JAVASCRIPT

5

PROGRAMMING ON THE WEB

- Follows same principles/best practices as programming for other media
- Good development process is very important!
- Different dev environments
- Mostly scripting languages (not compiled)
- Robust IDE may not always be available

6

PROGRAMMING ON THE WEB

INPUT

- Forms / GUI elements
- Database
- File
- Sensor
- Gather input ***before*** attempting to process

PROCESS

- Calculations
- Functions
- Get result from web service / API
- Process results ***before*** output

OUTPUT

- Display on screen
- Write to GUI element
- Output to database
- Output to file
- Make element on page *do* something
- *The user can see or inspect the output*

7

7

CLIENT-SIDE PROGRAMMING WITH JAVASCRIPT

- Similar to C/C++
- Scripting language
- Interpreted
- Supports OOP
- Object libraries: DOM (document object model)

Incorporate into HTML page using <script> tag:

```
<script>
    document.write('hello world');
</script>
```

8

8

I/O: OUTPUT

- Output to a popup
  - alert()
    - alert("hello");
- Output to the page
  - document.write(), document.writeln()
  - document.write ("hello");
- Output to the Console
  - console.log("hey");
- Write to an element on the page
  - The item must have been "loaded"
  - document.getElementById("result").innerHTML = "hello";

9

9

CONCATENATION

- *+ to concatenate (glue) strings*
  - x = "hello "; y = " there";
    z = x + y; // z is now "hello there"
- *If you concatenate a string with a number, the result is a string*
  - x = "hello "; y = "12"; z = x + y; // z is now "hello12"
  - `n = 4; n = n + "";  //n is now "4"`
  - `n =4; z = "The answer is " + n + 2);  // z is "The answer is 42"`
  - `n =4; z = "The answer is " + (n + 2));  // z is The answer is 6"`

Display a variable:
message = "hello";
document.write (message);

Display a variable along with text:
message = "hello";
document.write ("I said " + message);

10

10

I/O: INPUT

- Yes or no
  - confirm
    - answer = confirm ("Are you sure?");
- Open response
  - prompt(question, default)
    - answer = prompt ("School name?", "Tufts University");
    - answer = prompt ("School name?")
- Read value from a form element (most common)
  more on this later

11

11

VARIABLES IN JAVASCRIPT

- Declare a variable
  - Use the keyword "var", "let", or simply assign a value.
    - var x;
      x = 10;
    - let x = 10;
    - x = 10;
- Declare several variables at the same time with var
  - var x, y;
  - var x = 10, y;
- *null* assigns a non-value to a variable
  - var x=null;
  - Use the function isNull() to check for null
- const - can't change

12

12

SCOPE

- Global
- Local
- Block
- var vs let
  - var is function scoped
  - let is block scoped
- const is block scoped

```
<script>
        var abc;        // global
        function myfunction()
        {
                var x;  //local
                if (x)
                {
                        // abc, x are in scope
                        let w;  // block
                        var y;  // local
                }
                // abc, x, y are in scope
        }  //end function
        //abc is in scope
</script>
```

13

13

SCOPE EXAMPLE

```
<script>
        var globalScope=0;
        function foo() {
          if (true) {
                var funcScope = 1;
                let blockScope = 2;
                console.log(funcScope);          // will print 1
                console.log(blockScope);         // will print 2

                let globalScope = 3;
                if (true) {
                  console.log(funcScope);        // will print 1
                  console.log(blockScope);       // will print 2
                  console.log(globalScope);      // will print 3
                }
          }
          console.log(funcScope);                // will print 1
          console.log(globalScope);              // will print 0
          console.log(blockScope);               // error, not in scope
        }
        foo();
</script>
```

14

DATA TYPES IN JAVASCRIPT

- Javascript is loosely typed- data types are deduced
- Variables in Javascript are shape shifters!
- Numbers
  - Integral (no decimal point)
  - Floating point (*may* have a decimal point)
- Text (Strings)
  - Enclosed in quotes
  - Both single and double quotes are valid (end what you start with)
- Boolean
  - True or false

The Shape Shifter
Image credit: Tomasz Alen Kopera on Pinterest

15

15

WORKING WITH DATA

- Input from a user is always a string
- parseInt, parseFloat convert a string to a number
  - The converted value is returned *the original value is not changed*
- NaN (not a number)

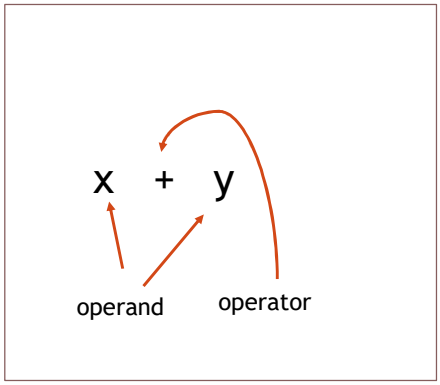
```
alert(parseInt("1 is the loneliest number"));
    //result is 1
alert(parseInt("The loneliest number is 1"));
    // result is NaN
alert(parseInt("3.14 is my favorite kind of pi"));
    // result is 3
alert(parseFloat("3.14 is my favorite kind of pi"));
    // result is 3.14
```

16

16

OPERATORS

- Arithmetic
- Arithmetic with assignment
- Assignment
- Comparison
- Logical
- Concatenation



17

17

ARITHMETIC

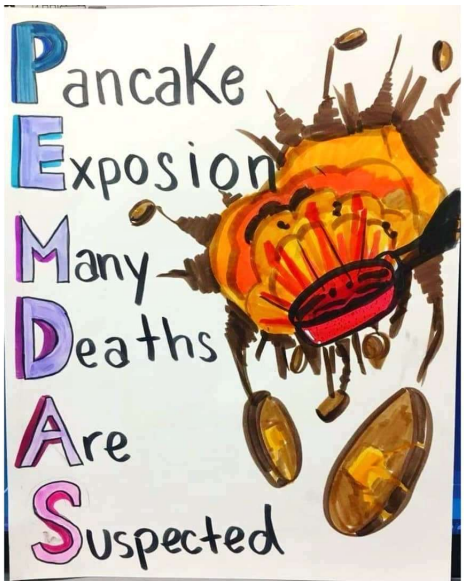
+	addition	Adds numeric operands.
-	subtraction	Used for negating or subtracting.
++	increment	Add 1 to the operand.
--	decrement	Subtract 1 from the operand.
*	multiplication	Multiplies two numerical operands.
/	division	Divides first operand by second operand
%	modulus	Calculates the remainder of first operand divided by second operand.

18

18

ORDER OF OPERATIONS

- PEMDAS
- Please Excuse My Dear Aunt Sally
- Parentheses
- Exponent
- Multiplication, Division
- Addition, Subtraction



19

EXAMPLE: TIP CALCULATOR

```
var checkAmount, tipAmount, TIP_PERCENT=.17;
checkAmount = prompt("What is the check amount? ","20");
tipAmount = checkAmount * TIP_PERCENT;
document.write("Check amount: $"+ checkAmount + "Tip: $ " + tipAmount);

// Why isn't parseInt() needed?
```

20

20

EXAMPLE: IS A NUMBER EVEN?

```
var num, result;
num = prompt("Enter a number", "5");
result = num % 2;
alert("The remainder of " + num + " divided by 2 is " + result);
```

21

21

COMPOUND ASSIGNMENT OPERATORS

- `+=, -=, *=, /=, %=`
- Performs the operation and does the assignment
- Example:
  `x = x + 3;`
  is the same as
  `x += 3;`

```
Four ways to add 1

   x = x+ 1;
   x++;
   ++x;
   x += 1;
```

22

22

COMPARISON

==	equality	True if two operands have the same value (data conversion may occur)
===	strict equality	True if two operands have the same value and same data type
!=	inequality	Opposite of the equality (==) operator.
!=	strict inequality	Opposite of the equality (===) operator.
>	greater than	True if first operand is larger
>=	greater than or equal	True if first operand is greater than or equal to a second operand
<	less than	True if first operand is smaller
<=	less than or equal	True if first operand is less than or equal to a second operand

23

23

CONDITIONAL OPERATOR
TEST ? RESULT1 : RESULT2

- Do the test, if it's true, then the value is *result1* otherwise it is *result2*
- Remember: this is an operator, there is no implied assignment
- "Absolute value" example:
  - n = n<0 ? -n : n;
- "Even" revisited

```
var num, result;
num = prompt("Enter a number", "5");
result = num % 2;
alert("The number: " + num
          +  (result==0 ? " is " : " is not ")
          + "even";
```

24

24

LOGIC OPERATIONS

- Javascript supports three logical operations:
  - && AND (shift-7) true when two operands are **both** true
  - || OR (shift \) true when **either** of two operands is true
  - ! NOT (shift 1) (opposite) true when an operand is false
- Logical operands "glue" two conditional expressions together.
- For example, there is no "between" operator, but you can accomplish "between" with AND

```
between = z>10 && z<=20;   // true when z is 11 through 20
```

25

25

TRUTH TABLES

- Truth Tables are another way to represent the outputs of a logical operator.
  - false is 0 and true is 1
- Truth Table for AND

exp1	exp2	result
F	F	F
F	T	F
T	F	F
T	T	T

Truth Table for OR

exp1	exp2	result
F	F	F
F	T	T
T	F	T
T	T	T

26

26

"FORCED" VALUES

- When a value of one input expression forces the output result of a logical operator, it is called a "Forcing Function"
- The forcing function for AND is when an input is false.
- The forcing function for OR is when an input is true.

- **Shortcut operations:**
  - If the first operand represents a forcing function, the second operand is not evaluated
  - Example: (6 > 3) && (5 > 3)

27

27

MIXING NUMBERS, STRINGS AND OPERATORS

- + operator
  - string + string concatenation
  - string + number convert number to string and then concatenate
  - number + number addition

- Comparison operators (ex, == >)
  - string > string alphabetical order
  - string > number convert string to number and compare numerically
  - number > number numerical order

28

28

FIGURE IT OUT

- What is:
  - `3 > 4 && 6 < 10`
  - `“6” == 6 || 8< 0`
  - `! (5 > 2)`
- Assume n = 10, what is:
  - `n > 0`
  - `n <= 5 && n >= 10`
  - `n >= 1 && n<= 10`
  - `n == “10” || n < 3`

29

29

PREFIX VS POSTFIX NOTATION

- x++
  value is x
  add one to x
- ++x
  value is x+1
  add one to x

Given, x = 3

```
y = x++;     //y = 3,  x = 4
y = ++x;     //y = 5,  x = 5
```

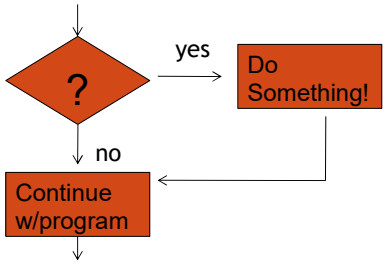
30

30

CONDITIONAL STATEMENTS

- The IF construct helps determine PROGRAM FLOW based on a yes/no question.

- ```
  if (n>1)
      alert ("Do something");
  ```



31

31

SIMPLE IF SYNTAX

- ```
  if (n>=0)
      alert("N is a positive number");
  ```
- After the keyword "if" is a *conditional expression* - an expression that is evaluated as true of false
  - Comparison operators
  - Boolean logic operators (and, or)
  - Anything can be evaluated as true/false (recall that zero is false)
- If the expression is true, then do the statement immediately following the "if" **Otherwise**, skip that statement.
  - Several statements can be included in the if block by enclosing them in a code block `{ . . . }`

32

32

EXAMPLES: SOLVE THE FOLLOWING USING IF STATEMENTS

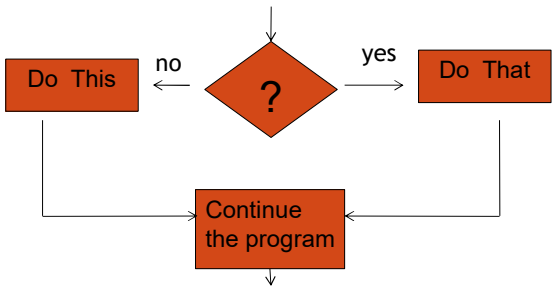
- If a variable named "likesBeer" has a value of true, display "I like beer too!"
  - ```
    if (likesBeer)
        alert("I like beer too!");
    ```
- If a variable called "numPets" is 0, display "you do not have pets"
  - ```
    if (!numPets)
        alert("You do not have pets");
    ```
- Compare two numbers, n1 and n2. If the first number is greater than the second display "The first number is bigger"
  - ```
    if (n1 > n2)
        alert( "The first number is bigger" );
    ```

33

33

OTHERWISE …

- Sometimes we want to take different actions when the conditional expression is true and when it is false.
  - Use an ELSE statement (think "otherwise")
- ```
  if (n>=0)
        alert("n is a positive number");
  else
        alert("n is a negative number");
  ```
- You can do several statements after the else by enclosing them in a code block { . . . }



34

34

NOTES

- Each part of an if / else if/ else statement is mutually exclusive. i.e., only one can be true.
- *Do not* put a conditional expression after a lone "else"
  ie: else (n>3) //WRONG!
- You must use curly brackets when there is more than one statement in an if or else block. Curly brackets are optional for a single statement.

35

35

EXAMPLES: SOLVE USING IF/ELSE STATEMENTS

- If a guess matches a number, display "You are correct", otherwise display "Sorry, try again"

```
if (guess==number)
        alert("correct");
else
        alert("Sorry, try again");
```

- If a guess matches a number, display "You are correct", otherwise display "Try again - you may have three guesses" and add one to a variable called numGuesses.

```
 if (guess==number)
        alert("correct");
else
{
        alert("Try again - you may have three guesses");
        numGuesses+= 1;
}
```

36

36

SWITCH – SHORTHAND FOR IF

- ```
  switch(expression)
  {
     case value1: do this; break;
     case value2: do that; break;
     default: do the other thing; break;
  }
  ```
- *break* keeps it from "falling through"
- *default* placed at the end and is optional

37

37

QUESTIONS, QUESTIONS …

- Use "else if" to keep asking questions

```
if (mm<4)
            alert("You have a few m&m's");
else if (mm< 10)
     alert("You have a handful of m&m's");
else if (mm< 30)
     alert("You have several m&m's");
else
{
     alert("You have a lot of m&m's");
     alert("Can I have some?");
}
```

38

38
```

PROBLEM: SOLVE THE FOLLOWING USING A SWITCH STATEMENT

- You have a variable called coin and a variable called value.
- If coin is “nickel”, value is 5
- If coin is “dime”, value is 10
- If coin is “quarter” value is 25
- Create a switch statement that determines the value of the coin

```
switch(coin)
{
    case “nickel” :
        val = 5;
        break;
    case “dime” :
        val = 10;
        break;
    case “quarter” :
        val = 25;
}
```

39

39

LOOPS: WHEN ONCE IS NOT ENOUGH!

- A loop is when your program does something over, and over, and over and over …
- Two types of loops: Counting and Waiting

In a **counting loop**, you do something for a specified number of times.
For example - display “Hello World” on the screen 10 times.
Or maybe, move two squares - 4 times.

In a **waiting loop**, you do something until something else happens.
For example - get numbers from the user until the user enters: -1
Or, move a robot forward 1 inch at a time-until a boundary is detected.

*In Javascript, a **for** loop is optimized for counting and a **while** loop is optimized for waiting.*

40

40

FOR LOOP

- Three parts built-in to the header:
  - **initialization**- statement that occurs prior to any iteration
  - **test** - conditional expression that is evaluated at the beginning of each iteration. When expression evaluates to false, exit the loop
  - **update** - statement that occurs at the end of each iteration. Often used to update a counter.
- *Any or all of these can be omitted.*

41

41

GENERAL FORMAT

```
for (init ; test ; update)
{
 // loop statement(s)
}
```

Display the numbers: 1 to 10

```
for (n=1; n<=10; n++)
  alert(n);
```

Notes:

- Do not put a semicolon at the end of a "for" header
- for (;;) is an intentional infinite loop
- Omit the initialization and update segments to emulate a *while* loop.

42

42

WHILE LOOP

- Continue to *iterate* as long as the conditional expression is true
- Alternate form: do-while allows for at least one iteration through the loop

```
while (test)
{
  // loop statement(s)
}

do  {
  // loop statement(s)
} while (test);
```

43

43

EXAMPLE: DISPLAY THE NUMBERS FROM 1 TO 10

```
n = 1;                  // initialization
while (n<=10)           // test
{
 alert(n);
 n++;                   // update
}
```

Notes:

- Do not put a semicolon at the end of a while header *except* in a do-while construct.
- while(true) is an intentional infinite loop

44

44

FIGURE IT OUT

How many times will "hello" be printed?

```
for (i=1; i<=5; i++)
  document.write( "hello" );
while (i<= 5)
  document.write( "hello" );
```

How many times will "hello" be printed?

```
for (i=2; i<=5; i++);
  document.write( "hello" );
```

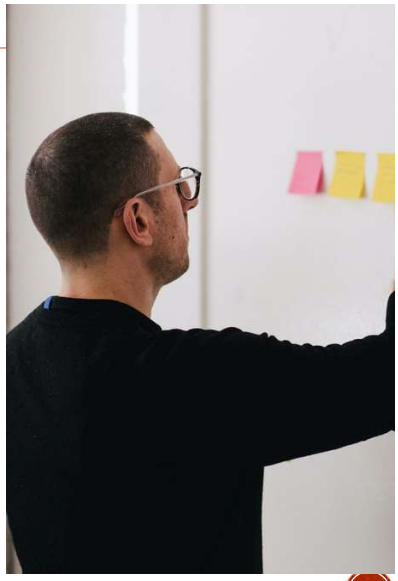
45

45

SENTINEL OR ACCUMULATED VALUES

- Sometimes a loop is used to gather information across iterations of the loop.
- In this case a sentinel or accumulated value is used.
- Scenario – find the tallest of 5 kids
- Example: add the even numbers from 2 – 20

```
sum=0;              //sum is the sentinel value
for (n=2;n<=20;n++)
  sum+= n;
alert(“ the sum is:”+ sum);
```



46

46

BREAK AND CONTINUE

- Break and continue statements provide additional control for directing the flow through a loop.

Break exits the loop

- Generally used as part of an if construct.
- Used to provide an alternate exit from the loop
- Use carefully to avoid unreadable code.

Continue ends the current iteration of the loop.

- In a while loop, continue goes directly to the test
- In a for loop, continue goes directly to the update.

47

47

BREAK AND CONTINUE

```
for (i=1; i<=5; i++)
{
  if (i==2) continue;
  document.write( "hello" );
  if (i==4) break;
}
```

```
i=0;
while (i<=5)
{
  if (i==2) continue;
  document.write( "hello"
  );
  if (i==4) break;
}
```

48

EXAMPLE

```
//this loop demonstrates two exit points
 while(true)
 {
       if (x == 10)  break;        //get out here
       if (x == 20)  break;        //or get out here
       x++;
 }
// special case for first iteration
 for (x=0; x<10; x++)
 {
             document.write(x);
             if (x==0)
                    continue;
       // more loop statements can go here
 }
```

49

49

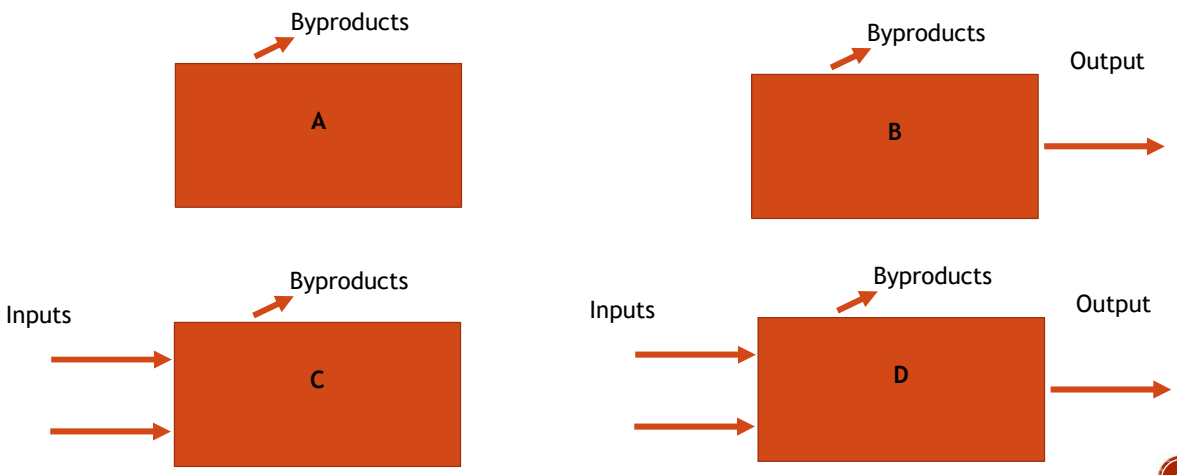
FUNCTIONS

- A function is a named set of tasks that are not executed until the function is *called*
- A function can be *anonymous* (no name) and are used as a parameter to another function.
- Functions can have inputs, outputs, and byproducts

50

50

BLACK BOX MODEL FOR FUNCTIONS



51

51

JAVASCRIPT FUNCTION SYNTAX

```
function func_name(arg_name1, arg_name2)
{
    // meat of function goes here
}
```

```
//Define the function
function foo(n, m)
{
   alert(n + m);
}
```

```
//Call the function
foo();
```

52

52

SIMPLE FUNCTION

Function definition (Type A)

```
function add1()
{
  var a = 2, b = 3, sum;
  sum = a + b;
 alert ("The sum is: " + sum);
}
```

Calling the function

```
add1();
```

53

53

GIVING BACK A VALUE - RETURN

Function definition (Type B)

```
function add2()
{
 var a = 2, b = 3, sum;
 sum = a + b;
 return sum;
}
```

Calling the function

```
var sum;
sum = add2();
alert("The sum is: " + sum);
```

54

54

PROVIDING ARGUMENTS (PARAMETERS)

Function definition (Type C)

```
function add3(a,b)
{
 var sum = a + b;
 alert(sum)
}
```

Calling the function

```
add3(2,3);
```

55

55

PROVIDING ARGUMENTS AND OUTPUT

Function definition (Type D)

```
function add4(a,b)
{
 var sum = a + b;
 return sum;
}
```

Calling the function

```
var sum;
sum = add4(2,3);
alert ("The sum is: " + sum);
```

56

56

ARROW FUNCTIONS

- Use the arrow as a shortcut to define and then call a function
- Assume you want a simple function as follows:
  function hello() { return “Hey there!”; }
  - Call the function using: hello()
- Using an arrow function:
  hello = () => {return "Hello World!“};
- Simplfy further to:
  hello = () => "Hello World!";
  - Call the function using: hello()
- Add parameters:
  hello = (num) => "Two times " + num + " is " + 2 * num;

57

EXAMPLE: USE AN ARROW FUNCTION AS A PARAMETER

```
add = (a,b) => a+b;
sub = (a,b) => a-b;
function operate (a, b, op)
{
        return op(a,b);
}
alert(operate (4,7, add));     //what is displayed?
```

58

CALLBACK FUNCTIONS

- A function can be called automatically when something else has completed.
- Example: setTimeout

```
setTimeout(function() {
        alert("a second just passed!") }, 1000)
)
```

- The function doesn't need a name because it won't be called again (it's an anonymous function)

- setTimeout(()=>alert("one second") , 1000)

59

59

JAVASCRIPT OBJECT LIBRARY (DOM)

- Properties
  - Characteristics, State
- Methods
  - Things the object can do
    These will require parenthesis because they area function
- Events
  - The object can respond to events

- Use "new" to instance an object (sometimes there are shortcuts without *new*)
- *new* will call the constructor for the object
- Use the dot notation to call a method or access a property using an object
  arr = new Array()
  count = arr.length

60

60

THE MATH OBJECT: HELPFUL METHODS

- Math.random()
  - Returns a number between 0 and 1. Multiply it to get a larger range
  - Example- random number from 0 to 5
    `n = Math.random() * 5;`
  - Example- random number from 1 to 20
    `n = Math.random() * 19 + 1;`
- `Math.ceil()  Math.floor()`
- `Math.max ()  Math.min()`

61

61

THE STRING OBJECT

- Easily instanced using assignment to a quoted string.
- Helpful properties:
  - length
- Helpful methods:
  - charAt
  - indexOf / lastIndexOf
  - substr / substring
  - toLowerCase / toUpperCase
  - split

62

62

STRING

- length — number of characters in a string
- charAt() — returns the character at the specified index
- concat() — joins two or more strings, and returns a copy of the joined strings
- indexOf() — returns the position of the first occurrence of a specified string
- lastIndexOf() — returns the position of the last occurrence of a string
- slice() — extracts a part of a string and returns a new string
- split() — splits a string into an array of substrings
- substr() — gets a substring defined by a start position and a number of characters
- substring() — gets a substring defined by a start and end index
- toLowerCase() — returns the string in lower case
- toUpperCase() — returns the string in uppercase

63

63

STRING EXAMPLE

- Display each character of a string with an asterist (*) between each character

```
s = "I am a string";
for (n=0; n<s.length;n++)
  document.write (s.charAt(n) + " * " );
```

64

64

DATE

Given: d = new Date();

- d.getDate() Returns day of the month (1-31)
- d.getDay() Returns day of the week (0-6)
- d.getFullYear() Returns the year (four digits)
- d.getHours() Returns the hour (0-23)
- d.getMinutes() Returns the minutes (0-59)
- d.getMonth() Returns the month (0-11)
- d.getSeconds() Returns the seconds (0-59)