array.
- `slice()`: returns a portion of the array, called a subarray.
- `concat()`: combines the elements of two arrays into a third.

The String Objects

A String object has many mmethods (functions) that allow you to manipulate strings. We will only consider some of these here.

- `string.charAt(index)` – returns the character which is at position *index* in the string. One thing to remember is that the first character in a string is considered to be in position 0 (zero); not 1. So, for example:

```
var mystring = "Have a nice day";
var mychar;
mychar =  mystring.charAt(1);    // will return "a"
mychar =  mystring.charAt(9);    // will return "c"
```

String Methods

- `stringa.indexOf(stringb)` – allows you to find the index of the first occurrance of stringb within stringa. If the string is not found, then the value -1 is returned. So, for example:

```
var mystring = "Have a nice day";
var myIndex;

myIndex = mystring.IndexOf("a");  // will return 1
myIndex =  mystring.IndexOf("b"); // will return -1
```

String Methods

- `string.substring(start, end)` – allows you to capture a string within a string. The end value is the position AFTER the one to be returned. If the end value is omitted, then the length of the string is assumed. So, for example:

```
var mystring = "Have a nice day";
var mychars;
mychars = mystring.substring(0, 4 );// will return "Have"
mychars = mystring.substring(12);  // will return "day"
```

String Methods

Method	Parameters	Result
charAt	A number	Returns the character in the String object that is at the specified position
indexOf	One-character string	Returns the position in the String object of the parameter
substring	Two numbers	Returns the substring of the String object from the first parameter position to the second
toLowerCase	None	Converts any uppercase letters in the string to lowercase
toUpperCase	None	Converts any lowercase letters in the string to uppercase

Regular Expressions

- Regular expressions are sets of rules that define a set of possible matching strings. Regular expressions have been used in other languages for a long time, and have also been introduced to JavaScript.
- Regular expressions in JavaScript are based on the syntax used in Perl, but there are some small differences.
- JavaScript provides many methods in `String` object to handle regular expressions.
- JavaScript also has an object RegExp that provides more powerful facilities for handling regular expressions.

Regular Expressions

- Regular expressions can be used in JavaScript and PHP (which will be covered in a later topic), so we will spend some time looking at the syntax in JavaScript first.
- A regular expression is an object defined with the following syntax:

  /pattern/modifier

  The modifier can be one of “i” (ignore case), “g” (global match), and “m” (multiline matching) .

  For example:

```
var  re = /topic\d+/i;
```

A Simple Example

- The `match` method from `String` object returns an array of substrings that matches the pattern, or null if there is no match.
- Eg:

```
var MaryString = "Mary had a little lamb";
var MatchingString =
MaryString.match(/Mary/);
 console.log(“Matching string: “
                 + MatchingString);
```

Case-Insensitive Matching

- The letter `i` following the string pattern means ignoring the case during pattern matching. Eg:

```
var MaryString = "Mary had a little lamb";
var re = /LAMB/i;

if (MaryString.match(re)) {
   console.log(”with lamb");
}
else {
   console.log(”no lamb");
}
```

String Search

- The search method from `String` object returns the index of matching string or -1 if there is no match.
- The output of the following code is as follows: 'bits' appears in position 3.

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
if (position >= 0)
  console.log("'bits' appears in position", position);
else
  console.log("'bits' does not appear in str");
```

Metacharacters

- Some characters have special meaning to JavaScript regular expressions, so they must be used in character matching in a special way - this is called escaping. In JavaScript you place a backward slash in front of the special character in a regular expression. These metacharacters are:

```
\  |  ^  $  .  ?  *  +  {  }  [  ]  (  )
```

- Example:

```
var metaString="Does this string contains metacharacters?";
var re = /\?/;  // escape the special meaning of ? in RE
if (metaString.match(re))
   console.log("Your string contains a metacharacter");
```

Character Classes

Symbol	Function	Example
[xyz]	Match any one character enclosed in the character set .	/[AN]BC/ matches ABC and NBC but not BBC.
[a-z]	Match any character between a and z; other ranges are possible.	/[A-C]BC/ matches ABC and BBC but not NBC.
.	A wildcard; matches a single character except a newline.	/b.t/ matches bat, bit, but, bet, …
\d	Match any single digit. Equivalent to [0-9]	
\s	Matches any single space character.	
\w	Match any single word (non-punctuation or non-whitespace) character.	

Position Matching

Symbol	Function	Example
^	Only matches the beginning of a string.	^P matches first *P* in "*Paul Patterson, President*."
$	Only matches the ending of a string.	t$ matches the last *t* in "*A cat in the hat*"
\b	Matches any word boundary (test characters must exist at the beginning or end of a word within the string).	ly\b matches *ly* in "*JavaScript is really cool*."
\B	Matches any non word boundary	*\Bor* matches the *or* in "*normal*" but not the one in "*origami*"

Alternatives and Grouping

- The | can be used like an “or”, so that several patterns can be tested. Characters can be placed inside round brackets () when testing for several alternatives:
- Example:

```
var metaString = "Does this string contains
  metacharacters?";
var re = /(ab)|(ac)|(ad)/;
if (metaString.match(re))  {
  console.log("Your string contains ab or ac or ad")
}
```

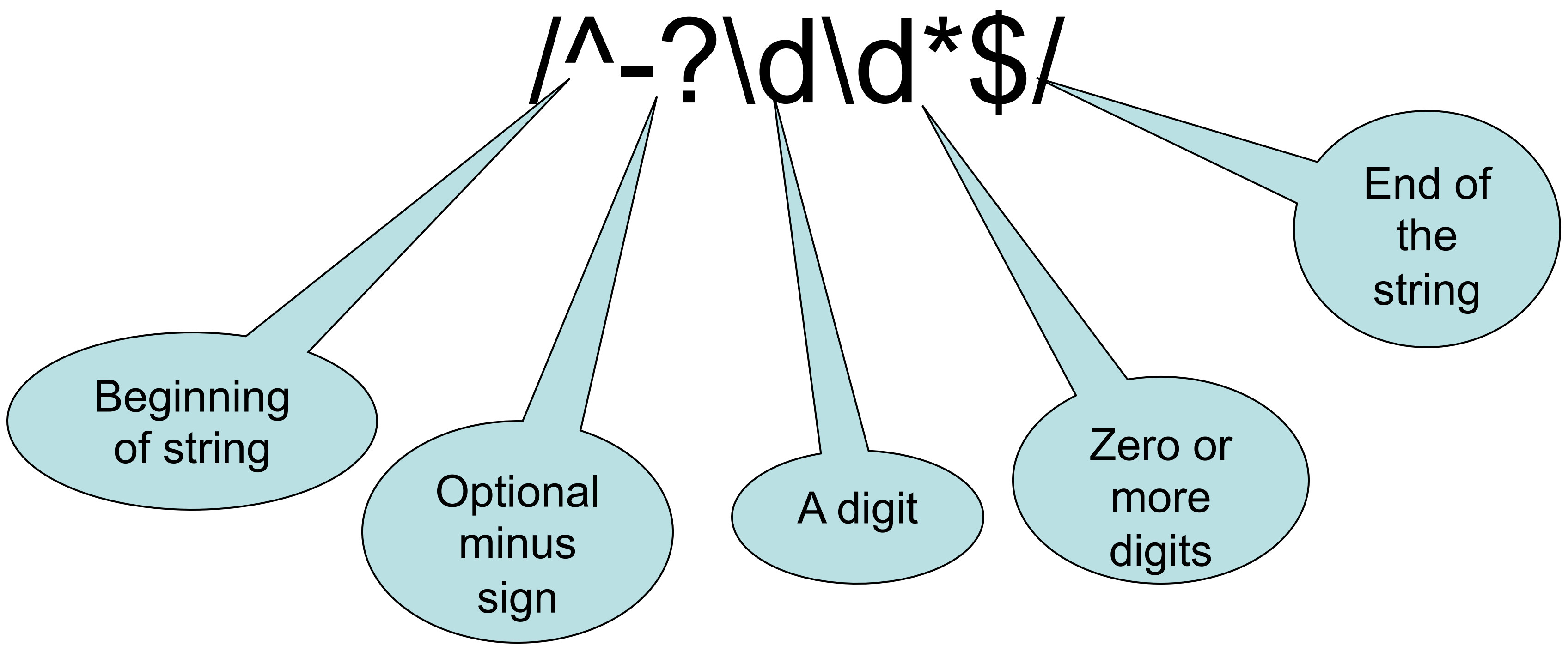
Repetition

Symbol	Function	Example
{x}	Match exactly x occurrences of a regular expression.	\d{5} matches 5 digits.
{x,}	Match x or more occurrences of a regular expression.	\s{2,} matches 2 or more spaces.
{x,y}	Matches x to y (inclusive) number of occurrences.	\d{2,3} matches at least two, but no more than three digits.
?	Matches zero or one occurrence.	Same as {0,1}.
*	Matches zero or more occurrences.	Same as {0,}.
+	Matches one or more occurrences.	Same as {1,}

Additional RegExp Methods in String Object

Method	Description
`replace(`*`regular_expression`*`, `*`replacement_text`*`)`	Returns a copy of the string with text replaced.
`split(`*`regular_expression`*`)`	Returns the array of strings that result when a string is separated into substrings. Splitting is done based on the occurrences of the regular expression matches.
`search(`*`regular_expression`*`)`	Returns the position of the first substring match in a regular expression search.

Some Examples



A valid integer in a line

Some Examples

```
function RemoveCommas(str) {
   // define a regular expression;
   // note the use of global
   // so that all commas are removed
   var re = /,/g;

   //replace all commas with a space
   return str.replace(re,’’);
 }
```

Readings

- Textbook:
  - Sebesta: Chapter 4
- W3Schools:
  - http://www.w3schools.com/js/default.asp
- Kindle book:
  - Mark Myers: A Smart Way to Learn JavaScript