**JavaScript Programming**

Table Of Contents

1. JS Introduction
2. JS Console – Debugging, Testing, Logging
3. JS Statements
4. JS Expression
5. JS Keywords
6. JS Comments
7. JS Case Sensitives
8. JS Values
9. JS Errors
10. JS Camel Case
11. JS Identifiers
12. JS Variables
13. JS Operators
14. JS Data Types
15. JS String
16. JS Numbers
17. JS BigInt
18. JS Boolean
19. JS Symbol
20. Js Function
21. JS Execution Context
22. JS Event Loop
23. JS Asynchronous
24. JS Scope
25. JS Truth Taling
26. JS Global Codes
27. **JS Introduction**
28. *JavaScript is one of the 3 languages all web developers must learn*
29. *JavaScript was invented by Brenan Eich in 1995.*
30. *ECMAScript is the official name of the JavaScript.*
31. *JavaScript is designed by c++ language.*
32. **JS Console - Debugging, Testing, Logging**
33. *console.log(loggedCode)*
34. *console.error(errorCode)*
35. *console.dir(loggedCodeDetails)*
36. **JS Statements**
37. *A JavaScript statement is a line or block of code that instructs the JavaScript interpreter or browser engine to perform a specific action or operation.*
38. *JavaScript statements are separerated(not closed) by semicolon (;).*
39. *JavaScript statements are composed of: values, operators, expressions, keywords,comments, variables.*
40. **JS Expressions**
41. *An expression is a combination of values, variables, and operators, which computes to value.*
42. *Js evalutes expressions from left to right.*[*See here*](#_toc162)
43. **JS Keywords**
44. *JavaScript keywords are used to indentify actions to be performed.*
45. *Keywords are called reserved words*
46. **JS Comments**
47. */\*comments\*/*
48. *//comments*
49. **JS Case Sensitive**
50. *all javascript identifiers are case sensitive.*
51. **JS Values**
52. *number, string, boolean, undefined, null, symbol, object, array, function, NaN, Infinity, -Infinity*
53. *let uname = “Sobuj”*
54. **JS Errors**
55. *js main errors:*
56. *syntax error: syntax errors occur when the code violates the language’s grammatical rules.*
57. *runtime error: runtime errors occur during the execution of a program when something unexpected happens.*
58. *this error handling using* [*try..catch*](#_toc140) *block.*
59. *logical(bug) error: logical errors are the most challenging to identify and fix because they don’t cause the program to crash or throw errors. Instead, they lead to incorrect program behavior or unexpected result.*
60. *js specific errors:*
61. *reference error: reference error is a sepecific type of runtime error that occurs when you try to access a variable of function that has not been declared or is not in scope.*
62. *type error: type error is a specific type of runtime error that occurs when an operations is performed on a value that is of an inappropriate or unexpected data type.*
63. **JS Camel Case**
64. *javascript programmers tend to use camel case that starts with lowercase letter.*
65. **JS Identifiers/Names**
66. *idenfifiers are javascript names*
67. *identifiers rules:*
68. *identifiers must begin with letter(A-Za-z), dollar sign($) or underscore(\_).*
69. *Numbers are not allowed as the first character in identifiers.*
70. **JS Variables**
71. *javascript variables can be declared in 4 ways:*
72. *automatically: uname= “Sobuj”*
73. *var: var uname= “Sobuj”*
74. *let: let unme= “Sobuj”*
75. *const: const uname= “Sobuj”*
76. *the var keyword was used in all JavaScript code from 1995 to 2015.*
77. *the let and const keywords were added to javascript in 2015.*
78. *automatically threats as var.*
79. *var,let,const charactersistics:*
80. *var: redefined,reassigned,global-scope*
81. *let: not-redefined, reassigned, block-scope*
82. *const: not-redefined, not-reassigned, local-scope*
83. *var, let, const usecases:*
84. *always use const if the value should not be changed.*
85. *always use const if the type should not be changed(arrays and objects).*
86. *only use let if you can’t use const.*
87. *only use var if you must support old browser.*
88. **JS Operators**
89. *operators are symbols or special keywords that are used to perform various operations on values and variables.*
90. *Javascript operators:*
91. *arithmetic operators: addition(+),subtraction(-),multiplication(\*),*[*exponentiation(\*\*)*](#_toc148)*,diviision(/),modulus(%),increment(++),decrement(--)*
92. *assignment operators: assignment(=),addition assignment(+=),subtraction assignment(-=),multiplication assignment(\*=),division assignment(/=),modulus assignment(%=),exponentiation assignment(\*\*=);*
93. *comparison operators: equal to(==),equal value and type(===),not equal(!=),not equal and type(!==),greater than(>),less than(<),greater than or equal to(>=),less than or equal to(<=);*
94. *logical operators: logical and(&&),logical or(||), logical not(!);*
95. *type operators: typeof,* [*instanceof*](#_toc155) *;*
96. *ternary operator: conditon ? output : output ;*
97. **JS Data Types**
98. *js data types:*
99. *string,number,bigint,boolean,undefined,null,symbol,object*
100. *object data type can contain: object,array,date,function;*
101. *primitive vs reference data type:*
102. *primitive: primitive types are single values that can not be changed after declared.*
103. *Primitive data are store in a call stack memory.*
104. *String,number,bigint,boolean,null,undefined,symbol are primitive data type.*
105. *Call stack means every declare time create a new memory space to store primitive data.*
106. *Reference: reference types are more complex data types that represent collection of data or objects.reference types are mutable.*
107. *Object, function, array,date are reference data type.*
108. *Refernce data are store in a heap memory.*
109. *Heap memory means root declare data are always store in heap memory as single and identifiers store in call stack identifiers always refer heap memory data.*
110. **JS Strings**
111. *js strings is declared within the single or double quotes;*
112. *Escape characters: \’,\”,\\,\n,b;*
113. *string as object: new String(“”);*
114. *string methods:*
115. *charAt(index);charCodeAt(index);concat(str1,str2…);endsWith(‘character’,optional[serialNo]);startsWith(‘character’,optional[serialNo]);String.fromCharCode(97,97,97);includes(“str”,strtIndex);indexOf(‘str’,startIndex);lastIndexOf(‘str’,startIndex);localeCompare();match(reqx);padEnd(length,str);padStart(length,str);repeat();replace();replaceAll();search();slice(startIndex,endIndex);split(separeator,limit);startsWith();substr();substring();toLocaleLowerCase();toLocaleUpperCase(); toLowerCase();toUpperCase();toString();trim();trimEnd();trimStart();valueOf();*
116. *string properties:*
117. *length;constructor;prototype;*
118. **JS Numbers**
119. *js has only one type of number;*
120. *js numbers global: NaN,isNaN(),Infinity,-Infinity; ...more;*
121. *js numbers as object: new Number(123);*
122. *js numbers method:*
123. *toString(base); toLocaleString(“bn-BD”); toFixed(); toPrecision(totalDigits); toExponential(); valueOf();*
124. **JS BigInt**
125. *js bigint variables are used to store big integer values that are too big to be represented by a normal javascript number.*
126. *Let score=232333n; bounce=BigInt(123213123123213123123);*
127. *BigInt();*
128. **JS Boolean**
129. *a javascript boolean represents one of two values: true or false;*
130. *Boolean();*
131. *everything with a “value” is true:*
132. *100; -12; “something”; true;*
133. *everything without a “value” is false:*
134. *0; -0; “”; undefined; null; NaN; false;*
135. *js booleans as object: new Boolean();*
136. **JS Symbol**
137. *symbol is a built-in object whose constructor returns a symbol primitive also called a symbol value or just a symbol that’s guaranteed to be unique.*
138. *Symbol()*
139. *symbol as object property:{[Symbol()]: “good”}*
140. **JS Function**
141. *function declaration; function invoke; function call; function return; function parameter; function arguments; function defualt parameter; rest operator; function argumetns array of object; anonymus function; callback fuction; arrow function; self-invoking function; function closure;*
142. *js function property: arguments;name;length;*
143. *js function method: call();bind();apply();*
144. **JS Execution Context**
145. *js engline: v8(chrome);spidermonkey(mozila);javascript core(safari);chakra(ms edge);*
146. *type of execution:*
147. *interpretetion: line by line read and execute and easy to debug;*
148. *compilation: full code once time read then once time execute and hard to debug;*
149. *mixture of both: lline by line direct compile to machine code;*
150. *execution context: part by part code;*
151. *global execution context: first time will be loading state, then execute;*
152. *function execution context: second time will be loading,execution state;return from execution;*
153. *loading/creation sate:*
154. *create a global object/create a parameter or arguments object;create this object;allocate variable and function body identifiers in memory;variable value assign undefined;*
155. *execution state:*
156. *assign variable new value; function invoke wiil be call with new function execution context;*
157. *scope behaviour for LIFO or still not loading, not execute;*
158. *closure*
159. *uses call stack data structure;*
160. *LIFO structure;*
161. **JS Event Loop**
162. *event loop is a loop that continously check it’s any event whether available in that event queue, and check call stack is pending or avaialable;*
163. *Evet loop is bridges between callStack and event(callback) queue;*
164. *event loop is use asynchronous tasks;*
165. *asynchronous task are completed via second hand worker like web api/c++ api using callback function;*
166. *callback queue;render queue;microtask queue;*
167. **JS Asynchronous**
168. *must be before know: execution context; event loop; callback;call stack; callback queue; single-thread; blocking;no-blocking;cpu operation task; I/O operation task;*
169. *asynchronous programming refers to the ability to execute code in a non-blocking manner, allowing the program to continue running other tasks while waiting for certain operation to complete.*
170. *Use callback();*
171. *new Promise(callBack(res,rej)); Promise.resolve(); Promise.all([p1,p2]); Promise.race([p1,p2]);*
172. *async/await; try..catch block;*
173. **JS Scope**
174. *js variable are parent to child access not child to parent;*
175. *Kinds of scope:*
176. *global scope: variable declared outsided of any function or block are in the global scope.*
177. *Local scope: variable declared inside a function or block are in local scope.*
178. *Block scope: variable declared with let and const are block scope.*
179. *Function scope: variable declared withing a function are in function scope.*
180. *Nested scope: inner scopes can access variables outer scopes are in nested scope.*
181. **JS Truths Talking**
182. *comparing javascript two reference(objects) data types always return false;*
183. *Comparing javascript two primitive data types always return true;*
184. **JS Globals Codes**
185. *isNaN();isFinite();Number.isInteger();Number.isSafeInteger();parseInt();parseFloat();*
186. *Number.MAX\_SAFE\_INTEGER;Number.MIN\_SAFE\_INTEGER;Number.MAX\_VALUE;Number.MIN\_VALUE;Number.POSITIVE\_INFINITY;Number.NEGETIVE\_INFINITY;Number.NaN;*
187. *BigInt(number);Boolean(condition);Symbol();*
188. *new String();new Number(); new Boolean();*
189. *String.prototype; Number.prototype; Boolean.prototype;*
190. *new Promise(); Promise.all();Promise.race(); Promise.resolve();Promise.rejected();*