

```
1  Array : Array is collection of elements. Each element is combination of index
2  and value. Javascript is loosely typed script so we can store any type of values
3  in array.
4
5  <script>
6    var arr=[10,20,30,40,50,"scott"];
7    alert(arr[5]);
8    alert(arr);
9  </script>
10
11  push : using this function we can add an element at the end of arraya and it returns
12  total number of elements.
13
14  <script>
15    var arr=[10,20,30,40,50];
16    var rv=arr.push(60);
17    alert(arr);
18    alert(rv);
19  </script>
20
21  pop : To remove the last element of array and returns value of that element.
22
23  Ex:
24  ----
25
26  <script>
27    var arr=[10,20,30,40,50];
28    var rv=arr.pop();
29    alert(arr);
30    alert(rv);
31  </script>
32
33  shift : To remove the first element of array and returns value of that element.
34
35  <script>
36    var arr=[10,20,30,40,50];
37    var rv=arr.shift();
38    alert(arr);
39    alert(rv);
40  </script>
41
42  unshift() : Adds an element at the begining of array and returns total number
43  of elements.
44
45  <script>
46    var arr=[10,20,30,40,50];
47    var rv=arr.unshift(2);
48    alert(arr);
49    alert(rv);
50  </script>
51
52  -----
53
54  slice() : To get some part of array.
55  <script>
56    var arr=[10,20,30,40,50];
57    var narr=arr.slice(2,4);
58    alert(narr);
59  </script>
60
61  -----
62  splice() : To add/remove elements based on the index number.
63
64  <script>
65    var arr=[10,20,30,40,50];
66    arr.splice(1,0,111,222,333,444,'abc');
67    alert(arr);
68  </script>
69
```

```

70 -----
71 for..in : Runs a loop through the elements of array and returns index of those
72 elements.
73
74 Ex:
75 -----
76 <script>
77   var arr=[10,20,30,40,50];
78   for(var x in arr){
79     alert(arr[x]);
80   }
81 </script>
82
83 -----
84 for..of : Same as for..in but returns values of elements.
85
86 <script>
87   var arr=[10,20,30,40,50];
88   for(var x of arr){
89     alert(x);
90   }
91 </script>
92
93 -----
94 forEach() : It is combination of previous two functions holds both index and values. It
95 also executes a function for every iteration.
96
97 Ex:
98
99 <script>
100   var arr=[10,20,30,40,50];
101   arr.forEach(function(val,ind){
102     console.log(ind,val);
103   })
104 </script>
105
106 -----
107
108 map() : It is same as forEach to run a loop through the elements of array. Map returns
109 values. Total number of return values are equal to total number of elements.
110
111 <script>
112   var arr=[10,20,30,40,50];
113   var narr=arr.map(function(val,ind){
114     console.log(ind,val);
115     return val+ind;
116   })
117   console.log(narr);
118 </script>
119
120 -----
121
122 filter() : Runs a loop through the elements of array and returns the truth values.
123
124 <script>
125   var arr=[10,20,30,40,50];
126   var narr=arr.map(function(val,ind){
127     return val>30;
128   })
129   console.log(narr);
130 </script>
131
132 -----
133
134 split() : To split a string as array elements based on input value.
135
136 <script>
137   var str="welcomescott";
138   var narr=str.split("o");

```

```

137     alert(narr[2])
138 </script>
139 -----
140
141 join() : To join the elements of array as string based on input value.
142
143 <script>
144     var arr=["scott","amith","suresh"];
145     var str=arr.join("/");
146     console.log(str);
147 </script>
148
149 -----
150
151 Object : Object is collection of properties. Each property is combination of name and
value.
152
153 <script>
154     var obj={uname:"scott",city:"hyderabad"};
155     console.log(obj.uname);
156 </script>
157
158 Ex2 :
159
160 <script>
161     var obj=[{uname:"scott",city:"hyderabad"},{uname:"john",city:"chennai"}];
162     console.log(obj[1].uname);
163     console.log(obj[1].city);
164 </script>
165
166
167 Ex3:
168
169
170 <script>
171     var obj={uname:"scott",city:"hyd",stt:"Tg"};
172     for(var x in obj){
173         console.log(obj[x]);
174     }
175 </script>
176
177 Object.keys() : To get all the keys of an object in the form of array
178
179 <script>
180     var obj={uname:"scott",city:"hyd"}
181     var keys=Object.keys(obj);
182     console.log(keys);
183 </script>
184
185 Object.values() : To get all the values of an object as array
186
187 <script>
188     var obj={uname:"scott",city:"hyd"}
189     var values=Object.values(obj);
190     console.log(values);
191 </script>
192
193 --
194 Object.hasOwnProperty() : To check whether the specified property is available or not in
an object.
195
196 <script>
197 var obj={uname:"scott",city:"hyd"};
198 alert(obj.hasOwnProperty("state"));
199 </script>
200
201 -----
202 Remove duplicate elements from Array
203

```

```

204 Ex 1:
205
206 <script>
207 var arr=[10,20,30,10,40];
208 var obj={};
209 for(var i=0;i<arr.length;i++){
210     obj[arr[i]]=1;
211 }
212 console.log(obj);
213 var keys=Object.keys(obj);
214 console.log(keys);
215 </script>
216
217
218 -----
219
220 Ex 2:
221 <script>
222 var arr=[10,20,30,10,40];
223 var narr=new Set(arr);
224 console.log(narr);
225 </script>
226
227 -----
228
229 unname-scott
230 wife - name - sw1
231     child : sons - sw1c1,ss1c2  dau - sw1d1,sw1d2
232
233     name - sw2
234     child : sons - sw2s1      dau - sw2d1
235
236 Ex:
237
238 <script>
239 var ob={uname:"scott",wives:[{
240     name:"sw1",child:{sons:["sw1c1","sw1c2"],dau:["sw1d1","sw1d2"]}
241 },{
242     name:"sw2",child:{sons:["sw2s1"],dau:["sw2d1"]}
243 }]}
244 console.log(ob);
245 alert(ob.wives[0].child.sons[1]);
246 </script>
247
248
249
250
251 -----
252
253
254 cmp - Hyd - prog - hp1,hp2 admin - ha1,ha2
255     bang - prog - bp1,bp2  hr - bhr1,bhr2
256
257
258 Ex:
259
260 <script>
261 var cmp = {hyd:{prog:["hp1","hp2"],admin:["ha1","ha2"]},
262     bang:{prog:["bp1","bp2"],hr:["bhr1","bhr2"]}}
263     alert(cmp.hyd.prog[0]);
264 </script>
265
266
267 -----
268
269 Microsoft Visual Studio Code : It is an open source IDE to work with react, angular,
node,... technologies.
270
271 ---

```

```

272
273 variable : variable is name of memory location to store some values. JavaScript is
loosely typed script so no need to provide data types.
274
275
276
277 Types :
278
279 Local Variable : A variable declaration inside the function we can call as local
variable.
280
281
282 Ex:
283
284 <script>
285     function fun1(){
286         var x=100;
287         alert(x)
288     }
289     function fun2(){
290         alert("From fun2")
291         // alert(x) Local variable
292     }
293     fun1();
294     fun2();
295 </script>
296
297 -----
298
299 Global Variable : Variable declaration outside all functions comes under global
variable. We can access global variable from any function with in the page.
300
301 <script>
302     var x=100;
303     function fun1(){
304         alert(x);
305     }
306     function fun2(){
307         alert(x);
308     }
309     fun1();
310     fun2();
311 </script>
312
313 -----
314
315 Block scope variable : Variable declaration inside the block using let key word. A block
scope variable we can not access from another block. Let introduced with ES- 6.ECMA
script provides standards to JavaScript.
316
317 Ex:
318
319 <script>
320     function fun1(){
321         {
322             var x=100;
323             let y=200;
324         }
325         alert(x);
326         alert(y); // Can not access because of block scope
327     }
328     fun1();
329 </script>
330
331 -----
332
333 Lexical scope : It is scope of outer function of inner function.
334
335 <script>

```

```

336     function funouter(){
337         var x=100;
338         alert("From outer")
339         function funinner(){
340             alert("From inner")
341             alert(x); // calling lexical scope variable
342         }
343         funinner();
344     }
345     funouter();
346 </script>
347
348

```

349 Function : Function is set of executable statements to perform a task. we can use functions for reusable. using function keyword we can create functions.

```

352
353 <script>
354 function fun1(){
355     console.log("Function exec...");
356 }
357 fun1();
358 </script>
359

```

360 Types of functions :

361 1) Anonymous function : If we create a function with out any name comes under anonymous function.

```

362
363 <script>
364 var x=function(){
365     console.log("Func exec..");
366 }
367 //alert(x);
368 x();
369 </script>
370

```

371 2) Nested Function: It is a concept of declaration of a function inside another function.

372 Ex:

```

373
374
375 <script>
376 function fun1(){
377     alert("From fun1");
378     function funinner(){
379         alert("From inner");
380     }
381
382     funinner();
383 }
384 fun1();
385 </script>
386

```

387 -----

388 Ex: 2

```

389 <script>
390 function fun1(){
391     alert("From fun1");
392     return function funinner(){
393         alert("From inner");
394     }
395 }
396
397 var rv=fun1();
398 //alert(rv);
399 rv();
400 </script>
401

```

402 -----

```

403
404 Callback function : If we pass a function as argument of another function comes under
callback.
405 <script>
406 function fun1(x){
407 alert("From fun1");
408 //alert(x);
409 x();
410 }
411 function fun2(){
412 alert("From fun2");
413 }
414 function fun3(){
415 alert("From fun3");
416 }
417 fun1(fun2);
418 fun1(fun3);
419 </script>
420
421 -----
422 <script>
423 function calc(no1,no2,funref){
424     funref(no1,no2);
425 }
426 function add(x,y){
427 alert(x+y);
428 }
429 function mul(x,y){
430 alert(x*y);
431 }
432 calc(10,20,add)
433 calc(10,20,mul);
434 </script>
435 -----
436
437 Arrow Function : It is shortcut of function declaration and also we can use arrow
function as
438 nested function inside the class.
439
440 <script>
441 var arr=[10,20,30,40,50];
442 var narr=arr.map((x,y)=> x*222)
443 alert(narr);
444 </script>
445
446 -----
447 IIFE(Immediate Invoking Function Expression):
448
449 <script>
450 var count=0;
451 var increment=(function(){
452 var count=0;
453     return function(){
454         count=count+1;
455         alert(count);
456     }
457 })()
458 function fun2(){
459     count=222;
460     alert(count);
461 }
462 </script>
463 <body>
464     <input type="button" value="Click" onclick="increment()" />
465     <input type="button" value="Click" onclick="fun2()" />
466 </body>
467
468 -----
469 Event :Event is an action which we are performing on html control.

```

```

470 click , mouseover, mouseout,....
471
472 Every event provides attributes to call the functions
473
474 click - onclick
475 mouseover - onmouseover
476 mouseout - onmouseout
477
478 this : In javascript this is an object refers current control.
479
480 Ex:
481
482 <body>
483   <input type="button" value="Click" onclick="alert('scott');" />
484   
487
488   <input type="button" value="Click Me"
489     onmouseover="this.value='Dont click me';this.type='text'"
490     onmouseout="value='Please click me' " />
491   <input type="button" value="Click" onclick="fun1(this)" />
492 </body>
493 <script>
494 function fun1(t){
495 t.value='Login'
496 t.type='text'
497 }
498 </script>
499
500
501 -----
502 id : Using this property we can provide identity to a control. id we can use to call a
503 control
504 from another control.
505
506 <body>
507   
508   
509 </body>
510
511 Ex:2
512
513 <body>
514 
515 <br />
516 
517 
518 
519 </body>
520
521 -----
522 Hoisting : It is a concept of declaration of variables before starting the execution of
523 script.
524 var contains undefined and let does not have any value at the time of hoisting. If we
525 are trying to access
526 let before initialization it gives reference error and stops the execution of script.
527
528 functions can also hoist.
529
530 <script>
531 fun1();
532 function fun1(){
533 alert("From function");
534 }
535 </script>
536
537 -----

```


536 Note : Arrow functions can not be hoist

537

538 Ex:

539

540 <script>

541 fun1();

542 fun1={()=>{

543 alert("From arrow");

544 };

545 </script>

546

547 -----

548

549 Object : object is collection of properties. Each property is combination of name and value.

550

551 There are different ways to create object

552

553 1) Object Literals : using {} we can create the object.

554

555 <script>

556 var obj={user:"scott",admin:"amith"};

557 console.log(obj.user);

558 console.log(obj.admin);

559 </script>

560

561 2) Using Object.create() function

562

563 <script>

564 var obj=Object.create({

565 username:"john",

566 city:"Mumbai"

567 })

568 //console.log(obj);

569 console.log(obj.username);

570 </script>

571

572 3) Using new keyword

573

574 <script>

575 var obj=new Object({

576 username:"Rajesh",

577 wife:"rw"

578 })

579 console.log(obj.username);

580 </script>

581

582 4) Using class

583 -----

584 Object Destructor : It is a concept of creating the variables with the names of object properties.

585

586 Ex:

587

588 <script>

589 var obj={username:"scott",city:"hyd"}

590 var {city,username}=obj;

591 console.log(username);

592 console.log(city);

593 </script>

594

595 Array Destructor : It is concepts of creating variables with the values of array.

596

597 <script>

598 var arr=[10,20,30,40,50];

599 var [x,y,z,a,b]=arr;

600 alert(x);

601 alert(y);

602 alert(z);

```
603 alert(a);
604 alert(b);
605 </script>
606
607 -----
608
609 BOM(Browser Object Model); Every browser provides some objects those are window,
document, navigator, ...
610
611 window is primary object provides many properties and methods along with other objects
like
612 document, navigator,....
613
614 alert() : using this method we can display message box on browser.
615
616 <script>
617 window.alert("Hi scott");
618 </script>
619
620 prompt() : To display input dialog box.
621
622 <script>
623 var rv=window.prompt("Enter name");
624 alert(rv);
625 </script>
626
627 confirm() : To display confirmation dialog box
628 <script>
629 var rv=confirm("You want to close?")
630 alert(rv);
631 </script>
632
633 print() : To display print properties dialog box.
634
635 <script>
636 window.print();
637 </script>
638 <body>
639 
640 </body>
641
642 Ex:
643
644 <script>
645 function funprint(){
646     var cnf=window.confirm("You want to print ?");
647     if(cnf){
648         window.print();
649     }
650 }
651 </script>
652 <body>
653 <input type="button" value="Print" onclick="funprint()" />
654 </body>
655
656 -----
657
658 location : Using this property we can navigate from one web page to another web page.
659
660 Ex:
661
662 <script>
663 function fun1(){
664     location="http://google.com";
665 }
666 </script>
667 <body>
668 <input type="button" value="Click" onclick="fun1()" />
669 </body>
```

```

670
671 -----
672 open() : Using this method we can open another web page in current tab/ new tab/ new
        window.
673
674 <script>
675 function fun1(){
676     // window.open("http://google.com","_self");
677     window.open("http://fb.com","_blank");
678     window.open("http://google.com","_blank","width=600,height=400")
679 }
680 </script>
681 <body>
682     <input type="button" value="Click" onclick="fun1()" />
683 </body>
684
685 -----
686 setTimeout() : To execute set of statements after specified time. Arguments are a
        function/set of statements and time in milli seconds
687
688 <script>
689     function fun1(){
690         alert("Function called")
691     }
692     window.setTimeout("fun1()",5000);
693 </script>
694
695 -----
696 setInterval() : It is same as setTimeout() but executes a function for every regular
        intervals of time.
697
698 clearInterval() : To stop the functionality of setInterval();
699
700 <script>
701     function fun1(){
702         alert("Function called")
703     }
704     var timer= window.setInterval("fun1()",5000);
705 </script>
706 <body>
707     <input type="button" value="Click" onclick="clearInterval(timer)" />
708 </body>
709
710 -----
711
712 document : using this object we can work with current document. document is an object
        available inside the window object.
713 title : To get/set the title of document.
714
715 <script>
716     function fun1(){
717         document.title="My New Site"
718         alert(window.document.title)
719     }
720 </script>
721 <body>
722     <input type="button" value="Click" onclick="fun1()" />
723 </body>
724
725 -----
726 document.URL : To get the current document url address.
727
728 <script>
729     function fun1(){
730         alert(document.URL)
731     }
732 </script>
733 <body>
734     <input type="button" value="Click" onclick="fun1()" />

```

```

735 </body>
736
737 -----
738
739 document.write() : to write some content on current document.
740
741 <script>
742     function fun1(){
743         document.write("Welcome")
744     }
745 </script>
746 <body>
747     <input type="button" value="Click" onclick="fun1()" />
748 </body>
749
750 -----
751 document.getElementById() : To get an element from the current document based on id of
element.
752
753 <script>
754     function fun1(){
755         var con1=document.getElementById("t1");
756         alert(con1.type);
757         alert(con1.id);
758         alert(con1.value)
759         con1.value="ABCD";
760         var con2=document.getElementById("t2")
761         con2.value="Amith";
762         con2.type="button";
763         con2.onclick=function(){
764             alert("Button clicked")
765         }
766     }
767 </script>
768 <body>
769     <input type="text" id="t1"/>
770     <br />
771     <input type="text" id="t2" />
772     <br />
773     <input type="button" value="Click" onclick="fun1()" />
774 </body>
775 -----
776 03-02
777 -----
778 <script>
779 function funcalc(op){
780     var txt1=document.getElementById("t1").value;
781     var txt2=document.getElementById("t2").value;
782     if(op=="mul")
783         document.getElementById("res").value=txt1*txt2;
784     else if(op=="sub")
785         document.getElementById("res").value=txt1-txt2;
786     else if(op=="div")
787         document.getElementById("res").value=txt1/txt2;
788     else if(op=="add")
789         document.getElementById("res").value=parseInt(txt1)+parseInt(txt2);
790 }
791 </script>
792 <body>
793 <input type="text" id="t1" placeholder="No1"/>
794 <br />
795 <input type="text" id="t2" placeholder="No2"/>
796 <br />
797 <input type="text" id="res" placeholder="Result"/>
798 <br />
799 <input type="button" value=" * " onclick="funcalc('mul')"/>
800 <input type="button" value=" / " onclick="funcalc('div')"/>
801 <input type="button" value=" - " onclick="funcalc('sub')"/>
802 <input type="button" value=" + " onclick="funcalc('add')"/>

```

```

803
804 </body>
805
806 -----
807 navigator : Using this object we can get the information of browser.
808 <script>
809 console.log(navigator.appVersion);
810 </script>
811
812 -----
813 history : To get the history of browser.
814
815 back() : To go to the back page of browser.
816 go() : To go to the specific page of browser.
817 forward() : To go to the forward page of browser.
818
819 <body>
820 <h2>This is p2</h2>
821 <a href="p3.html">Open P3</a>
822 <br />
823 <input type="button" value="Previous" onclick="history.back()" />
824 <input type="button" value="Next" onclick="history.forward()" />
825 <input type='button' value='Reload' onclick="history.go(0)" />
826 </body>
827
828
829 -----
830 Ex:
831
832 <script>
833 function funswap(){
834     var tmp=document.getElementById("txt1").value;
835     document.getElementById("txt1").value=document.getElementById("txt2").value;
836     document.getElementById("txt2").value=tmp;
837 }
838 </script>
839 <body>
840     <input type="text" id="txt1" />
841     <br />
842     <input type="text" id="txt2" />
843     <br />
844     <input type="button" value="Swap" onclick="funswap()" />
845 </body>
846
847 -----
848
849 screen : Using this object we can get the information of current screen.
850
851 availHeight : Returns the height of the screen. It excludes scroll bar
852
853 availWidth : Returns the width of the screen.It excludes task bar
854
855 height : Returns height of screen. Includes task bar
856
857 width: returns the width of screen
858
859 <script>
860 alert(screen.availWidth)
861 alert(screen.width);
862 </script>
863
864 -----
865 innerText : To set/get text on html controls
866 innerHTML : To get/set html content on controls.
867 <script>
868 function fun1(){
869 //document.getElementById("div1").innerText="Rajesh"
870 document.getElementById("div1").innerHTML="<img src='orange.jpg' width='50px' />"
871 //document.getElementById("div1").innerHTML="<img src='orange.jpg' width='50px' />"

```

```

872
873 }
874 </script>
875 <div id="div1">Scott</div>
876 <input type="button" value="Click" onclick="fun1()" />
877
878 -----
879 05-02
880
881 Date : Using this object we can get client system current date and time information.
882 It provides many methods like
883 getHours(), getMinutes(), getSeconds(),getDate(), getMonth(), getYear()...
884
885 Ex:
886
887 <script>
888 var dt=new Date();
889 alert(dt.getDay());
890 alert(dt.getHours())
891 alert(dt.getMinutes())
892 alert(dt.getSeconds())
893 alert(dt.getDate())
894 alert(dt.getMonth())
895 alert(dt.getYear())
896 alert(dt.getFullYear())
897 </script>
898
899 -----
900
901 Ex:
902
903 <script>
904 function fun1()
905 {
906     var dt=new Date()
907     var str=dt.getHours()+ " : "+dt.getMinutes()+" : "+dt.getSeconds();
908     document.getElementById("div1").innerText=str;
909 }
910 setInterval("fun1()",1000);
911 </script>
912 <body onload="fun1()">
913     <div id="div1"></div>
914 </body>
915
916 -----
917
918 Math : using this object we can get mathematical related information.
919
920 Math.random() : using this function we can get the random value between 0 and 1
921
922 <script>
923 alert(Math.random())
924 </script>
925
926 Math.round() : rounds a float value to its nearest integer value
927 <script>
928 alert(Math.round(10.50))
929 </script>
930
931 Math.floor() : Rounds a float value to its nearest lowest integer value.
932 Math.ceil() : Rounds a float value to its nearest highest integer value
933
934 <script>
935 alert(Math.ceil(10.88))
936 </script>
937
938 Math.sin() : to get mathematical sin value
939 <script>
940 alert(Math.sin(0))

```

```

941 </script>
942 -----
943
944 string : string is collection of characters. Every character contains index number.
945
946 <script>
947 var str="welcome";
948 alert(str);
949 alert(str[0]);
950 </script>
951
952 length: using this property we can get total number of characters in a string.
953 <script>
954 var str="welcome";
955 alert(str.length);
956 </script>
957
958 toLowerCase() : To convert the characters of a string into lowercase.
959
960 toUpperCase() : To convert the characters of a string into uppercase.
961
962 <script>
963 var str="Hello";
964 alert(str.toLowerCase());
965 alert(str.toUpperCase());
966 </script>
967
968 -----
969
970 indexOf() : to get the index number of a character from a string
971
972 <script>
973 var str="welcomeoshshs jddjdjojdjdjojkjffjokdkd";
974 alert(str.indexOf("x",5));
975 </script>
976
977 -----
978 charAt() : To get the character of specified index number
979
980 <script>
981 var str="welcomeoshshs";
982 alert(str.charAt(1));
983 </script>
984
985 -----
986 split() : To split a string as array based on input character(s)
987 <script>
988 var str="welcomescott";
989 var arr=str.split("o")
990 alert(arr[1]);
991 </script>
992
993 -----
994 join() : To join all the elements of array as string based on input value
995
996 <script>
997 var arr=[10,20,30,40,50];
998 var str=arr.join("");
999 alert(str);
1000 </script>
1001
1002 -----
1003
1004 06-02
1005 -----
1006 charCodeAt() : to get ASCII value of input character. ASCII is a value of input key
1007
1008 A- 65
1009 B -66

```

```
1010 C - 67
1011 Z-90
1012
1013 a-97
1014 b-98
1015 z-122
1016
1017 space - 32
1018 enter - 13
1019 bksp=8
1020 tab - 9
1021
1022 0-48
1023 1-49
1024 9-57
1025
1026 Ex:
1027 ---
1028 <script>
1029 var str="abcd";
1030 alert(str.charCodeAt(0));
1031 </script>
1032
1033 substr() : To get the substring of input string. total return characters are same as
second argument
1034
1035 substring() : It is same as substr but total return characters are equal to second
argument-first argument.
1036
1037 <script>
1038 var str="welcome scott";
1039 alert(str.substring(2,5))
1040 </script>
1041
1042 -----
1043 replace() : To replace some part of a string with new string. It replaces the first
occurrences.
1044
1045 replaceAll() : Same as replace but replaces all occurrences
1046
1047 <script>
1048 var str="welcome scott john scott";
1049 alert(str.replaceAll("scott","amith"));
1050 </script>
1051
1052 -----
1053 conversion functions : using these functions we can convert the data from one format to
another format.
1054
1055 parseInt() : using this function we can convert input value as integer value
1056
1057 ex:
1058
1059 <script>
1060 var x=100;
1061 var y="200";
1062 alert(x+parseInt(y));
1063 </script>
1064
1065 parseFloat() : Converts input value as float value
1066
1067 <script>
1068 var x="100.20";
1069 alert(parseFloat(x))
1070 </script>
1071
1072 eval() : Using this function we can evaluate a string as expression.
1073
1074 <script>
```



```

1075 var x="100*20+50";
1076 alert(eval(x))
1077 </script>
1078
1079 isNaN() : To check the input value is number or not. It returns true if the input value
            is not a number.
1080
1081 <script>
1082 var x=100;
1083 alert(isNaN(x))
1084 </script>
1085
1086 <script>
1087 var x="10abcd0";
1088 alert(isNaN(x))
1089 </script>
1090
1091 -----
1092
1093 Ajax : It is a web technology to send a request from browser to server without submit
        the web page.
1094
1095 Ajax stands for Asynchronous JavaScript and XML
1096
1097 Asynchronous : It is a process of sending a request from browser to server with
        irrespective of previous request / response
1098
1099 JavaScript : Using javascript we can create ajax object and send that object to server.
1100
1101 XML : Data between browser and server transfers in the form of xml/json.
1102
1103 -----
1104
1105 Ajax we can use to call server API services.
1106
1107 -----
1108
1109 XMLHttpRequest
1110 fetch
1111 axios
1112
1113 using these APIs we can create ajax object and send that object to server
1114 -----
1115
1116 We have different types of methods to send a request from browser to server
1117
1118 get : Send a request with out any data from browser to server.
1119
1120 post : send a request from browser to server with some data.
1121
1122 put : Same as post , we can use to execute update functionalities in server.
1123
1124 delete : Same as post , to work with delete functionalities in server.
1125
1126 -----
1127 then() : then() is a callback executes when the ajax object execution is completed. It
        executes when the promise object execution is completed. we need to pass a function as
        arguments to execute some statements after execution of associated function.
1128
1129 Ex:
1130
1131 <script>
1132 function fun1(){
1133 fetch("https://restcountries.com/v3.1/all").then(function(dt){
1134     return dt.json();
1135 }).then(function(data){
1136     data.map((oneCon)=>{
1137         document.getElementById("tab1").innerHTML+=`<tr>
1138         <td>${oneCon.name.common}

```

```

1139     <td>${oneCon.capital}
1140     <td>${oneCon.population}
1141     <td><img src='${oneCon.flags.png}' width='100px' height='100px' />
1142     `;
1143 })
1144 })
1145 }
1146 </script>
1147 <body>
1148   <input type="button" value="Click" onclick="fun1()" />
1149   <table border="1" id="tab1">
1150   </table>
1151 </body>
1152
1153 -----
1154 promise : promise is an object to call functions asynchronously. By default javascript
executes functions synchronously to execute them asynchronously we can use promise.
1155
1156 then() : It is a callback executes some statements when the promise object execution is
completed.
1157 we need to pass 2 functions as arguments one executes if promise returns success value
another one executes if it returns failure value.
1158
1159 resolve() : using this function we can pass success value from promise object.
1160 reject() : using this function we can pass failure value from promise object.
1161
1162 ex:
1163
1164 <script>
1165 function fun1(){
1166   return new Promise(function(resolve,reject){
1167     reject("promise rejected")
1168   })
1169 }
1170 function fun2(){
1171   console.log("Fun2");
1172 }
1173 fun1().then(function(data){
1174   console.log("First fun exec..")
1175   console.log(data);
1176 },function(err){
1177   console.log("Sec fun exec..")
1178   console.log(err);
1179 });
1180 fun2();
1181 </script>
1182
1183 -----
1184
1185 Ex:2
1186
1187 <script>
1188 function fun1(no){
1189   return new Promise(function(res,rej){
1190     if(isNaN(no)){
1191       rej("Error");
1192     }
1193     else{
1194       no++;
1195       res(no);
1196     }
1197   })
1198 }
1199
1200 function fun2(){
1201 }
1202 fun1('abc').then(function(dt){
1203   alert(dt);
1204   fun1(dt).then(function(dtt){

```

```

1205         alert(dtt);
1206         fun1(dtt).then(function(dttt){
1207             alert(dttt);
1208             },function(err){
1209                 alert(err)
1210             })
1211             },function(er){
1212                 alert(er);
1213             })
1214         },function(e){
1215             alert(e)
1216         })
1217 </script>
1218
1219 -----
1220
1221 sal - 1L - 4L
1222     Tds - no
1223     4L- 8L
1224     Tds - 10%
1225     8L - 12L
1226     Tds - 20%
1227     12L - 16L
1228     Tds - 25%
1229     > 16L
1230     Tds - 30%
1231
1232
1233 Hra - Tds - 5000-7000
1234     Hra - 5000
1235     Tds >7000 <10000
1236     Hra - 8000
1237     Tds > 10000 < 12000
1238     Hra - 10000
1239     >12000
1240     Hra - 15000
1241
1242 Pf : Hra - 7000 && 8000
1243
1244     pf-1000
1245     Hra>8000 < 9000
1246     pf-1500
1247     Hra >9000 <10000
1248     pf-2000
1249     Hra > 10000
1250     pf-30000
1251
1252
1253 -----
1254
1255 Promise.all() : This function executes multiple promise objects and returns success
value if all are executed successfully. If any one fails it returns failure value and
call error callback.
1256
1257 <script>
1258 var p1=new Promise(function(rs,rej){
1259     rs("From p1");
1260 })
1261 var p2=new Promise(function(res,rej){
1262     res("From p2");
1263 })
1264 Promise.all([p1,p2]).then(function(dt){
1265     console.log(dt);
1266 }).catch(e=>console.log(e))
1267 </script>
1268
1269 -----
1270
1271 Promise.race() : Executes the first occurred promise if it is resolved / rejected one.

```

```

1272
1273 <script>
1274   var p1=new Promise(function(res,rej){setTimeout(()=>rej("From p1"),100)})
1275   var p2=new Promise(function(res,rej){res("From p2")})
1276
1277   Promise.race([p1,p2]).then(function(dt){
1278     console.log(dt);
1279   }).catch(e=>console.log(e))
1280 </script>
1281
1282 -----
1283 Promise.any() : Returns the first resolved promise object.
1284
1285 <script>
1286   var p1=new Promise(function(res,rej){res("From p1")})
1287   var p2=new Promise(function(res,rej){res("From p2")})
1288
1289   Promise.any([p1,p2]).then(function(dt){
1290     console.log(dt);
1291   }).catch(e=>console.log(e))
1292 </script>
1293
1294 -----
1295 Promise.allSettled() : To get the information of all api services (resolve/reject)
1296 <script>
1297   var p1=new Promise(function(res,rej){rej("From p1")})
1298   var p2=new Promise(function(res,rej){res("From p2")})
1299
1300   Promise.allSettled([p1,p2]).then(function(dt){
1301     console.log(dt);
1302   })</script>
1303
1304 -----
1305 call() : using call we can call a function of an object by passing another object as own
          object of that function.
1306
1307 <script>
1308 var obj={
1309   uname:"scott",
1310   city:"hyd",
1311   fun1:function(x,y){
1312     alert(x)
1313     alert(y)
1314     var uname="John"
1315     alert("From fun")
1316     alert(uname);
1317     alert(this.uname);
1318     alert(this.city);
1319   }
1320 }
1321 //alert(obj.uname);
1322 var nobj={uname:"Amith",city:"chennai"}
1323 obj.fun1.call(nobj,100,200);
1324 </script>
1325
1326 -----
1327 apply() : It is same as call but we need to pass array as argument
1328
1329 <script>
1330 var obj={
1331   uname:"scott",
1332   city:"hyd",
1333   fun1:function(x,y){
1334     alert(x)
1335     alert(y)
1336     var uname="John"
1337     alert("From fun")
1338     alert(uname);
1339     alert(this.uname);

```

```

1340     alert(this.city);
1341 }
1342 }
1343 //alert(obj.uname);
1344 var nobj={uname:"Amith",city:"chennai"}
1345 obj.fun1.apply(nobj,[100,200]);
1346 </script>
1347
1348 -----
1349
1350 bind() : It is same as call but returns function instead of execute.
1351
1352 <script>
1353 var obj={
1354     uname:"scott",
1355     city:"hyd",
1356     fun1:function(x,y){
1357         alert(x)
1358         alert(y)
1359         var uname="John"
1360         alert("From fun")
1361         alert(uname);
1362         alert(this.uname);
1363         alert(this.city);
1364     }
1365 }
1366 //alert(obj.uname);
1367 var nobj={uname:"Amith",city:"chennai"}
1368 var rv=obj.fun1.bind(nobj,100,200);
1369 //alert(rv);
1370 rv();
1371 </script>
1372
1373 -----
1374
1375 Object Oriented Concepts : JavaScript supports OOPs concepts like inheritance,
1376 constructor,....
1377 OOPs we can use to implement applications in a structured way.
1378
1379 According to oops executable statements we should place inside the class.
1380
1381 class is collection of members. class members are properties and methods. Variable
1382 declaration inside the class we can call as
1383 property. function declaration inside the class we can call as method.
1384
1385 object : Object is instance of class. class members we can access using object. we can
1386 use 'new' keyword to create class object.
1387
1388 constructor() : constructor is a type of method contains class name as method name. By
1389 default every class contains a pre defined constructor to create class object. we can
1390 also create user defined constructor to execute some statements at the
1391 time of creating class object. using constructor keyword we can create user defined
1392 constructor.
1393
1394 <script>
1395     class cls1{
1396         sno=100;
1397         fun1(){
1398             alert("From class method");
1399         }
1400     }
1401     var obj=new cls1();
1402     alert(obj.sno);
1403     obj.fun1();
1404 </script>
1405
1406 -----

```

```

1403
1404 <script>
1405     class cls1{
1406         sno=100;
1407         fun1(){
1408             alert("From class method");
1409         }
1410         constructor(){
1411             console.log("Cons exec....")
1412         }
1413     }
1414     var obj=new cls1();
1415     //alert(obj.sno);
1416     //obj.fun1();
1417 </script>
1418
1419 -----
1420
1421 this : using this keyword we can access the members of class from the methods of same
class.
1422
1423 <script>
1424     class cls1{
1425         uname="scott";
1426         fun1(){
1427             var uname="Alex"
1428             alert(uname);
1429             alert(this.uname);
1430         }
1431     }
1432     var obj=new cls1();
1433     obj.fun1();
1434 </script>
1435
1436 -----
1437
1438 Inheritance : It is a concept of inheriting the members of a class from another class.
using extends keyword we can inherit the members of a class from another class.
1439
1440 We have different types of inheritances
1441
1442 1) Single Inheritance
1443 2) Multiple Inheritance
1444 3) Multi level Inheritance
1445 4) Hybrid Inheritance
1446 5) Hierarchical Inheritance.
1447
1448
1449 Single Inheritance: In this concept we have only 2 classes one is base class another one
is derived class.
1450
1451 Ex:
1452
1453 <script>
1454 class cls1{
1455     fun1(){
1456         alert("From class1");
1457     }
1458 }
1459 class cls2 extends cls1{
1460     fun2(){
1461         alert("From class2");
1462     }
1463 }
1464 var obj=new cls2()
1465 obj.fun1();
1466 </script>
1467
1468

```

```
1469 Multiple Inheritance : In this concept a class can have multiple parent classes.
1470 JavaScript does not support this concept.
1471 Multi Level Inheritance : In this concept a class can behave as both base class and
1472 derived class.
1473 <script>
1474 class cls1{}
1475 class cls2 extends cls1{}
1476 class cls3 extends cls2{}
1477 </script>
1478
1479 Hierarchical Inheritance : In this concept a class can have multiple derived classes.
1480
1481 <script>
1482 class cls1{}
1483 class cls2 extends cls1{}
1484 class cls3 extends cls1{}
1485 </script>
1486
1487 Hybrid Inheritance : It is collection of single, multiple, multi level and hierarchical
1488 inheritances. JavaScript does nit support this concept.
1489 -----
1490 super : using this keyword we can access the members of parent class from the derived
1491 class.
1492 <script>
1493 class cls1{
1494   fun1(){
1495     alert("Fun1 from class1");
1496   }
1497 }
1498
1499 class cls2 extends cls1{
1500   fun1(){
1501     alert("Fun1 from class2");
1502   }
1503   fun2(){
1504     alert("Fun2 from class2");
1505     this.fun1();
1506     super.fun1();
1507   }
1508 }
1509 var obj=new cls2();
1510 obj.fun2();
1511 </script>
1512
1513 -----
1514 Ex:
1515
1516 <script>
1517 class cls1{
1518   sno=1234;
1519   fun1(){
1520     alert("fun1 from class1")
1521     alert(this.sno);
1522     let funinner=()=>{
1523       alert("From inner function");
1524       alert(this.sno);
1525     }
1526     funinner();
1527   }
1528 }
1529
1530 var obj=new cls1();
1531 obj.fun1();
1532 </script>
1533
```

```

1534 -----
1535 Async / Await : It is a concept of working with asynchronous functionalities. Async we
can apply to a function to work with asynchronous functionalities and await we can use
to wait until the current asynchronous functionality is completed.

1536
1537 With out async we can not use await.
1538
1539 Ex:
1540
1541 <script>
1542     async function getData(){
1543         var final_data=await fetch("https://restcountries.com/v3.1/all");
1544         console.log(final_data);
1545     }
1546     function fun2(){
1547
1548     }
1549     getData();
1550     fun2()
1551 </script>
1552
1553 -----
1554
1555 event : event is an object to get the current event information. It provides many
properties related to the current event which if fired.
1556
1557 Events are 2 types
1558
1559 1) Mouse Events : These events fires when we perform any action with mouse
1560     Ex: onclick,onmousedown,onmouseover,....
1561
1562 2) Keyboard Events : These events fires through selected character of keyboard.
1563     Ex: onkeyup,onkeydown...
1564
1565 Properties of mouse events.
1566
1567 clientX : To get the x position of mouse.
1568 clientY : To get the y position of mouse.
1569
1570 These two properties are available in onclick,onmousemove,...
1571
1572 Ex:
1573
1574 <script>
1575     function fun1(e){
1576         var xpos=(e.clientX);
1577         var ypos=(e.clientY);
1578         document.getElementById("div1").innerText=`X pos is ${xpos} and Y pos is
        ${ypos}`;
1579     }
1580 </script>
1581 <body onmousemove="fun1(event)">
1582     <div id="div1">
1583
1584     </div>
1585 </body>
1586 <style>
1587     body{
1588         margin:0px;
1589     }
1590 </style>
1591
1592 -----
1593
1594 button : using this property we can get the information of mouse button clicked by user.
Ig user click on left button if holds value 0 , if center button value is 1, if right
button value is 2. It is available in onmousedown event.
1595

```



```

1596
1597 Ex:
1598
1599 <script>
1600     function fun1(e){
1601         alert(e.button)
1602     }
1603 </script>
1604 <body onmousedown="fun1(event)">
1605
1606 </body>
1607
1608 Ex: 2
1609
1610 <body>
1611     
1612 </body>
1613 <script>
1614     function fun1(e){
1615         if(e.button==2){
1616             alert("Can not copy")
1617         }
1618     }
1619 </script>

```

```

1620 -----

```

1621 type : to get the type of event which is executed.

```

1622
1623
1624
1625 <body>
1626     
1627 </body>
1628 <script>
1629     function fun1(e){
1630         alert(e.type);
1631     }
1632 </script>

```

```

1633 -----

```

1634 key : To get the information of key clicked by user. It is available in keyboard events like onkeydown/onkeyup

```

1635
1636 Ex:
1637
1638
1639 <body>
1640     <input type="text" onkeyup="fun1(event)"/>
1641 </body>
1642 <script>
1643     function fun1(e){
1644         alert(e.key);
1645     }
1646 </script>

```

```

1647 -----

```

1648 onkeydown : This event triggers when we press the character in keyboard. First it calls function later it displays key in the textbox

1649 onkeyup : This event triggers when we release the key. first it prints character in textbox later it calls the function.

```

1650
1651 Ex:
1652
1653
1654 <body>
1655     <input type="text" onkeydown="fun1()" />
1656 </body>
1657 <script>
1658     function fun1(){
1659         alert("Func called")

```

```

1662     }
1663 </script>
1664
1665 -----
1666 keyCode : To get the keycode value of input character. keyCode is same as ASCII but it
1667           is common for both alphabets.
1668
1669 ASCII
1670 -----
1671 A - 65
1672 B - 66
1673 Z - 90
1674
1675 a-97
1676 b-98
1677 z-122
1678
1679 -----
1680 0-48
1681 1-49
1682 9-57
1683
1684 -----
1685 space - 32
1686 enter - 13
1687 bksp - 8
1688 tab - 9
1689
1690 Ex:
1691
1692 Note : keyCode returns purely ASCII values when we use onkeypress event.
1693 -----
1694
1695 Ex:
1696
1697 <script>
1698     function fun1(e){
1699         document.getElementById("div1").innerHTML='';
1700         if(e.keyCode < 48 || e.keyCode>57){
1701             document.getElementById("div1").innerText=("Enter Number")
1702             e.preventDefault();
1703         }
1704     }
1705 </script>
1706 <body>
1707     <input type="text" onkeypress="fun1(event)" />
1708     <div id="div1"></div>
1709 </body>
1710
1711 -----
1712
1713 Event Bubbling/Trickling : It is a concept of execution of parent element events when
1714                             we perform an action on child element.
1715
1716 Bubbling is the concept of calling events from down to up direction (bottom to top).
1717
1718 Trickling is the concept of calling events from up to down(top to bottom);
1719
1720 By default events flow if bubbling.
1721
1722 These two flow execute every time but events execute in the specified flow.
1723
1724 The first executable flow is trickling later bubbling.
1725
1726 if third argument is true that event will call in trickling flow, if false it will call
1727 in bubbling flow
1728 Ex:

```

```

1728
1729 <body>
1730     <div id="div1" >
1731         This is div1
1732         <div id="div2" >
1733             This is div2
1734             <div id="div3">
1735                 This is div3
1736             </div>
1737         </div>
1738     </div>
1739 </body>
1740 <style>
1741     div{
1742         padding:10px;
1743         border:1px solid silver;
1744     }
1745 </style>
1746 <script>
1747     document.getElementById("div1").addEventListener('click',function(){
1748         alert("div1 called")
1749     },true)
1750     document.getElementById("div2").addEventListener('click',function(){
1751         alert("div2 called")
1752     },false)
1753     document.getElementById("div3").addEventListener("click",function(){
1754         alert("div3 called")
1755     },true)
1756 </script>
1757
1758 -----
1759 event.stopPropagation() : using this function we can stop the flow of bubbling and
trickling
1760
1761 Ex:
1762
1763 <body>
1764     <div id="div1" >
1765         This is div1
1766         <div id="div2" >
1767             This is div2
1768             <div id="div3">
1769                 This is div3
1770             </div>
1771         </div>
1772     </div>
1773 </body>
1774 <style>
1775     div{
1776         padding:10px;
1777         border:1px solid silver;
1778         background-color: white;;
1779     }
1780 </style>
1781 <script>
1782     document.getElementById("div1").addEventListener('click',function(e){
1783         alert("div1 called")
1784         this.style.backgroundColor="lightblue"
1785         //e.stopPropagation();
1786     },false)
1787     document.getElementById("div2").addEventListener('click',function(e){
1788         alert("div2 called")
1789         this.style.backgroundColor="lightgreen"
1790         e.stopPropagation();
1791     },true)
1792     document.getElementById("div3").addEventListener("click",function(e){
1793         alert("div3 called")
1794         this.style.backgroundColor="yellow"
1795         //e.stopPropagation();

```

```

1796     },false)
1797 </script>
1798
1799 -----
1800 rest : using rest operator we can get rest of the array elements into variables.
1801 <script>
1802     var arr=[10,20,30,40,50];
1803     var [x,y,...z]=arr;
1804     alert(x);
1805     alert(y);
1806     alert(z[0]);
1807 </script>
1808
1809 spread : To spread the values of array into multiple variables
1810
1811 <script>
1812     var arr=[10,20,30,40,50];
1813     function fun1(x,y,z,a,b){
1814         alert(x);
1815         alert(y);
1816         alert(z);
1817         alert(a);
1818         alert(b);
1819     }
1820     fun1(...arr);
1821 </script>
1822
1823 -----
1824 default : To set default value to the argument of function.
1825
1826 <script>
1827     function fun1(x,y=333){
1828         alert(x+y);
1829     }
1830     fun1(100,200)
1831     fun1(1)
1832 </script>
1833
1834 -----
1835 protocols: protocol is set of rules to transfer the data from one location to another
location.
1836
1837 protocols are 2 types
1838 1) state full protocols
1839 2) state less protocols
1840
1841 state full protocols : These protocols can maintain the state of application. Means they
can transfer the data of one page to another page. These protocols we can use in windows
applications.
1842
1843 ex: tcp/ftp,...
1844
1845 state less protocols : These protocols can not maintain the state of application. Means
these protocols can not transfer the data of one page into another page. We can use
these protocols to work with we applications. Because these protocols dont carry the
data of one page to another page so that the transmission speed is more.
1846
1847 ex: tcp/ip, http, https,...
1848
1849
1850 state management : using this concept we can maintain the state of application. This
concept provides objects like
1851 cookies
1852 sessions
1853 local storage
1854 session storage
1855 ....
1856
1857 JavaScript supports localStorage object with many methods to maintain the state of

```

```

1858 application.
1859 getItem() : using this method we can read the data of local storage
1860
1861 setItem() : To create a property in local storage
1862
1863 removeItem() : To remove a property of local storage
1864
1865 clear() : To clear all the properties of local storage
1866
1867 Ex:
1868 ---
1869 p1.html
1870
1871 <script>
1872     localStorage.setItem("x","100");
1873     localStorage.setItem("y","200")
1874     alert(localStorage.getItem("x"));
1875
1876 </script>
1877 <body>
1878     <h2>This is p1</h2>
1879     <a href="p2.html">Open P2</a>
1880 </body>
1881
1882 p2.html
1883 -----
1884 <script>
1885     alert("from p2");
1886     alert(localStorage.getItem("x"));
1887     //localStorage.removeItem("y")
1888     localStorage.clear();
1889 </script>
1890 <body>
1891     <h2>This is p2</h2>
1892 </body>
1893
1894 -----
1895 session storage : It is same as local storage but the data of session storage will
1896 destroy once we closes the browser.
1897
1898 Note : Local storage data is available in browser even we close, but not session storage
1899         local storage we can access from another tab but not session storage
1900
1901 -----
1902 closure : closure is an object to maintain the data of outer function which we can
1903 access from inner function.
1904
1905 The variable of outer function will destroy when the execution of that function is
1906 completed.
1907 but if we access the outer function variables from inner function(lexical scope) then
1908 the data should prevent in an object, that object is closure.
1909
1910 console.dir() is a function to display the closure data
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921

```

1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990

1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059

2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128

2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197

2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266

2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335

2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404

2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473

2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542

2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611

2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680

2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749

2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818

2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887

2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956

2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025

3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063