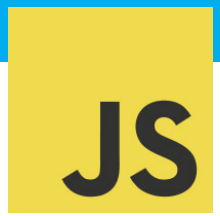

What is it?

- Javascript is a high-level interpreted programming language.
- One of the core technologies of the World Wide Web.
- Enables you to create dynamically updating content, control multimedia, animate images, and more.
- Requires a program that can read it and an environment.
- Is standardized by ecma international.
- Has been around since mid-1990s.
- Can be used client-side or server-side.

Who should learn it?

- Web developers.
  - Front-end, back-end or full-stack.

- Mobile-App Developers
- Desktop-App Developers
- Game Developers

Why?

- Average Salary is $78,000
- High Demand - top 5 programming langauges
- Easy to learn
- Easy to begin
- Large community for support
- Can be Fun

Where to write Javascript Code

- Plain Text Editors
  - NotePad (Windows) TextEdit (Mac)
- Integrated Development Environment
  - VSCode, Brackets, DW, Atom, Eclipse
- Browser developer tools Console
  - Chrome, Firefox, Edge

Basic Rules

- Extra white spaces are ignored outside of quotation marks.
  - `let  fruit  =  "apple";` *same as* `let fruit= "apple";`
- Instructions end with a semi-colon (`;`).
  - `let age = 234; sayHello(); alert("Hi");`
- Order *usually* matters.
  - `let x = 30; console.log(x);` returns: 30
  - `console.log(y); var y = 10;` returns: undefined
- Javascript can be written inside an HTML file or saved as a separate Javascript file.
  - If placed inside an HTML file, it must be within a **script element** inside the **body or head.**
  - If placed in a separate javascript file, you must insert it into the page through the **src** attribute of a **script element.**

Placing Javascript in an HTML File

- Directly Inside the HTML File
  - `<html>`
  - `<head>`
  - *here* `<script>let a = "apple";</script>`
  - `</head>`
  - `<body>`
  - *or here* `<script>let b = "banana";</script>`
  - `</body>`
  - `</html>`

Placing Javascript in an HTML File

- In a separate file
  - `<html>`
  - `<head>`
  - *here* `<script src="mycode.js"></script>`
  - `</head>`
  - `<body>`
  - *or here* `<script src="mycode.js"></script>`
  - `</body>`
  - `</html>`

Variables

- Containers which hold reusable data.
- Basic unit of storage in a program.
- Declared with `var`, `let`, or `const` and followed by a name.
  - `let` and `const` were standardized in ES2015 (ES6).
  - `var name1 = "Jeff";`
  - `let name2 = "Jack";`
  - `const name3 = "Joe";`

    Note: There are differences between var, let and const.
- Value is assigned with `=`
- The value of a variable can be changed after it is assigned unless it is declared with the `const` keyword.

Naming Variables

- Variable names can contain **only the following**:
  - letters (`a-z, A-Z`)
  - whole numbers (`1 - 9`)
  - Underscores (`_`)
  - dollar sign (`$`)
- Variable names **cannot** begin with numbers.
- Variable names **cannot** contain spaces.
- Variable names **are case sensitive.**
  - `var country` and `var Country` are 2 different variables.
- Some words are reserved and cannot be used as names for variables.

Reserved Words

- The following words cannot be used as variable names

abstract	boolean	break	byte	case	catch
char	class	const	continue	debugger	default
delete	do	double	else	enum	export
extends	false	final	finally	float	for
function	goto	if	implements	import	in
instanceof	int	interface	long	native	new
null	package	rivate	protected	public	return
short	static	super	switch	synchronized	this
throw	throws	transient	true	try	typeof
var	void	volatile	while	with	

Data Types

- Variables in javascript can contain any data type.
- Data types can be divided into 2 categories:
  - Primitives and Objects
- Primitives are basic data and contain values.
- There are currently 6 Primitive data types in Javascript:
  - string, number, boolean, null, undefined, and symbol
- Everything else is a type of Object.
  - Object, Array, Function, Date, Math, Regular Expressions
- Primitives *contain* data, Objects *refer* to data.
- Primitives with the same value are equal but Objects with the same value are different.
- Objects can refer to other objects.
- The data type of a variable can change.

String

- A string in Javascript is a series of characters.
- Strings are wrapped in single or double quotes.
  - "Bob", "Airplane", "1,2,3,4,5", "This is also a string"
  - 'Hello', 'One 2 Three', 'a@#$%^&*)'
- Quotes inside a string can't match the surrounding string unless you use the backslash escape character.
  - "There's something under the bed."
  - "My \"friends\" forgot my birthday again."
  - 'What\'s up doc?'
- Strings can be combined or ***concatenated*** with (`+`)
  - `let a = "Refrig"; let b = "erator";`
  - `let c = a+b; console.log(c):` *"Refrigerator"*

Number

- Javascript has one standard data type for all numbers.
  - BigInt was recently introduced for very large numbers.
- Numbers are not wrapped in quotes like strings.
  - Numbers wrapped in quotes will become strings.
- Numbers can be accurately calculated within the range of $-(2^{53} - 1)$ to $2^{53} - 1$.
- By default, JavaScript displays numbers as base 10 decimals.
  - Can also display in Hexadecimal, Octal, and Binary.
- Arithmetic operators can be used to do math with numbers.
  - +, -, *, /, %
- Increment and Decrement can increase or decrease by 1.
  - ++ , --
- +=, -=, *=, /=, and %= will calculate and assign values.

Boolean

- A boolean can contain only 1 of 2 possible values.
  - True or False
- Useful when you need only 2 possible values:
  - Yes or No, On or Off, True or False
- Variables with values are true.
- Variables without values are false.
- 0 is also false.
- Comparing with == or === will return a Boolean.

Null and Undefined

- Null and undefined are similar but not exactly the same.
- **Null** means Nil, empty or non-existant.
- When a variable is declared but not given a value, javascript automatically assigns the **undefined** value.
- Null is never automatically assigned by javascript.
- Null has to be assigned by the programmer.

Symbols

- The Symbol data type is fairly new, introduced in ES6.
  - *Every symbol value returned from Symbol() is unique. A symbol value may be used as an identifier for object properties; this is the data type's only purpose.*

    -MDN web docs
- Basically gives you a random unique value.
- Unlike other primitives, must be declared with `Symbol( )`

Objects

- All non-primitives are **objects.**
- They are assigned with `var`, `let`, or `const` the same as primitives but their values can be numerous other variables.
- Variables in objects are called **properties**.
- Each **property** is stored as a **key/value pair**.
- The **key** is the variable name.
- The key's value is separated by a colon: `key:value`
- Key/value pairs are separated by commas (`,`) and all of the key/value pairs of an object are wrapped in curly braces (`{}`).
- Keys don't need to be wrapped in quotes.
- Values that are strings must be wrapped in quotes.
    - `let car = {make:'honda', model:'civic', year:2001}`
    - `let topSpeed = {car:250, train:300, plane:3000}`

Objects 2

- You can access a property with a dot or square brackets.
- Using **dot notation**, properties can't have quotes.
- With square bracket notation, they must have quotes.
  - `car.model` and `car['model']` return the same value
- You can assign or modify object properties using dot notation or square brackets.
  - `let car.model = 'accord';` or `car['model'] = accord.`
- You can create an empty object and assign values later.
- Unlike primitives, objects are not compared by value.
  - `var num1 = 3; var num2 = 3; num2 === num2;` returns true
  - `var obj1 = [1,2,3], var obj2 = [1,2,3]`
  - `obj1 === obj2;` returns false

Arrays

- Javascript **Arrays** are special objects that store values and automatically assign index numbers as their keys.
- You can define an array by using the var, let, or const keywords just like primitive data types.
- The values can be assigned using square brackets, with each value separated by a comma.
  - ex: `let arr1 = [10,20,30,40,50]`
- Each value in the array has an index number, starting with 0
- The index number is a key that is automatically assigned.
- Values are accessed by index numbers enclosed in brackets.
  - In the above example, `arr1[0]` is 10, `arr1[1]` is 20.
- You can create an empty array and assign values later.
  - `let arr2 = [];`
  - `arr2[0] = 'Dog'; arr2[1] = 'cat';`

Arrays 2

- Arrays can also be created using the 'new' keyword.
  - `let arr3 = new Array('red', 'green', 'blue');`
- Arrays have built in properties and **methods** that they inherit from the Array prototype.

  A method is a property with a function definition.
- Use the **length *property*** to get the number of items.
  - Referring to the array above, `arr3.length;` gives you 3.
- The **push *method*** adds values to the end of an array
  - ex: `arr3.push('yellow');`
  - `arr3;` should now return ('red', 'green', 'blue', 'yellow');
  - The values you are adding are enclosed in parentheses.
- `pop()` will remove the last item.
- `concat()` will combine arrays and return a new one.

Array Methods

concat()	Merge two or more arrays, and returns a new array.
copyWithin()	Copies part of an array to another location in the same array and returns it.
entries()	Returns a key/value pair Array Iteration Object.
every()	Checks if every element in an array passes a test in a testing function.
fill()	Fill the elements in an array with a static value.
filter()	Creates a new array with all elements that pass the test in a testing function.
find()	Returns the value of the first element in an array that pass the test in a testing function.
findIndex()	Returns the index of the first element in an array that pass the test in a testing function.
forEach()	Calls a function once for each array element.
from()	Creates an array from an object.
includes()	Determines whether an array includes a certain element.
indexOf()	Search the array for an element and returns its first index.
isArray()	Determines whether the passed value is an array.
join()	Joins all elements of an array into a string.
keys()	Returns a Array Iteration Object, containing the keys of the original array.

lastIndexOf()	Search the array for an element, starting at the end, and returns its last index.
map()	Creates a new array with the results of calling a function for each array element.
pop()	Removes the last element from an array, and returns that element.
push()	Adds one or more elements to the end of an array, and returns the array's new length.
reduce()	Reduce the values of an array to a single value (from left-to-right).
reduceRight()	Reduce the values of an array to a single value (from right-to-left).
reverse()	Reverses the order of the elements in an array.
shift()	Removes the first element from an array, and returns that element.
slice()	Selects a part of an array, and returns the new array.
some()	Checks if any of the elements in an array passes the test in a testing function.
sort()	Sorts the elements of an array.
splice()	Adds/Removes elements from an array.
toString()	Converts an array to a string, and returns the result.
unshift()	Adds new elements to the beginning of an array, and returns the array's new length.
values()	Returns a Array Iteration Object, containing the values of the original array.

Functions

- Functions are objects that can be called to perform a task.
- They are defined with the '**function**' key-word followed by a name, followed by parentheses `()`, followed by curly braces `{}` that will contain the code to be executed.
    - ```
      function sayHi(){
          console.log('Hi!');
      }
      ```
- The code in the curly brackets will not execute until the function is ***called***.
- You call a function when you type its name with parentheses.
    - `sayHi()` will now log 'Hi!' into the console.
      ```

Functions 2

- When creating a function, you can optionally require input data to be used inside the function.
- To allow additional input for the function, placeholders are put within the parentheses after the function name.
    - `function sayMyName(name){console.log(name)}`
- The placeholder is called a **parameter.**
- The value that replaced the placeholder when calling the function is called the **argument.**
- A functions can have many parameters and arguments.

Conditional Statements

- A conditional statement is a set of rules performed if a certain condition is met.
- The condition evaluates to a **boolean** value: **True** or **False**.
- Javascript has the following **Comparison Operators** to test if a condition is either true or false:
  - `==` **(equal to)**
  - `===` **(equal value and equal type)**
  - `!=` **(not equal)**
  - `!==` **(not equal value or not equal type)**
  - `>` **(greater than)**
  - `<` **(less than)**
  - `>=` **(greater than or equal)**
  - `<=` **(less than or equal)**

Conditional Statements 2

- The **if statement** allows you to write code that will only execute if the condition provided is true.
- The syntax for an if statement is similar to declaring a function.

```
if(condition){
    code that will execute if condition is true;
}
```

- If the condition is false, the code will not execute.
- Use **else** to provide instructions in case the condition is false.

```
if(age > 21){
    alert("Go ahead")
}else{
    alert("Beat it")
}
```

Conditional Statements 3

- Use **else if** to test an alternate condition and run different code if the alternate condition is true.
  - ```
    if(sides === 3){
        alert("triangle");
    }else if(sides === 4){
        alert("quad");
    }else{
        alert("ngon");
    }
    ```
    ```

Conditional Statements 4

An if statement can have multiple else if's.

```
if(year % 4 != 0){
    days = 365;
}else if(year % 100 != 0){
    days = 366;
}else if(year % 400 !=0){
    days = 365;
}else{
    days = 366
}
```

An if statement can contain other if statements.

```
if(year % 4 == 0){
    if(year % 100 != 0){
        days = 366;
    }else if(year % 400 == 0){
        days = 366;
    }else{
        days = 365;
    }
}else{
    days = 365;
}
```

Conditional Statements 5

- Javascript has **logical operators** that can be used to determine the logic between variables or values:
  - `&&` (and)
  - `||` (or)
  - `!` (not)
- `&&` will test if all of the conditions are true.
- `||` will test if at least one of the conditions is true.
- `!` can be used to test if a condition is not true.
  - ```
    if(age > 21 && expDate > today){
       alert("You may enter")
    }
    ```
  - ```
    if(!guestList.includes(name) || tip < 99){
       alert("Beat it!")
    }
    ```

Conditional Statements 6

An alternative to if statements with many else ifs is a **switch statement.**

```
switch(new Date().getMonth()){
    case 0:
        month = "January";
        break;
    case 1:
        month = "February";
        break;
    case 2:
        month = "March";
        break;
    case 3:
        month = "April";
        break;
    case 4:
        month = "May";
        break;
    case 5:
        month = "June";
        break;
    case 6:
        month = "July";
        break;
    case 7:
        month = "August";
        break;
    case 8:
        month = "September";
        break;
    case 9:
        month = "October";
        break;
    case 10:
        month = "November";
        break;
    case 11:
        month = "December";
        break;
}
```

Conditional Statements 7

- The `break` command is necessary to stop the rest of the code from executing.
- If no condition is met, a `default` can be provided.
    -

```
switch(today){
    case "Friday":
        alert("Almost the weekend!");
        break;
    case "Saturday":
        alert("It's the weekend!");
        break;
    case "Sunday":
        alert("I have work tomorrow.")
        break;
    default:
        alert("Cant wait til Saturday");
}
```

Conditional Statements 8

- Javascript also contains a **conditional operator**, also called a **ternary operator**.

  - `variablename = (condition) ? value1:value2`
  - `Or`
  - `(condition) ? variablename = value1:value2`
  - `(5>3)?x = "a":"b";` x = "a";
  - `(5>6)?x = "a":"b";` x = "b";

Math Object

Javascript's built in **Math** object has many useful properties.

E	Returns Euler's number (approx. 2.718)
LN2	Returns the natural logarithm of 2 (approx. 0.693)
LN10	Returns the natural logarithm of 10 (approx. 2.302)
LOG2E	Returns the base-2 logarithm of E (approx. 1.442)
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)
PI	Returns PI (approx. 3.14)
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)
SQRT2	Returns the square root of 2 (approx. 1.414)
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
acosh(x)	Returns the hyperbolic arccosine of x
asin(x)	Returns the arcsine of x, in radians
asinh(x)	Returns the hyperbolic arcsine of x
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
atan2(y, x)	Returns the arctangent of the quotient of its arguments
atanh(x)	Returns the hyperbolic arctangent of x
cbrt(x)	Returns the cubic root of x

ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
cosh(x)	Returns the hyperbolic cosine of x
exp(x)	Returns the value of Ex
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x, y, z, ..., n)	Returns the number with the highest value
min(x, y, z, ..., n)	Returns the number with the lowest value
pow(x, y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sinh(x)	Returns the hyperbolic sine of x
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle
tanh(x)	Returns the hyperbolic tangent of a number
trunc(x)	Returns the integer part of a number (x)

Date Object

getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year
getHours()	Returns the hour (from 0-23)
getMilliseconds()	Returns the milliseconds (from 0-999)
getMinutes()	Returns the minutes (from 0-59)
getMonth()	Returns the month (from 0-11)
getSeconds()	Returns the seconds (from 0-59)
getTime()	Returns the number of milliseconds since midnight Jan 1 1970, and a specified date
getTimezoneOffset()	Returns the time difference between UTC time and local time, in minutes
getUTCDate()	Returns the day of the month, according to universal time (from 1-31)
getUTCDay()	Returns the day of the week, according to universal time (from 0-6)
getUTCFullYear()	Returns the year, according to universal time
getUTCHours()	Returns the hour, according to universal time (from 0-23)

getUTCMilliseconds()	Returns the milliseconds, according to universal time (from 0-999)
getUTCMinutes()	Returns the minutes, according to universal time (from 0-59)
getUTCMonth()	Returns the month, according to universal time (from 0-11)
getUTCSeconds()	Returns the seconds, according to universal time (from 0-59)
getYear()	Deprecated. Use the getFullYear() method instead
now()	Returns the number of milliseconds since midnight Jan 1, 1970
parse()	Parses a date string and returns the number of milliseconds since January 1, 1970
setDate()	Sets the day of the month of a date object
setFullYear()	Sets the year of a date object
setHours()	Sets the hour of a date object
setMilliseconds()	Sets the milliseconds of a date object
setMinutes()	Set the minutes of a date object
setMonth()	Sets the month of a date object

Date Object (continued)

setSeconds()	Sets the seconds of a date object
setTime()	Sets a date to a specified number of milliseconds after/ before January 1, 1970
setUTCDate()	Sets the day of the month of a date object, according to universal time
setUTCFullYear()	Sets the year of a date object, according to universal time
setUTCHours()	Sets the hour of a date object, according to universal time
setUTCMilliseconds()	Sets the milliseconds of a date object, according to universal time
setUTCMinutes()	Set the minutes of a date object, according to universal time
setUTCMonth()	Sets the month of a date object, according to universal time
setUTCSeconds()	Set the seconds of a date object, according to universal time
setYear()	Deprecated. Use the setFullYear() method instead
toDateString()	Converts the date portion of a Date object into a readable string

toGMTString()	Deprecated. Use the toUTCString() method instead
toISOString()	Returns the date as a string, using the ISO standard
toJSON()	Returns the date as a string, formatted as a JSON date
toLocaleDateString()	Returns the date portion of a Date object as a string, using locale conventions
toLocaleTimeString()	Returns the time portion of a Date object as a string, using locale conventions
toLocaleString()	Converts a Date object to a string, using locale conventions
toString()	Converts a Date object to a string
toTimeString()	Converts the time portion of a Date object to a string
toUTCString()	Converts a Date object to a string, according to universal time
UTC()	Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time
valueOf()	Returns the primitive value of a Date object

Loops

- Javascript **loops** allow you to repeat a block of code until a certain condition is met.
- There are a few different types of loops in Javascript:
  - **For, While, Do While, For In, and For Of, For Each**
- Loops can take different arguments and output different results each time with the same code block.

For Loop

- The **for loop** lets repeats a block of code until a specified condition evaluates to false.
- There are 3 statements in the for loop before the code block:
  - The **initial expression** initializes the counter variable.
  - Statement 2 defines the **condition** that must be true in order for the loop to run.
  - Statement 3, the **increment Expresson**, will increase or decrease the value each time the code block is executed.

```
for ([initialExpression]; [condition];
[incrementExpression])
```

```
  Statement
```

- To console log numbers 1 through 5

```
for(i = 1; i <= 5; i++){
   console.log(i);
}
```

For Loop 2

- The statements at the beginning of the for loop are optional. The initial expression can be created before the for loop, the increment can happen inside the loop but leaving the condition out will lead to an infinite loop and possible crash.
    - ```
      let i = 0;
      for(;i<=10;){
          console.log(i);
          i++;
      }
      ```

Notice the semicolons are still there. Without them you will get an error.

- You can break out of a loop anytime with `break;`.
- You can skip an iteration with `continue;`.
      ```

While Loop

- The **while loop** is an alternative way to loop in javascript.
- While loops consist of a condition and a block of code that executes as long as the condition is true.
    - ```
      while(a <= 10){
          console.log(a);
          a++;
      }
      ```
- Again you can use `break;` and `continue;` to stop or skip.
      ```

For in

- The **for in** loop is used to iterate through all properties of an object using a variable. The specified statement is executed for each distinct property.
- syntax:

```
for(variable in object){
   statement;
}
```

- ex:

```
let car = {make:"Ford", model:"GT", color:"Grey"}
for(x in car){
   console.log(x);
}
```

- The above logs the property names: make, model, color.

For of

- Similar to the for in loop but...
- The **for of** loop is used to iterate through all property *values* of an *iterable* object using a variable. The specified statement is executed for each distinct value.
- syntax:
  - ```
    for(variable of object){
        statement;
    }
    ```
- ex:
  - ```
    let cars = {"Honda", "Ford", "BMW"}
    for(x of car){
        console.log(x);
    }
    ```
- The above logs the array values: "Honda", "Ford", "BMW".

For Each

- Unlike the previous loops, **for each** is an **array method** that can be used to call a function for each array element.
- syntax:
  - ```
    array1.forEach(function(element) {
      console.log(element);
    });
    ```
    ```

OOP

Object Oriented Programming

- Style of programming.
- DRY: Don't repeat yourself.
- Encapsulate.
- Provide Namespaces.
- Represent everything as objects.
- Use factory functions or classes to create similar objects.
- Use inheritance to create objects with the same properties.

Factory Function

- A **factory function** is a function that returns a new object.
- Including parameters allows the creation of objects with different properties.
- To create a factory function, define a function with a return statement followed by an object.
    - ```
      function createPerson(name, age, gender){
          return{
              name:name;
              age:age;
              gender:gender;
              sayHello:function(){
                  alert("hello");
              }
          }
      }
      ```
      ```

Factory Function 2

- Call the factory function to create a new object.
- Set it to a variable to save it.
  - ```
    let Bob = createPerson("Bob", 50, "M");
    Bob.sayHello();
    ```
  - The above should alert "Hello"
    ```

Constructor Function

- A **constructor function** is a function that creates objects with the use of the **new** keyword.
- The **this** keyword is used in the constructor function to assign properties and methods.

```
function Person(name, age, gender){
    this.name = name;
    this.age = age;
    this.gender = gender;
    this.sayHello = function(){
        alert("hello");
    }
}
```

```
let Jack = new Person("Jack", 30, "M");
```

ES6 Classes

- Classes were introduced in ES6.
- Many programming languages have classes.
- Looks like a constructor function wrapped in a function.

```
class Person{
    constructor (name, age, gender) {
        this.name = name;
        this.age = age;
        this.gender = gender;
    }
    sayHello(){
        alert("hello");
    }
}
```

```
let Jack = new Person("Jack", 30, "M");
```

Inheritance

- An advantage of classes is that they make inheritance easy.
- To create a class with the same properties and methods as another class, use the **extends** keyword.
- You can change an existing method by reassigning it.
- The following inherits from the Person class created earlier.

```
class FilipinoPerson extends Person{
    sayHello(){
        alert('Kamusta. Ang pangalan ko '+this.name)
    }
}
```

```
let jennifer = new
FilipinoPerson('Jennifer',21,'F');

jennifer.sayHello();
```

- Try it yourself. Create a Person class and extend it.

HTML Canvas

- Used to draw graphics on a webpage using scripting.
- The canvas is the container.
- The canvas object has different methods for drawing paths, boxes, circles, and adding images.
- Objects on the canvas can be animated.
- The canvas can have events attached to it for interactivity.