

Curso de java script desde cero

Friday, June 21, 2024 1:42 PM

- TENER INSTALADO VISUAL STUDIO CODE Y NODE JS

Instala Node.js

Publicado el 24 de noviembre de 2023

Clase

Node.js es un entorno de ejecución back-end diseñado para ejecutar código JavaScript. Para comenzar, necesitamos instalar Node.js y tener una terminal.

Dirígete a <https://nodejs.org> y descarga la última versión disponible.

El número de versión puede variar según el momento en que realices este curso, pero ten en cuenta que las prácticas que realizaremos no dependerán de una versión específica de Node.

En este curso, no escribiremos sitios web Node.js directamente. Sin embargo, es esencial tener Node.js instalado, en caso de que uses herramientas, paquetes, formateadores, etc., que utilicen Node.js internamente.

Para verificar si tenemos Node.js instalado, abrimos una ventana de terminal.

¿Qué terminal usar?

¿Qué terminal usar?

- Puedes usar la aplicación de terminal integrada en tu sistema (como Terminal en Mac o Símbolo del sistema o Cmdr en Windows).
- O también usar terminales como Hyper, iTerm o la terminal de VSCode.

Todas estas terminales son equivalentes.

Ahora sí, ¿Cómo verificar si tenemos Node.js instalado usando nuestra terminal?

Podemos probar escribiendo 'node -v' en la terminal y presionamos Enter. Esto nos mostrará la versión de Node.js que tenemos instalada.

Una vez tengamos Node.js en nuestra terminal, podemos escribir nuestro código JavaScript. Para ello, escribimos en la terminal 'node' y comenzamos a escribir nuestro Hola Mundo.

- ¡Hola Mundo!

Formas de ejecutar JavaScript:

- Consola del navegador
- Etiqueta <script> de HTML
- Archivos JS externos
- Node.js

The screenshot shows a browser window with the Google homepage. Below it, the developer tools console tab is active, showing the following interaction:

```
> console.log('Hola Mundo!')
Hola Mundo!
< undefined
> 1
```

At the bottom, the Sources tab is open, displaying the code of a file named "hola_mundo.html". The code includes a script tag with a console.log statement:

```
4. clase > hola_mundo.html > body > script
1   <!DOCTYPE html>
2   <html lang="en">
3     <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Document</title>
7   </head>
8   <body>
9     <script> log(...data: any[]): void
10    |   console.log('Hola mundo')
11    |>
```

```
11 |   </script>
12 | </body>
13 | </html>
```

... clase-hola-mundo.html x JS clase-hola-mundo.js

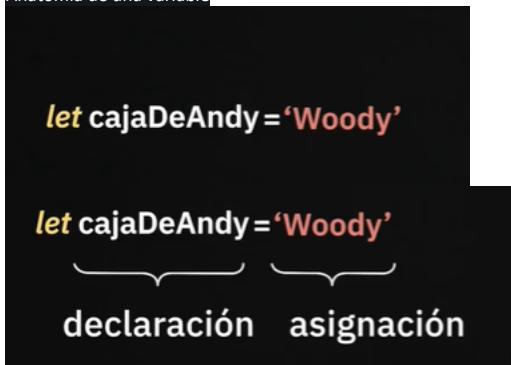
clase-hola-mundo.html > html > body > script

```
5  <meta http-equiv="X-UA-Compatible" content="IE=edge">
6  <meta name="viewport" content="width=device-width, initial-
7  <title>Document</title>
8  </head>
9  <body>
10 <script src="clase-hola-mundo.js">
11 </script>
12 </body>
13 </html>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

• Platzi node clase-hola-mundo.js
Hola Mundo
○ Platzi

- Anatomía de una variable



Welcome JS clase_variables.js

```
5.Clase > JS clase_variables.js > [o] idDelUsuario
1 let cajaDeAndy = 'Woody'
2 console.log(cajaDeAndy)
3
4 // Lo NO permitido
5 let c = 'Woody'
6 let cda = 'Woody'
7 let pcAndy = 'Woody'
8
9 // Lo permitido
10 let primerTrasteoDeAndy = 'Woody'
11 let urlDelUsuario = 'https://www.google.com'
12 let idDelUsuario = '123456789'
```

let

```
● ● ●
```

let contador
contador = contador + 1

const

```
● ● ●
```

const pi = 3.14

- Tipos de datos: Mutabilidad e inmutabilidad

10 tipos de datos

Primitivos	Complejos
string	object
number	array
boolean	function
null	
undefined	
symbol	
bignit	

```
VS Code Editor (Split)
JS Welcome JS clase_mutabilidad_inmutabilidad.js U ●
6.clase > JS clase_mutabilidad_inmutabilidad.js > ...
1 //tipo de dato primitivo imutables
2 let numero=23
3 numero=numero+10
4 console.log(numero)
5
6 let esVerdadero=true
7 esVerdadero=false
8 console.log(esVerdadero)
9
10 //tipo de dato complejo mutable
11 let usuario={nombre:'Pepito',edad:15}
12 usuario.edad=20
13 console.log(usuario)
14
15 let frutas=['manzana','pera']
16 frutas[0]='sandia'
17 console.log(frutas)
18
19 function cambiarNombre(objeto){objeto.nombre='Nuevo nombre'}
20 let persona={nombre:'Antonio'}
21 cambiarNombre(persona)
22 console.log(persona)
```

- **Paso por valor**

```
JS clase-paso-por-valor-y-paso-por-referencia.js > [●] manzana
14 b = 'Platzi'
15 c = undefined
16
17 // Paso por referencia
18
19 let manzana = ['manzana']
```

Variables	Valores	Dirección en memoria	Objeto
manzana	<#001>	#001	['manzana']

```
JS clase-paso-por-valor-y-paso-por-referencia.js > ...
17 // Paso por referencia
18
19 let frutas = ['manzana']
20 frutas.push('pera')
21 console.log(frutas)
```

Variables	Valores	Dirección en memoria	Objeto
frutas	<#001>	#001	['manzana', 'pera']

```
... JS clase-paso-por-valor-y-paso-por-referencia.js ●
JS clase-paso-por-valor-y-paso-por-referencia.js > [●] copiaDePanes
alo... 18
19 let frutas = ['manzana']
20 frutas.push('pera')
21 console.log(frutas)
22
23 let panes = ['🍌']
24 let copiaDePanes = panes
```

Variables	Valores	Dirección en memoria	Objeto
panes	<#001>	#001	['🍌']
copiaDePanes	<#001>		

JS clase-paso-por-valor-y-paso-por-referencia.js

```

1 let c = z
2
3 console.log(x, y, z, a, b, c)
4
5 a = 12
6 b = 'Platzi'
7 c = undefined

```

Variables	Valores
x	1
y	Hola
z	null
a	12
b	Platzi
c	undefined

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL zsh + v

```

1 Platzi node clase-paso-por-valor-y-paso-por-referencia.js
2 Hola null 1 Hola null
3 Platzi []

```

- Paso por referencia

JS clase-paso-por-referencia.js

```

1 // Tipo de dato complejo - Paso por referencia
2
3 let frutas = {
4   naranja: '🍊'
5 }
6 let vegetales = frutas
7 vegetales.naranja = '🥕'
8 console.log(frutas)

```

Variables	Valores	Dirección en memoria	Objeto
frutas	<#001>	#001	{ naranja: '🍊' }
vegetales	<#001>		

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

1 Platzi node clase-paso-por-referencia.js
2 { naranja: '🥕' }
3 Platzi []

```

- Creación de strings

- Creación de strings
- Concatenación
- String: literals y values
- Caracteres de escape
- Strings largos
- Métodos

JS Creación_de_strings.js

```

1 //creacion de strings
2 const primeraOpcion='Comillas simples'
3 const segundaOpcion="Comillas simples"
4 const terceraOpcion='Comillas simples'
5
6 //concatenacion: opcion +
7 const direccion='calle falsa 123'
8 const ciudad='Springfield'
9 const direccionCompleta='Mi dirección completa es '+direccion+ciudad
10 console.log(direccionCompleta)
11
12 const direccionCompletaConEspacio='Mi dirección completa es '+direccion+' '+ciudad
13 console.log(direccionCompletaConEspacio)
14
15 // 2. concatenacion template literals
16 const nombre='Estefany'
17 const pais='AVI-ANV'
18 const presentacion='Hola, soy ${nombre} de ${pais}'
19 console.log(presentacion)
20
21 // 3. concatenacion con join()
22 const primeraParte='Me encanta'
23 const segundaParte='la gente de'
24 const terceraParte='AVI-ANV'
25 const pensamiento=[primeraParte,segundaParte,terceraParte]

```

JS Creación_de_strings.js

```

25 const pensamiento=[primeraParte,segundaParte,terceraParte]
26 console.log(pensamiento.join(' _ '))

```

```

27 //concatenacion con concat
28 const hobbie1='correr'
29 const hobbie2='leer'
30 const hobbie3='estudiar'
31 const hobbies='Mis hobbies son: '+concat(hobbie1, ', ',hobbie2, ', ',hobbie3, '.')
32 console.log(hobbies)
33
34 //caracteres de escape
35 //escape Alternativo
36 const escapeAlternativo="I'm Software Engineer"
37
38 //escape BarraInvertida
39 const escapeBarraInvertida='I\'m Software Engineer'
40
41 //escape ComillaInvertida
42 const escapeComillaInvertida='I'm Software Engineer'
43
44

```

```

Welcome JS Creación_de_strings.js U ●
9.Clase y 10 > JS Creación_de_strings.js > ...
  ~0  const escapeComillaInvertida='I\'m Software Engineer'
41
42 //escape ComillaInvertida
43 const escapeComillaInvertida='I'm Software Engineer'
44
45 //escritura string largos
46 const poema='Las rosas son rojas,\n'+
47 'Las violetas son azules,\n'+
48 'Caracter inesperado,\n'+
49 'En la lnea 86.'
50 console.log(poema)
51 const poema2='Las rosas son rojas,\n\
52 Las violetas son azules,\n\
53 Caracter inesperado,\n\
54 En la lnea 86.'
55 console.log(poema2)
56 const poema3='Las rosas son rojas,
57 Las violetas son azules,
58 Caracter inesperado,
59 En la lnea 86.'
60 console.log(poema3)

```

- Manipulación de strings



```

'midudev'.length // 7
'midudev'[1] // i
'midudev'.includes('dev') // true
'midudev'.indexOf('midu') // 0
'midudev'.startsWith('midu') // true
'midudev'.endsWith('paint') // false
'midudev'.slice(0, 4) // 'midu'
'midudev'.slice(4) // 'dev'
'midudev'.toUpperCase() // 'MIDUDEV'
'MiduDev'.toLowerCase() // 'midudev'
'midudev'.replace('dev', '') // 'midu'
'midu'.repeat(3) // 'midumidumidu'
' mi du '.trim() // 'mi du'
'mi du dev'.split(' ') // [ 'mi', 'du', 'dev' ]

```

```

Welcome JS manipulacionstringsjs U X
11.Clase > JS manipulacionstringsjs > ...
1 // String primitivos
2
3 const stringPrimitivo = 'Soy un string primitivo'
4 console.log(typeof stringPrimitivo)
5
6 const stringPrimitivoTambien = String('Soy un string primitivo tambien')
7 console.log(typeof stringPrimitivoTambien)
8
9 // String objeto
10 const stringObjeto = new String('Soy un string objeto')
11 console.log(typeof stringObjeto)
12
13 // Acceder a caracteres
14
15 const saludo = 'Hola. ¿C mo est s?'
16
17 console.log(saludo[2])
18 console.log(saludo.charAt(2))
19 console.log(saludo.slice(0, 2))
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122

```

```
17  console.log(saludo.indexOf('o'))
18  console.log(saludo.indexOf('Cá'mo'))
19  console.log(saludo.indexOf('como'))
20  console.log(saludo.lastIndexOf('o'))
21  console.log(saludo.slice(3, 5))
22  console.log(saludo.length)
23  console.log(saludo.toLocaleUpperCase())
24
25
26
27
28  const saludoDividido = saludo.split(' ')
29  console.log(saludoDividido)
30  console.log(saludoDividido[1])
31
32  const saludoConEspacios = ' Hola Mundo '
33  const saludoSinEspacios = saludoConEspacios.trim()
34  console.log(saludoSinEspacios)
35
36  const saludoOriginal = 'Hola Mundo'
37  const nuevoSaludo = saludoOriginal.replace('Mundo', 'δV@p')
38  console.log([nuevoSaludo])
```

- **Tipo de dato primitivo: number**

```
12.Clase > JS manipulacionstrings.js U • JS tipodatoprimitivonumberjs U •
12.Clase > JS tipodatoprimitivonumberjs > ...
1  const entero=42
2  const decimal=3.14
3  const científico=5e3
4  const infinito=Infinity
5  const noEsUnNúmero=NaN
6  const suma=3+4
7  const resta=4-4
8  const multiplicación=4*7
9  const division=16/2
10 const modulo=15%8
11 const exponente=2**3
12 const resultado=0.1+0.2
13 const resultadotoFixed=resultado.toFixed(1)
14 console.log(resultado)
15 console.log(resultadotofixed(1))
16 console.log(resultado==0.3)
17 const raizCuadrada=Math.sqrt(16)
18 const valorAbsoluto=Math.abs(-7)
19 const aleatorio=Math.random()
20 console.log(raizCuadrada)
21 console.log(valorAbsoluto)
22 console.log(aleatorio)
```

- **Conversión a Booleanos**

```
13.Clase > JS conversionabooléanos.js U •
13.Clase > JS conversionabooléanosjs > ...
1  const isActive = true
2  const hasPermission = false
3
4  // Conversión implícita
5  const result = 5 > 3
6  console.log(result)
7
8  const name = 'Platzi'
9  console.log (!!name)
10
11 // Conversión explícita
12 const value = 0
13 const otherValue = -24
14 const explicitBoolean = Boolean(otherValue)
15 console.log([explicitBoolean])
```

- **Tipos de datos primitivos: null, undefined, symbol y bigint**

```
14.Clase > JS null_indefinet_symbol_bigint.js U • JS casting_coersion.js U
14.Clase > JS null_indefinet_symbol_bigintjs > ...
1  const snoopy=null
2  console.log(snoopy)
3  console.log(typeof snoopy)
4  let name
5  console.log(name)
6  const uniqueId=Symbol('id')
7  const symbol1=Symbol(1)
8  const symbol2=Symbol(1)
9  console.log(symbol1==symbol2)
10 const ID=Symbol('id')
11 let user={}
12 user[ID]='1234'
13 console.log(user)
14 const bigNumber=109823746783982764567847654782374n
```

```
15 console.log(bigNumber)
16 const numberOfParticlesInTheUniverse=1000000000000000000000000
17 console.log([numberOfParticlesInTheUniverse])
```

- Conversión de tipos: Type Casting y Coercion

The diagram consists of three overlapping dark grey rectangles. The top rectangle contains the title 'Lenguajes de programación' and two buttons: 'Compilados' and 'Interpretados'. Below it, the middle rectangle also has the same title and buttons. The bottom rectangle is partially visible and also contains the same title and buttons. In the center of the middle rectangle, there are two columns of language names: 'C, C++, Rust, Go, Swift' on the left and 'JavaScript, Python, Ruby, PHP' on the right. At the bottom of the middle rectangle, there are two more columns: 'Chequeo estático de tipos' on the left and 'Chequeo dinámico de tipos' on the right.

Lenguajes de programación

Compilados Interpretados

C, C++, Rust, Go, Swift JavaScript, Python, Ruby, PHP

Chequeo estático de tipos Chequeo dinámico de tipos

Conversion de tipos

const numero = 2
const booleano = true
console.log(numero + booleano)
// 3

Conversión

Implícita Explícita

2 + true String()
Number()
Boolean()

JS Conversióndetipos_explicita_e_ímplicitaj.js

```
1 //! Explicit Type Casting
2
3 const string = '42'
```

```

4 const integer = parseInt(string)
5 console.log(integer)
6 console.log(typeof integer)
7
8 const stringDecimal = '3.14'
9 const float = parseFloat(stringDecimal)
10 console.log(float)
11 console.log(typeof float)
12
13 const binary = '1010'
14 const decimal = parseInt(binary, 2)
15 console.log(decimal)
16 console.log(typeof decimal)
17
18 //! Implicit Type Casting
19
20 const sum = '5' + 3
21 console.log(sum, typeof sum)
22
23 const sumWithBoolean = '3' + true
24 console.log(sumWithBoolean)
25

```

16.Clase > JS Conversión de tipos explícita e implícita.js U ●

```

17
18 //! Implicit Type Casting
19
20 const sum = '5' + 3
21 console.log(sum, typeof sum)
22
23 const sumWithBoolean = '3' + true
24 console.log(sumWithBoolean)
25
26 const sumWithNumber = 2 + true
27 console.log(sumWithNumber)
28
29 //! Ejercicio: ¿Qué hace JavaScript?
30 const stringValue = '10'
31 const numberValue = 10
32 const booleanValue = true
33
34 console.log('-----');
35
36 // Combinaciones con stringValue
37 console.log(stringValue + stringValue); // CONCATENAR
38 console.log(stringValue + numberValue); // CONCATENAR
39 console.log(stringValue + booleanValue); // CONCATENAR
40
41 // Combinaciones con numberValue

```

16.Clase > JS Conversión de tipos explícita e implícita.js U ●

```

30 const stringValue = '10'
31 const numberValue = 10
32 const booleanValue = true
33
34 console.log('-----');
35
36 // Combinaciones con stringValue
37 console.log(stringValue + stringValue); // CONCATENAR
38 console.log(stringValue + numberValue); // CONCATENAR
39 console.log(stringValue + booleanValue); // CONCATENAR
40
41 // Combinaciones con numberValue
42 console.log(numberValue + stringValue); // CONCATENAR
43 console.log(numberValue + numberValue); // SUMAR
44 console.log(numberValue + booleanValue); // SUMAR
45
46 // Combinaciones con booleanValue
47 console.log(booleanValue + stringValue); // CONCATENAR
48 console.log(booleanValue + numberValue); // SUMAR
49 console.log(booleanValue + booleanValue); // SUMAR

```

- Proyecto: Crea un perfil para redes sociales con JavaScript

.. Welcome JS perfilredesociales.js U ●

17.Clase > JS perfilredesociales.js > ...

```

1 // Social Media Profile
2
3 // 1. User information
4 const username = 'codingAdventurer'
5 const fullName = 'Jhon Doe'
6 const age = 25
7 const isStudent = true
8
9 // 2. Address
10 const address = {

```

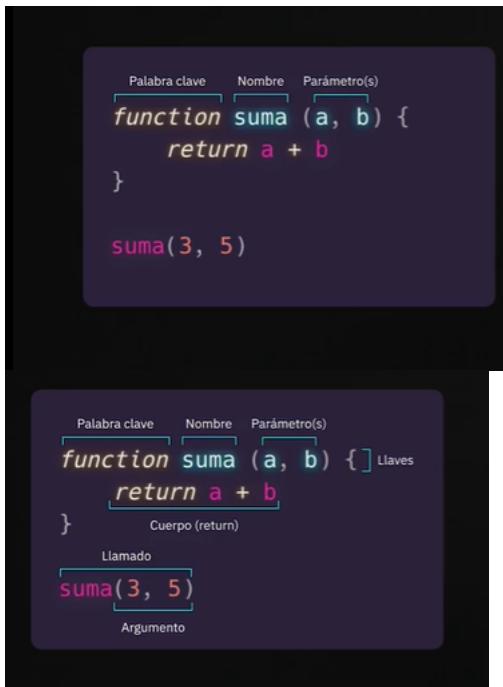
```

11 street: '123 Main Street',
12 city: 'Techville',
13 state: 'Codingland',
14 zipCode: 54321
15 }
16
17 // 3. Hobbies
18 const hobbies = ['Coding', 'Reading', 'Gaming']
19
20 // 4. Generating personalized bio
21 const personalizedBio = `Hi, I'm ${fullName}.
22 I'm ${age} years old.
23 I live in ${address.city}.
24 I love ${hobbies.join(', ')}.
25 Follow me for coding adventures!`
```

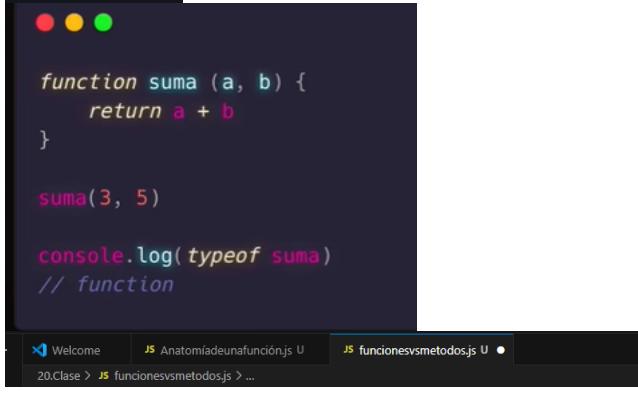
// 5. Print the user profile and bio

28 console.log(personalizedBio)

- Anatomía de una función



- Funciones vs Métodos



```

1 // Capacidades que tienen las funciones al igual que otros objetos
2
3 // 1. Pasar funciones como argumentos -> callback
4
5 function a () {}
6 function b (a) {
7   b(a)
8
9   // Retornar funciones
10
11  function a () {
12    function b () {}
13    return b
14  }
15
16 // Asignar funciones a variables -> Expresión de función
17
18 const a = function () {}
19
20 // Tener propiedades y métodos
21
22 function a () {}
23 const obj = {}
24 a.call(obj)
25
26 // Anidar funciones -> Nested functions
27
28 function a () {
29   function b () {
30     function c () {
31
32     }
33   }
34   c()
35   b()
36 }
37 a()
38
39 // ¿Es posible almacenar funciones en objetos?
40
41 const rocket = {
42   name: 'Falcon 9',
43   launchMessage: function launchMessage () {
44     console.log('ΔY')
45   }
46 }
47
48 rocket.launchMessage()

```

• Welcome JS Anatomíadeunafunción.js U •

20.Clase > JS funcionesvsmetodos.js ...

```

22 function a () {}
23 const obj = {}
24 a.call(obj)
25
26 // Anidar funciones -> Nested functions
27
28 function a () {
29   function b () {
30     function c () {
31
32     }
33   }
34   c()
35   b()
36 }
37 a()
38
39 // ¿Es posible almacenar funciones en objetos?
40
41 const rocket = {
42   name: 'Falcon 9',
43   launchMessage: function launchMessage () {
44     console.log('ΔY')
45   }
46 }
47
48 rocket.launchMessage()

```

• Funciones puras e impuras JS funcionesPurasEImpuras.js U •

21.Clase > JS funcionesPurasEImpuras.js ...

```

1 function sum(a,b){return a+b}
2 function sum(a,b){console.log('A: ',a)
3   return a+b}
4 let total=0
5 function sumWithSideEffect(a){total+=a
6   return total}
7 function square(x){return x*x}
8 function addTen(y){return y+10}
9 const number=5
10 const finalResult=addTen(square(number))
11 console.log(finalResult)

```

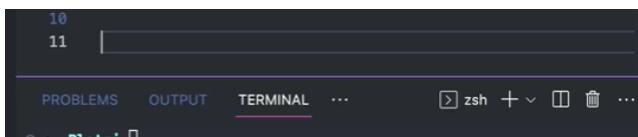
JS clase-funciones-puras-e-impuras.js •

JS clase-funciones-puras-e-impuras.js

```

2
3 // Side Effects
4 // 1. Modificar variables globales
5 // 2. Modificar parámetros
6 // 3. Solicitudes HTTP
7 // 4. Imprimir mensajes en pantalla o consola
8 // 5. Manipulación del DOM
9 // 6. Obtener la hora actual

```



.indentificador this



JS clase-identificador-this.js ×

```
JS clase-identificador-this.js > ...
1 const house = {
2   dogName: 'Fido',
3   dogGreeting: function () {
4     console.log(`Hi, I'm ${this.dogName}`)
5   }
6 }
7
8 house.dogGreeting()
```

PROBLEMS OUTPUT TERMINAL ...

```
zsh + ~
```

- → Platzi node clase-identificador-this.js
Hi, I'm Fido
- → Platzi

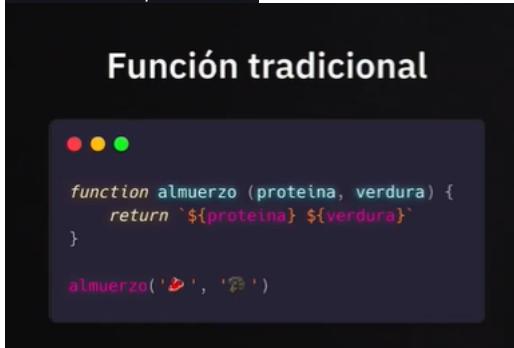
Métodos bind, call y apply

Go Run ... ← → JS Identificador this.js U JS Métodos bind, call y apply.js U ●

23.Clase > JS Métodos bind, call y apply.js > ...

```
1 const owner = 'Lucy'
2 const address = '123 Avenue'
3
4 function dogGreeting (owner, address) {
5   console.log(`Hi, I'm ${this.dogName} and I live with ${owner} on ${address}`)
6 }
7
8 const newHouse = {
9   dogName: 'Coconut'
10 }
11
12 dogGreeting.call(newHouse, owner, address)
13
14 const necessaryValues = [owner, address]
15 dogGreeting.apply(newHouse, necessaryValues)
16
17 const bindingWithBind = dogGreeting.bind(newHouse, owner, address)
18 bindingWithBind()
```

- Funciones flecha y enlace léxico



Arrow Function

```
const almuerzo = (proteina, verdura) => {
  return `${proteina} ${verdura}`
}

almuerzo('pollo', 'papa')
```

```
2Case 1: Funciones flecha y enlace lexicojs U
1 const greetingFunction(name){return `Hi, ${name}`}
2 const newGreeting(name){return `Hi, ${name}`}
3 const newGreetingImplicit(name=>`Hi, ${name}`)
4 const newGreetingImplicitWithTwoParameters(name,lastName)=>`Hi, I'm ${name} ${lastName}`
5 const functionalCharacter={name:'Uncle Ben',messageWithTraditionalFunction:function(message){console.log(`${this.name} says ${message}`)}}
6 functionalCharacter.messageWithTraditionalFunction('With great power comes great responsibility.')
7 functionalCharacter.messageWithArrowFunction(`Darew of Doctor Octopus.`)
```

- Implicaciones de duplicar código

Notación literal de objeto

```
const cohete = {
  nombre: 'Falcon 9',
  mensajeDeDespegue: function mensajeDeDespegue () {
    console.log('🔥')
  }
}
```

Duplicar código implica:

- Mantenimiento difícil
- Aumento de la complejidad
- Mayor probabilidad de errores
- Dificultad de escalar
- Tiempo y recursos
- Violación de principios de diseño (DRY)
- Dificultad en la identificación de errores

- Funciones constructoras

```
JS clase-funciones-constructoras.js •
JS clase-funciones-constructoras.js > [o] falconHeavyRocket
on...
1 function Rocket (name) {
2   this.name = name
3 }
4
5 const falcon9Rocket = new Rocket('Falcon 9')
6 const falconHeavyRocket = new Rocket('Falcon Heavy')
```

```
PROBLEMS OUTPUT TERMINAL ... zsh + ⌂ ...
JS clase-funciones-constructoras.js X
JS clase-funciones-constructoras.js > Rocket > launchMessage
...
1 function Rocket (name, ownMessage) {
2   this.name = name
3   this.launchMessage = () => ownMessage
4 }
5
6 const falcon9Rocket = new Rocket('Falcon 9', 'Goodbye ever
7 const falconHeavyRocket = new Rocket('Falcon Heavy', 'See
8 console.log(falcon9Rocket.name)
9 console.log(falcon9Rocket.launchMessage())
```

Clase

En JavaScript, los tipos de funciones pueden clasificarse de varias maneras según su comportamiento y uso. Aquí tienes una lista de algunos tipos comunes de funciones en JavaScript:

// 1. Funciones Declarativas (o con nombre):

- Se definen con la palabra clave function
- Pueden ser referenciadas antes de su declaración.

```
function suma(a, b) { return a + b; }
```

// 2. Funciones Expresivas (o anónimas):

- Se asignan a variables.
- A menudo se utilizan para asignar funciones como valores a variables.

```
function suma(a, b) { return a + b; }
```

// 2. Funciones Expresivas (o anónimas):

- Se asignan a variables.
- A menudo se utilizan para asignar funciones como valores a variables.

```
const suma = function(a, b) { return a + b; };
```

// 3. Funciones Flecha:

- Introducidas en ES6, proporcionan una sintaxis más concisa.
- Tienen un comportamiento ligeramente diferente con respecto al valor de this.

```
const suma = (a, b) => a + b;
```

// 4. Funciones Constructoras:

- Utilizadas para crear objetos con new.
- Utilizan this para asignar propiedades al nuevo objeto.

```
function Persona(nombre, edad) { this.nombre = nombre; this.edad = edad; }
const persona1 = new Persona('Juan', 25);
```

// 5. Funciones de Orden Superior (Higher-Order Functions):

- Aceptan funciones como argumentos o devuelven funciones.
- Ejemplos incluyen map, filter, reduce.

// 5. Funciones de Orden Superior (Higher-Order Functions):

- Aceptan funciones como argumentos o devuelven funciones.
- Ejemplos incluyen map, filter, reduce.

// 6. Funciones Recursivas:

- Llamadas a sí mismas durante la ejecución.
- Útiles para problemas que se pueden dividir en subproblemas más pequeños.

```
function factorial(n) { if (n === 0 || n === 1) { return 1; } else { return
n * factorial(n - 1); } }
```

// 7. Funciones Anidadas (Nested Functions):

- Definidas dentro de otra función.
- Pueden acceder a las variables de la función contenedora (closure).

```
function exterior() { let variableExterior = 'Exterior';
interior() { console.log(variableExterior); } interior(); } exterior();
```

// 8. Métodos de Objeto:

- Funciones que son propiedades de objetos y se llaman métodos cuando se invocan en el contexto de ese objeto.

```
const objeto = { metodo: function() { console.log('Hola desde el método');
} }; objeto.metodo();
```

// 9. Funciones Asincrónicas:

- Utilizadas para manejar operaciones asíncronas con callbacks, Promesas o Async/Await.

```
async function fetchData() { const response = await
fetch('https://api.example.com/data'); const data = await
response.json(); console.log(data); }
```

// 10. Funciones Puras:

- Dado el mismo conjunto de entradas, siempre producirán el mismo resultado sin causar efectos secundarios observables.
- No dependen de ni modifican estados externos.

```
function suma(a, b) { return a + b; }
```

Estas son algunas de las categorías comunes de funciones en JavaScript. Es importante comprender estas diferentes formas de definir y utilizar funciones para escribir código más claro y eficiente.

• Objeto window y modo estricto

```
function favoriteCharacter(){
    console.log(`I'm ${this.name}`)
}

const character={
    name:'Batman',
    age:'T5'
}

favoriteCharacter()
```

```
I'm
< undefined
> function favoriteCha f(...data) {
    console.log(this)
}

favoriteCharacter()
```

75 Issues: 1 74

```
favoriteCharacter()

I'm
< undefined
> function favoriteCha f(...data) {
    console.log(this)
}

favoriteCharacter()
```

75 Issues: 1 74

- caches: CacheStorage {}
- cancelAnimationFrame: f cancelAnimationFrame()
- cancelIdleCallback: f cancelIdleCallback()
- captureEvents: f captureEvents()
- chrome: {loadTimes: f, csi: f}
- clearInterval: f clearInterval()
- clearTimeout: f clearTimeout()
- clientInformation: Navigator {vendorSub: '', productSub: '2'}
- close: f close()
- closed: false
- closure_lm_466647: _Se {src: Window, i: {}, j: 5}
- closure_uid_720394987: 1

Mozilla Firefox

Buscar en Google o escribir una URL

Elements Console Sources Network

one-google-bar Filter

```
> window.name = 'Batman'

function favoriteCharacter () {
    console.log(this.name)
}

favoriteCharacter()
Batman

<- undefined

>
```

A screenshot of a browser's developer tools console. The URL bar shows 'one-google-bar'. The console tab is selected. The code is identical to the first snippet. An error message is displayed: 'Uncaught TypeError: Cannot read properties of undefined (reading 'name')' at favoriteCharacter (<anonymous>:6:22) at <anonymous>:9:1'. Below the console, a link titled 'Tipos de binding' is visible.

```
> *use strict*
window.name = 'Batman'
function favoriteCharacter () {
    console.log(this.name)
}
favoriteCharacter()
✖ Uncaught TypeError: Cannot read properties of undefined (reading 'name')
    at favoriteCharacter (<anonymous>:6:22)
    at <anonymous>:9:1
```

Tipos de binding

Publicado el 24 de noviembre de 2023

Clase

Binding en JavaScript se refiere a cómo la palabra clave this está vinculada o asociada en una función. Así que, entender los diferentes tipos de "binding" es esencial para comprender cómo se comporta this en diferentes situaciones. Veamos los cinco tipos principales de "binding".

Implicit Binding:

Ocurre cuando se invoca un método de un objeto, y this se vincula al objeto que contiene el método.

Ejemplo:

Implicit Binding:

Ocurre cuando se invoca un método de un objeto, y this se vincula al objeto que contiene el método.

Ejemplo:

```
const person = {
  name: 'Adam',
  greet: function() {
    console.log(`Hello, I'm ${this.name}`);
  }
};

person.greet(); // Output: Hello, I'm Adam
```

New Binding:

Ocurre cuando una función se invoca con la palabra clave new, creando así un nuevo objeto y vinculando this a ese objeto.

Ejemplo:

```
function Person(name) {
  this.name = name;
}

const adam = new Person('Adam');
console.log(adam.name); // Output: Adam
```

Lexical Binding:

Ocurre cuando se utiliza this en una función dentro de otra función. En este caso, this se vincula al contexto léxico de la función exterior.

Ejemplo:

```
const person = {
  name: 'Adam',
  greet: function() {
    const innerFunction = () => {
      console.log(`Hello, I'm ${this.name}`);
    }
  }
};
```

```

        }
    }

person.greet(); // Output: Hello, I'm Adam

```

Window Binding:

Ocurre cuando ninguna de las reglas anteriores se aplica y this se vincula al objeto global (por ejemplo, window en el navegador).

Ejemplo (ejecutar en el navegador):

```

function showName() {
    console.log(this.name);
}

window.name = 'Adam';
showName(); // Output: Adam

```

- Expresiones vs Sentencias

Expresiones

```

'Hola' // Produce 'Hola'
2 + 3 // Produce 5
6      // Produce 6

```

3 * (2 + 1)
#5

5 expresiones en total

```
const numeroEntero = 1
```

```
const suma
```

Declaración

- Proyecto: Crea biografías de personajes con JavaScript

```

1 Welcome   JS proyectobiografias.js U
1 Class > JS proyectobiografias.js > ...
1 //create powerpuffgirl object
2 function Powerpuffgirl(name,color,superpower){this.name=name
3     this.color=color
4     this.superpower=superpower
5     this.isleader=false
6     this.displayInfo=function(){console.log(`Powerpuff Girl Information:
7         Name: ${this.name}
8         Color: ${this.color}
9         Superpower: ${this.superpower}
10        ${this.isleader?'Leader: Yes':'Leader: No'}
11    `)}
12     this.becomeLeader=function(){this.isLeader=true
13     console.log(`${this.name} has become the leader of the Powerpuff Girls !`)}
14     const blossom=new Powerpuffgirl('Blossom','Pink','Ice Breath')
15     const buttercup=new Powerpuffgirl('Buttercup','Green','Super Strength')

```

```

16 const bubbles=new Powerpuffgirl('Bubbles','blue','Flight')
17 blossom.displayInfo()
18 buttercup.displayInfo()
19 bubbles.displayInfo()
20 blossom.becomeLeader()
21 blossom.displayInfo()
22 buttercup.displayInfo()
23 bubbles.displayInfo()

```

- Operadores de comparación

The screenshot shows a code editor interface with a sidebar containing a search history. The main area displays a file named 'index.js' with the following content:

```

1  /*
2
3  Operadores de comparación
4
5 */
6 ===
7 ====
8 ====
9
10

```

==== pra valor y tipo de datos
!= diferente != diferente y tipo de dato

- Operadores lógicos

The screenshot shows a code editor interface with a sidebar containing a search history. The main area displays a file named 'operadoreslogicos.js' with the following content:

```

1 /*
2     operadores logicos
3     y &&
4     o ||
5     negacion !
6 */
7
8 const a=10;
9 const b=20;
10 const c="10";
11 a==b && a==c
12 !b || a==c
13 !(a==c)

```

- Ejecución condicional: if

The screenshot shows a code editor interface with a sidebar containing a search history. The main area displays a file named 'if.js' with the following content:

```

1 let nombre="Nico";
2 if(nombre==="Diego"){
3     console.log("Hola Diego");
4 }else if(nombre==="Nico"){
5     console.log("Hola Nico");
6 }else{
7     console.log("Nombre no encontrado");
8 }

```

- Proyecto: Adivina el número

The screenshot shows a code editor interface with a sidebar containing a search history. The main area displays a file named 'proyecto_Adivinaelnumero.js' with the following content:

```

1 const numeroSecreto=Math.floor(Math.random()*10+1);
2 const numeroJugador=parseInt(
3     prompt("Adivina el numero secreto entre el 1 al 10"));
4
5 console.log("Este es el numero con el que juegas ${numeroJugador}");
6 if(numeroJugador==numeroSecreto){
7     console.log("¡Felicitaciones, adivinaste el numero secreto!");
8 }else if(numeroJugador<numeroSecreto){
9     console.log("El numero es demasiado bajo, intenta de nuevo");
10 }else{
11     console.log("El numero es muy alto, intenta de nuevo");
12 }

```

- Ternario

The screenshot shows a code editor interface with a sidebar containing a search history. The main area displays a file named 'ternario.js' with the following content:

```

1 const edad=17;
2 const mayorDeEdad=(edad>18)? "Es mayor de edad": "No es mayor de edad";
3 console.log(mayorDeEdad);

```

```
1 const mensaje = "El valor es menor de cero, no puedes pagar";
2 const mensaje = "El valor es mayor de cero, puedes pagar";
3 console.log(mensaje);
```

- Ejecución condicional: switch

```
Clase > JS switch.js > ...
1 // switch(expresión) {
2 //   case valor1:
3 //     // código a ejecutar
4 //     break;
5 //   case valor2:
6 //     // código a ejecutar
7 //     break;
8 //   case valor1:
9 //     // código a ejecutar
10 //    break;
11 //   case valor2:
12 //     // código a ejecutar
13 //     break;
14 //   default:
15 //     // código
16 // }

let expr = "Uvas";

switch (
expr // ===
) {
  case "Naranjas":
    console.log("Las naranjas cuestan $20 el kilo");
    break;
  case "Manzanas":
    console.log("Las manzanas cuestan $43 el kilo");
    break;
  case "Plátanos":
    console.log("El plátano esta en $30 el kilo");
    break;
  case "Mangos":
  case "Papayas":
    console.log("Los mangos y las papayas cuestan $ 25 pesos el kilo");
    break;
  default:
    console.log(`Lo siento, no contamos con ${expr}`);
}

console.log("¿Hay algo más que deseas?");
```

- For

```
38.Clase > JS for.js > ...
1 // for (variable; condición; incremento) {
2 //   código a ejecutar
3 //
4
5 let list = ["eat", "sleep", "code", "repeat"];
6
7 for (let i = 0; i < list.length; i++) {
8   console.log(list[i]);
9 }
```

- Loop: forEach

```
40.Clase > JS forof.js > ...
1 /*
2
3 array.forEach((item) => {
4   // código a ejecutar
5 })
6
7 */
8
9 let list = ["eat", "sleep", "code", "repeat"];
10
11 list.forEach((item) => {
12   console.log(item);
13 });
```

- Loop: for of

40.Clase > **JS** forof.js > ...

```

1  /*
2
3  for of arrays, strings [algo]
4
5  var (variable de objeto) {
6  |   codigo
7  }
8
9 */
10
11 let canasta = ["manzana", "pera", "naranja", "uva"];
12
13 for (fruta of canasta) {
14 |   console.log(fruta);
15 }

```

- Loop: for in

41.Clase > **JS** forin.js > ...

```

1 const listaDeCompras={manzana:5,pera:3,naranja:2,uva:1};
2 for(fruta in listaDeCompras){console.log(fruta);}
3 for(fruta in listaDeCompras){console.log(`${fruta}: ${listaDeCompras[fruta]}`);}
4 for(fruta of listaDeCompras){console.log(fruta);}

```

Loop: while

42.Clase > **JS** while.js > ...

```

1 let contador=0;
2 while(contador<10){
3 |   console.log(contador);
4 |   contador++;
5 }

```

- Loop: do while

43.Clase > **JS** dowhile.js > ...

```

1 let contador=0;
2 do{
3 |   console.log(contador);
4 |   contador++;
5 }while(contador<10);

```

- Proyecto: Juego adivina la palabra

44.Clase > **JS** juego_AdivinalaPalabra.js > jugarAdivinalaPalabra

```

1 let palabraOculto="javascript";
2 let intentos=3;
3 function verificarSuposicion(suposicion,palabraOculto){
4   if(suposicion.toLowerCase()===palabraOculto){
5     return true;
6   }
7   return false;
8 }
9 function jugarAdivinalaPalabra(){
10   alert("Bienvenido a jugar adivina la palabra oculta");
11   alert("Tienes 3 intentos para adivinar la palabra");
12   alert("-pista- la palabra oculta es un lenguaje de programacion");
13   while(intentos>0){
14     let suposicion=prompt("Adivina la palabra: ");
15     if(verificarSuposicion(suposicion,palabraOculto)){
16       alert("Correcto! Has adivinado la palabra!");
17       break;
18     }else{
19       intentos--;
20       if(intentos>0){
21         alert("Incorrecto, Aun tienes ${intentos} intento restantes");
22       }else{
23         alert("Agotaste tus intentos, la palabra era ${palabraOculto}");
24       }
25     }
26   }
27 }

```

```

28 jugarAdivinaLaPalabra();

```

• Introducción a Arrays

```

1 Welcome JS uintroductionarrays.js U
45.Clase > JS uintroductionarrays.js > ...
1 const fruits=Array('apple','banana','orange')
2 console.log(fruits)
3 const justOneNumber=12
4 console.log(justOneNumber)
5 const number=Array(1,2,3,4,5)
6 console.log(number)
7 const oneNumber=[4]
8 console.log(oneNumber)
9 const emptyArray=[]
10 console.log(emptyArray)
11 const sports=['soccer','tennis','rugby']
12 console.log(sports)
13 const recipeIngredients=['Flour',true,2,{ingredient:'Milk',quantity:'1 cup'},false]
14 console.log(recipeIngredients)
15 const firstFruit=fruits[0]
16 console.log(firstFruit)
17 const numberOfruits=fruits.length
18 console.log(numberOfruits)
19 fruits.push('watermelon')
20 console.log(fruits)
21 const newFruits=fruits.concat(['grape','kiwi'])
22 console.log(fruits)
23 console.log(newFruits)
24 const isArray=Array.isArray(fruits)
25 console.log(isArray)
26 const numbersArray=[1,2,3,4,5]
27 let sum=0
28 for(let i=0;i<numbersArray.length;i++){sum+=numbersArray[i]}
29 console.log(sum)

```

- Modificación básica del final con push(), pop()

Ampliando Arrays con push() 🎨

Añade uno o más elementos al **final** de un array y devuelve la nueva longitud del array.

Contrayendo Arrays con pop() ⏪

Elimina el **último** elemento de un array.

```

1 Welcome JS uintroductionarrays.js U
2 JS push_pop.js U
47.Clase > JS push_pop.js > ...
1 // Methods that modify the original array (Mutability).
2
3 // push()
4
5 const countries = ['USA', 'Canada', 'UK']
6 const newCountries = countries.push('Germany', 'Australia')
7
8 console.log(countries)
9 console.log(newCountries)
10
11 // pop()
12
13 const removedCountry = countries.pop()
14
15 console.log(countries)
16 console.log(removedCountry)
17
18 // Exercise: Managing a Stack
19
20 let bookCart = []
21
22 const ADD_TO_CART_ACTION = 'addToCart'
23 const REMOVE_FROM_CART_ACTION = 'removeFromCart'
24 const VIEW_CART_ACTION = 'viewCart'
25
26 function viewCart () {
27   console.log('Current Cart of Books: ', bookCart)
28 }
29
30 function performCartActions (action, newBook) {
31   switch (action) {
32     case ADD_TO_CART_ACTION:
33       bookCart.push(newBook)
34       break;
35     case REMOVE_FROM_CART_ACTION:
36       bookCart.pop()
37       break;
38     case VIEW_CART_ACTION:
39       console.log(bookCart)
40       break;
41   }
42 }

```

• push.pop.js

```

1 Welcome JS uintroductionarrays.js U
2 JS push_pop.js U
47.Clase > JS push.pop.js > ...
26 function ViewCart () {
27
28 }
29
30 function performCartActions (action, newBook) {
31   switch (action) {
32     case ADD_TO_CART_ACTION:
33       bookCart.push(newBook)
34       break;
35     case REMOVE_FROM_CART_ACTION:
36       bookCart.pop()
37       break;
38     case VIEW_CART_ACTION:
39       console.log(bookCart)
40       break;
41   }
42 }

```

```
36 if (bookCart.length === 0) {
37   console.log('Cart is empty. No books to remove.')
38 } else {
39   const removedBook = bookCart.pop()
40   console.log(`Removed book from the cart: ${removedBook}`)
41 }
42 break;
43 case VIEW_CART_ACTION:
44   viewCart()
45   break;
46 default:
47   console.log('Invalid action. Please choose a valid option.')
48 }
49 }
50
51 performCartActions(ADD_TO_CART_ACTION, 'Think like a monk')
52 performCartActions(VIEW_CART_ACTION)
53 performCartActions(ADD_TO_CART_ACTION, 'Ego is the enemy.')
54 performCartActions(VIEW_CART_ACTION)
55 performCartActions(REMOVE_FROM_CART_ACTION)
```

- Modificación del principio con shift(), unshift()

Despejando el camino con shift()

```
49.Clase > JS shift_unshift.js > ...
1 ~ // Methods that modify the original array (Mutability).
2
3 // shift()
4
5 const colors = ['yellow', 'blue', 'red']
6 const removedColors = colors.shift()
7
8 console.log(colors)
9 console.log(removedColors)
10
11 // unshift()
12
13 const newColors = colors.unshift('pink', 'purple')
14
15 console.log(colors)
16 console.log(newColors)
17
18 // = . . .
19 function managePlaylist(playlist: any, newSong: any): any
20
21 function managePlaylist (playlist, newSong) {
22     playlist.shift()
23     playlist.unshift(newSong)
24     return playlist
25 }
26
const initialPlaylist = ['Sweater Weather', 'What you know', 'Eventually']
const newSongToAdd = 'The Adults Are Talking'

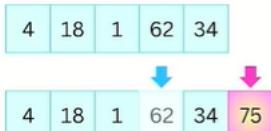
const updatedPlaylist = managePlaylist(initialPlaylist, newSongToAdd)

console.log('Initial playlist: ', initialPlaylist)
console.log('New song to add: ', newSongToAdd)
console.log('Updated playlist: ', updatedPlaylist)
```

- Modificación avanzada con splice(), reverse(), sort(), fill()

Esculpiendo con splice()

Cambia el contenido de un array eliminando elementos existentes y/o agregando nuevos elementos



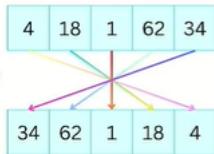
```

Welcome JS splice_reverse_sort_fill.js •
50.Clase > JS splice_reverse_sort_fill.js ...
1 const vegetables=['carrot','broccoli','spinach','tomato']
2 const removedVegetables=vegetables.splice(2,1,'cucumber','onion')
3 console.log(vegetables)
4 console.log(removedVegetables)
5 const numbers=[1,2,3,4,5]
6 const reversedNumbers=numbers.reverse()
7 console.log(numbers)
8 console.log(reversedNumbers)
9 const unsortedNumbers1=[4,18,1,62,34]
10 const unicodeSortedNumbers=unsortedNumbers1.sort()
11 console.log(unsortedNumbers1)
12 console.log(unicodeSortedNumbers)
13 const unsortedNumbers2=[4,18,1,62,34]
14 const sortedNumbers=unsortedNumbers2.sort((a,b)=>a-b)
15 console.log(unsortedNumbers2)
16 console.log(sortedNumbers)
17 const cities=['New York','Paris','Tokyo','London']
18 const sortedCities=cities.sort()
19 console.log(cities)
20 console.log(sortedCities)
21 const ages=[22,35,45,50]
22 console.log(ages.fill(0,2,4))
23 console.log(ages.fill(15,1))
24 console.log(ages.fill(15))

```

Invirtiendo el rumbo con reverse()

Invierte el orden de los elementos de un array *in place*. El primer elemento pasa a ser el último y el último pasa a ser el primero.



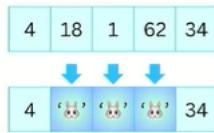
```

clase-splice-reverse-sort-fill.js — Platzi
JS clase-splice-reverse-sort-fill.js •
18
19 // sort() with number
20
21 const unsortedNumbers1 = [4, 18, 1, 62, 34]
22 const unicodeSortedNumbers = unsortedNumbers1.sort()
23
24 console.log(unsortedNumbers1)
25 console.log(unicodeSortedNumbers)
26
27 const unsortedNumbers2 = [4, 18, 1, 62, 34]
28 const sortedNumbers = unsortedNumbers2.sort((a, b) => a - b)
29 // 4 - 18 = -14
30 // 18 - 1 = 17
31

```

Llenando espacios con fill()

Cambia todos los elementos en un array por un valor estático, desde el índice start (por defecto 0) hasta el índice end (por defecto array.length).



• Proyecto: Juego de cartas

```

Welcome JS proyecto.juegodecartas.js •
51.Clase > JS proyecto.juegodecartas.js ...
1 const deck=[...Array(52)].map((v,i)=>i+1).sort((a,b)=>a-b)
2 const numCards=Math.floor(Math.random()*deck.length-1);const i=Math.floor(Math.random()*(51));
3 function dealCards(numCard)(const dealCards=deck.splice(0,numCard))
4 return dealCards();
5 shuffleDeck();
6 const player1Hand=dealCards(3)
7 const player2Hand=dealCards(3)
8 const player3Hand=dealCards(3)
9 console.log(`Player 1 Hand: ${player1Hand}`);
10 console.log(`Player 2 Hand: ${player2Hand}`);
11 console.log(`Player 3 Hand: ${player3Hand}`);

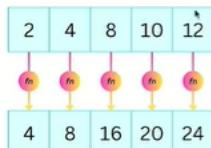
```

```
JS clase-juego-de-cartas.js •
1 // Exercise: Card Game Implementation
2
3 const deck = ['♦', '♦', '♥', '♦', '♦', '♦', '♥', '♦', '♦', '♦', '♦', '♦']
4
5 // Fisher-Yates Algorithm
6 function shuffleDeck () {
7   for (let i = deck.length - 1; i > 0; i--) {
8     const j = Math.floor(Math.random() * (i + 1))
9     [deck[i], deck[j]] = [deck[j], deck[i]]
10   }
11 }
```

- Iteración con map() y forEach()

Transformando con map() 🎨

Permite aplicar una función a cada elemento de un array y construir un nuevo array con los resultados.



```
clase-map-forEach.js — Platzi
JS clase-map-forEach.js •
1 // Methods that iterate over an array.
2 // Methods that DO NOT modify the original array (Immutability).
3
4 // map()
5
6 const numbers = [1, 2, 3, 4, 5]
7 const squaredNumbers = numbers.map(num => num * num)
8
9 console.log(numbers)
10 console.log(squaredNumbers)
```

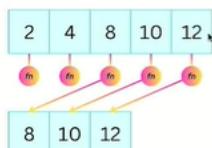
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL zsh + ✎

```
Platzi node clase-map-forEach.js
[ 1, 2, 3, 4, 5 ]
[ 1, 4, 9, 16, 25 ]
```

Filtrado y reducción con filter() y reduce()

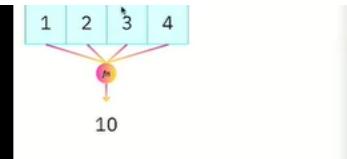
Seleccionando con filter() 🔍

Crea un nuevo array con elementos que cumplen una condición dada por una función.



Sintetizando datos con reduce() 📊

Ejecuta una función reductora sobre cada elemento de un array, devolviendo como resultado un único valor.



JS clase-filter-reduce.js ×

```

1 // Methods that iterate over an array.
2 // Methods that DO NOT modify the original array (Immutability).
3
4 // filter()
5
6 const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
7 const evenNumbers = numbers.filter(number => number % 2 === 0)
8
9 console.log(numbers)
10 console.log(evenNumbers) | I

```

```

base > $ filter_reduce.js > ...
// reduce() A- case 2

const words = ['apple', 'banana', 'hello', 'bye', 'banana', 'bye', 'bye']

const wordFrequency = words.reduce((accumulator, currentValue) => {
  if (accumulator[currentValue]) {
    accumulator[currentValue]++;
  } else {
    accumulator[currentValue] = 1;
  }
  return accumulator
}, {})

console.log(wordFrequency)

// Exercise: Passing Grade Average

const grades = [85, 92, 60, 78, 95, 66, 88, 50, 94]

const passingGrades = grades.filter(grade => grade >= 70)

const averagePassingGrade = passingGrades.reduce((sum, grade) => sum + grade, 0) / passingGrades.length

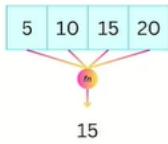
console.log('Original Grades: ', grades)
console.log('Passing Grades: ', passingGrades)
console.log('Average Passing Grade: ', averagePassingGrade) | I

```

- Búsqueda de elementos con `find()` y `findIndex()`

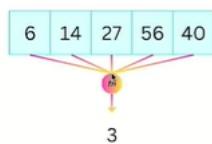
Descubriendo tesoros con `find()` 🏴

Devuelve el valor del primer elemento del array que cumple la función de prueba proporcionada.



Localizando el mapa con `findIndex()` 🗺📍

Devuelve el índice del primer elemento en un array que satisface una condición proporcionada en forma de función. Si no encuentra ningún elemento que cumpla la condición, devuelve -1.



Proyecto: Análisis de transacciones

```

Welcome to the terminal! 🚀
File: Análisis de transacciones.js | Line: 10 | Column: 1 | Size: 1000 | Encoding: UTF-8 | Syntax: JavaScript | Linter: ESLint | Linting: Off | Watch: Off | Run: Off | Help: Off | Exit: Ctrl+C
File: Análisis de transacciones.js | Line: 10 | Column: 1 | Size: 1000 | Encoding: UTF-8 | Syntax: JavaScript | Linter: ESLint | Linting: Off | Watch: Off | Run: Off | Help: Off | Exit: Ctrl+C
1 const transactions=[
2   {id:1,description:'Salary Deposit',amount:200},
3   {id:2,description:'Utility Bill Payment',amount:200},
4   {id:3,description:'Utility Bill Payment',amount:-100},
5   {id:4,description:'Online Purchase',amount:-100}
6   ];
7 const totalBalance=transactions.reduce((acc,transaction)=>acc+transaction.amount,0)
8
9 console.log(`Total Balance: ${totalBalance}`) | I

```

```

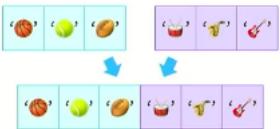
7 console.log('Total balance: ', totalBalance)
8
9 const largestTransaction = transactions.reduce((maxTransaction, transaction) => {
10   return Math.abs(transaction.amount) > Math.abs(maxTransaction.amount) ? transaction : maxTransaction
11 })
12 console.log(`The largest transaction was ${largestTransaction.description} of ${largestTransaction.amount}`)
13
14 const purchaseTransactions = transactions.filter(transaction => transaction.amount < 0)
15 console.log(`Purchase Transactions: ${purchaseTransactions.length}`)
16
17 const specificTransaction = transactions.find(transaction => transaction.description === 'Online Purchase')
18 console.log(`Specific Transaction: ${specificTransaction}`)
19
20 const index = transactionWithAmount.transactions.findIndex(transaction => transaction.amount === -30)
21 console.log(`Index transaction with amount -30: ${index}`) // Index transaction with the amount
22 transactions.forEach(transaction => (transaction.amount < 0) ? transaction.description = 'Expenses' : transaction.description = 'Income')
23 console.log(`Updated Transactions: ${transactions}`)

```

- Unir y entrelazar con concat(), spread operator y join()

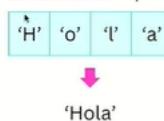
Uniendo elementos con concat()

Une dos o más arrays.



Entrelazando con join()

Concatena todos los elementos de un array en una cadena de texto, separados por un delimitador especificado.



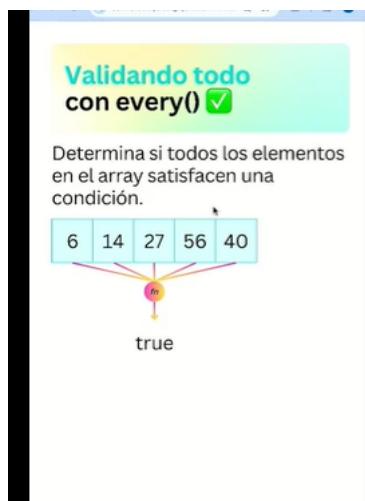
```

Welcome JS concat_join.js U ●
60.Clae > JS concat_join.js > ...
1  // Methods that DO NOT modify the original array (Immutability).
2
3  const morseCode1 = ['.-.-.', '---'] // H O
4  const morseCode2 = ['.-.-.', '-.-'] // L A
5
6  // Combine with concat() ¯ way 1
7
8  const morseCodeMessage = morseCode1.concat(morseCode2)
9
10 console.log(morseCode1)
11 console.log(morseCode2)
12 console.log(morseCodeMessage)
13
14 // Combine with concat() ¯ way 2
15
16 const morseCodeMessage2 = [].concat(morseCode1, morseCode2)
17
18 console.log(morseCode1)
19 console.log(morseCode2)
20 console.log(morseCodeMessage)
21
22 // Combine with Spread Operator
23
24 function combineMorseMessages (morseCode1, morseCode2) {
25   | console.log(...morseCode1, ...morseCode2)
26 }
27
28 combineMorseMessages(morseCode1, morseCode2)
29
30 const numbers = [1, 2, 3]
31 const string = 'string'
32
33
34
35 // join()
36
37 const morseCodeMessageString = morseCodeMessage.join('')
38
39 console.log(morseCode1)
40 console.log(morseCode2)

```

```
41 console.log(morseCodeMessageString)
```

Verificación y evaluación con every() y some()



```
61.Clase > JS every_some.js > ...
1 // Methods that DO NOT modify the original array (Immutability).
2
3 const ages = [21, 25, 30, 19, 22]
4
5 // every()
6
7 const allAreAdults = ages.every(age => age > 20)
8
9 console.log(ages)
10 console.log(allAreAdults)
11
12 // some()
13 const atLeastOneIsOver30 = ages.some(age => age > 30)
14
15 console.log(ages)
16 console.log(atLeastOneIsOver30)
```

• Métodos de búsqueda con includes(), indexOf() y lastIndexOf()

```
clase-includes-indexOf-lastIndexIndexOf.js
1 // includes() with numbers
2
3 const numbers = [1, 2, 3, 4, 5]
4
5 const result1 = numbers.includes(3)
6 console.log(result1)
7
8 const result2 = numbers.includes(8)
9 console.log(result2)
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL zsh + 
● → Platzi node clase-includes-indexOf-lastIndexIndexOf.js
true
● → Platzi node clase-includes-indexOf-lastIndexIndexOf.js
true
false
○ → Platzi
```

```
2.Clase y 63 > JS includes_indexOf_lastIndexOf.js > ...
9 console.log(result2)
10
11 // indexOf()
12
13 const fruits = ['apple', 'cherry', 'grape', 'mango']
14
15 const index1 = fruits.indexOf('grape')
16 console.log(index1)
17
18 const index2 = fruits.indexOf('blueberry')
19 console.log(index2)
20
21 // lastIndexOf()
22
23 const numbersAgain = [2, 4, 6, 8, 10, 6]
24
25 const lastIndex1 = numbersAgain.lastIndexOf(6)
26 console.log(lastIndex1)
27
28 const lastIndex2 = numbersAgain.lastIndexOf(3)
29 console.log(lastIndex2)
30
31.Clase y 63 > JS includes_indexOf_lastIndexOf.js > ...
32
33 const numbersAgain = [2, 4, 6, 8, 10, 6]
34
35 const lastIndex1 = numbersAgain.lastIndexOf(6)
36
```

```

40 console.log(lastIndex)
41
42 const lastIndex2 = numbersAgain.lastIndexOf(3)
43 console.log(lastIndex2)
44
45 // Exercise A- Findind Substring Indices
46
47 const stringArray = ['apple', 'banana', 'orange', 'grape', 'banana', 'kiwi']
48 const target = 'banana'
49
50
51 function findStringIndicesInArray (array, target) {
52   const result = {
53     includesTargetString: false,
54     firstOccurrenceIndex: -1,
55     lastOccurrenceIndex: -1
56   }
57
58   array.forEach((element, index) => {
59     if (element.includes(target)) {
60       result.includesTargetString = true
61       result.firstOccurrenceIndex = array.indexOf(target)
62       result.lastOccurrenceIndex = array.lastIndexOf(target)
63     }
64   })
65
66   return result
67 }

```

Finding Substring Indices

Given an array of strings and a target string, write a function to determine if the target string is present in the array. If it is present, return the index of the first occurrence and the index of the last occurrence; otherwise, return -1.

```

35
36   function findStringIndicesInArray (array, target) {
37     const result = {
38       includesTargetString: false,
39       firstOccurrenceIndex: -1,
40       lastOccurrenceIndex: -1
41     }
42
43     array.forEach((element, index) => {
44       if (element.includes(target)) {
45         result.includesTargetString = true
46         result.firstOccurrenceIndex = array.indexOf(target)
47         result.lastOccurrenceIndex = array.lastIndexOf(target)
48       }
49     })
50
51     return result
52   }

```

- Slice

```

1 // slice()
2
3 const animals = ['ant', 'bison', 'camel', 'duck', 'elephant']
4
5 console.log(animals.slice(2))
6 console.log(animals.slice(2, 4))
7 console.log(animals.slice(1, 6))
8 console.log(animals.slice(-2))
9 console.log(animals.slice(2, -1))
10 console.log(animals.slice())

```

- Spread operator: casos de uso

```

1 const originalArray=[1,2,3,4,5]
2 const copyOfAnArray=[...originalArray]
3 console.log(originalArray)
4 console.log(copyOfAnArray)
5 const array1=[1,2,3]
6 const array2=[4,5,6]
7 const combinedArray=[...array1,...array2]
8 console.log(array1)
9 console.log(array2)
10 console.log(combinedArray)
11 const baseArray=[1,2,3]
12 const arrayWithAdditionalElements=[...baseArray,4,5,6]
13 console.log(baseArray)

```

```
14 console.log(arrayWithAdditionalElements)
15 function sum(a,b,c){return a+b+c}
16 const numbers=[1,2,3]
17 const result=sum(...numbers)
18 console.log(result)
```

Arrays Bidimensionales

```
base > $ arr=(([{"id":1,"x":2,"y":3}, {"id":2,"x":4,"y":6}, {"id":3,"x":7,"y":9}])  
let array1D=[[1,2,3],[4,5,6],[7,8,9]]  
let matrix=[[1,2,3],[4,5,6],[7,8,9]]  
matrix[1][2]=4  
console.log(matrix)  
let value=matrix[0][1]  
console.log(value)  
for(let i=0;i<matrix.length;i++){for(let j=0;j<matrix[i].length;j++)console.log(matrix[i][j])}  
function findElement (matrix,element){for(let i=0;i<matrix.length;i++){for(let j=0;j<matrix[i].length;j++){if(matrix[i][j]==element) return true}}}  
function findDuplicate (matrix,i){  
function duplicateMatrix(matrix){let newMatrix=[];  
for(let i=0;i<matrix.length;i++){newMatrix[i]=[...matrix[i]]};  
return newMatrix}  
console.log(duplicateMatrix(matrix))}
```

Proyecto: Encuentra al ganador del torneo

```
  Welcome          JS ejercicio_ganadortomeojs U ●
67.Clase y 68 > JS ejercicio_ganadortomeojs > ...
1   function tournamentWinner(competitions,results){const scores={}
2     let winner=''
3     for(let i=0;i<competitions.length;i++){const [home,away]=competitions[i]
4       const winningTeam=results[i]==0?away:home
5       scores[winningTeam]=(scores[winningTeam]||0)+3
6       if((winner||scores[winningTeam]>scores[winner])&(winner!=winningTeam))
7         return winner
8     }
9     const competitions=[['JavaScript','C#'],['C#','Python'],['Python','JavaScript']]
10    const results=[0,0,1]
10  console.log(tournamentWinner(competitions,results))
```

- Anatomía de un Objeto

```
VS Code Welcome | JS Anatomia de un Objeto.js U ●
70 Clase > JS Anatomia de un Objeto.js > ...
1 const persona={
2   nombre:"John",
3   edad:30,
4   direccion:{
5     calle:"Av Insurgentes 187",
6     ciudad:"CDMX",),
7     saludar(){
8       console.log(`hola, mi nombre es ${persona.nombre}`);
9     },
10   };
11 console.log(persona);
12 console.log(persona.nombre);
13 persona.saludar();
14 persona.telefono="555-555-5555";
15 console.log(persona.telefono);
16 persona.despedir={()>{console.log("Adios");}};
17 persona.despedir();
18 console.log(persona.direccion.calle);
19 delete persona.telefono;delete persona.despedir();
```

```
index.js

obj&Classes > objetos > index.js > persona > direccion

  1 // Objeto
  2   propiedad: valor,
  3   propiedad: valor,
  4   propiedad: valor
  5   Metodos
  6 }
  7
  8 */
  9
10 const persona = [
11   nombre: "John",
12   edad: 30,
13   direccion: {
14     calle: "Av Insurgente 187",
15     ciudad: "CDMX",
16   },
17   saludar() {
18     console.log(`hola, mi nombre es ${persona.nombre}`)
19   }
20 }

21
22
23
24
25

6:32 / 6:48
```

- Trabajando con objetos

```
    Welcome JS Trabajandoconobjetos.js U ●  
71.Clase > JS Trabajandoconobjetos.js > ...  
1 const persona={  
2   nombre:"John",  
3   edad:30,  
4   direccion:{
```

```

5     calle:"Av Insurgente 187",
6     ciudad:"CDMX",},
7     saludar(){
8     console.log(`hola, mi nombre es ${persona.nombre}`);
9   },
10 };
11 console.log(persona);
12 console.log(persona.nombre);
13 persona.saludar();
14 persona.telefono="555-555-5555";
15 console.log(persona.telefono);
16 persona.despedir=()=>{
17   console.log("Adios");
18 }
19 persona.despedir();
20 console.log(persona.direccion.calle);
21 delete persona.telefono;delete persona.despedir();

```

- Función constructora

```

1 Welcome JS funcionconstructora.js U
2 Clase > JS funcionconstructora.js ...
3
4 function Persona(nombre,apellido,edad){
5   this.nombre=nombre;
6   this.apellido=apellido;
7   this.edad=edad;
8 }
9 const persona1=new Persona(
10   "Juan","Perez",20);
11
12 console.log(persona1);
13 persona1.nacionalidad="Mexicano";
14
15 const persona2=new Persona(
16   "Diego","De Granda",35);
17 console.log(persona2);
18
19 Persona.prototype.saludar=function(){
20   console.log(`Hola, me llamo ${this.nombre} ${this.apellido}`);
21 };persona1.saludar();persona2.saludar();

```

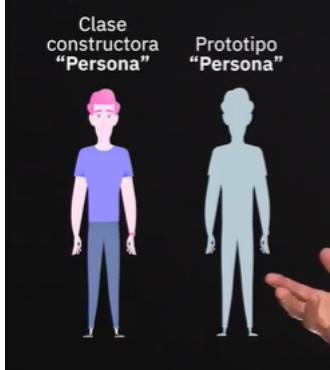
- ¿Qué es una clase?

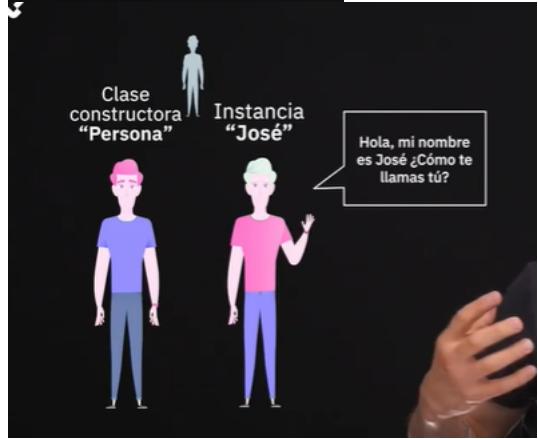
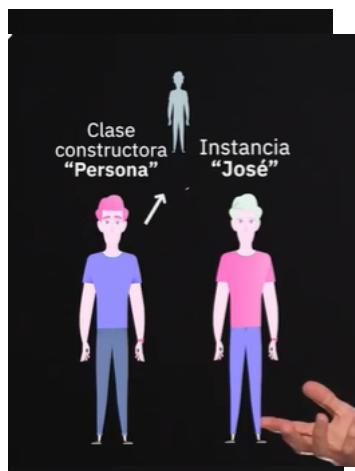
```

1 Welcome JS dsejs U X
2 Clase > JS dsejs > Person
3
4 class Persona{
5   constructor(nombre,edad){
6     this.nombre=nombre;this.edad=edad;
7   }
8   saludar(){
9     console.log(`Hola, mi nombre es ${this.nombre} y tengo ${this.edad} años.`);
10 }
11
12 const persona1=new Persona("Mariana",25);persona1.saludar();

```

- Prototipos y herencias





- Herencia en la práctica

The screenshot shows a browser's developer tools console tab titled "Welcome" with the URL "herencia.js". The console output is as follows:

```
File Edit Selection ... ← → JavaScriptCuso
Welcome JS herencia.js ●
75.Clase > JS herencia.js ... 
1 class Animal{
2     constructor(nombre,tipo){
3         this.nombre=nombre;
4         this.tipo=tipo;
5     }
6     emitirSonido(){
7         console.log("El animal emite un sonido");
8     }
9 }
10 class Perro extends Animal{
11     constructor(nombre,tipo,raza){
12         super(nombre,tipo);
13         this.raza=raza;
14     }
15     emitirSonido(){
16         console.log("El perro ladra");
17     }
18     correr(){
19         console.log(`${this.nombre} corre alegremente`);
20     }
21 }
22 const perro1=new Perro("Bobby","Perro","Pug");
23 console.log(perro1);
24 perro1.correr();
25 perro1.emitirSonido();
26 perro1.nuevoMetodo=function(){
27     console.log("Este es un metodo");
28 };
29 Perro.prototype.segundoMetodo=function(){
30     console.log("Es otro nuevo metodo");
31 };
32 Animal.prototype.tercerMetodo=function(){
```

```
75.Clase > JS herencia.js > ...
10  <class Perro extends Animal{
11    ...
12    }
13    }
14    }
15    emitirSonido(){
16      console.log("El perro ladra");
17    }
18    correr(){
19      console.log(` ${this.nombre} corre alegremente`);
20    }
21  }
22 const perro1=new Perro("Bobby","Perro","Pug");
23 console.log(perro1);
24 perro1.correr();
25 perro1.emitirSonido();
26 perro1.nuevoMetodo=function(){
27   console.log("Este es un metodo");
28 };
29 Perro.prototype.segundoMetodo=function(){
30   console.log("Es otro nuevo metodo");
31 };
32 Animal.prototype.tercerMetodo=function(){
33   console.log("Un metodo mas");
34 };
```

Prototipos en la práctica

```
}
```

```
const perro1=new Perro("Bobby","Perro","Pug");
console.log(perro1);
perro1.correr();
perro1.emitirSonido();
perro1.nuevoMetodo=function(){
  console.log("Este es un metodo");
};
Perro.prototype.segundoMetodo=function(){
  console.log("Es otro nuevo metodo");
};
Animal.prototype.tercerMetodo=function(){
  console.log("Un metodo mas");
};
```

```
nombre: "Bobby"
▶ nuevoMetodo: f ()
  raza: "Pug"
  tipo: "Perro"
  ▶ [[Prototype]]: Animal

> let currentPrototype =
Object.getPrototypeOf(perro1);

for (
;
currentPrototype !== null;
currentPrototype =
Object.getPrototypeOf(currentPrototype)
) {
  console.log(currentPrototype);
}
```

Console What's New X

Highlights from the Chrome 119 update

THIS

```
77.Clase > JS this.js > nuevoMetodo
1  class Persona{
2    constructor(nombre,edad){
3      this.nombre=nombre;this.edad=edad;
4    }
5  }
6  const persona1=new Persona("Alice",25);
7  console.log(persona1);
8  persona1.nuevoMetodo=function(){
9    console.log(`Mi nombre es ${this.nombre}`);
10 };


```

```
js index.js •
obj&Classes > clases > this > index.js > Persona > constructor
1  /*
2  this --- class
3
```

```
4  this --- objeto --- class
5  */
6
7  class Persona {
8    constructor(nombre, edad) {
9      this.nombre = nombre;
10   this.edad = edad;
11 }
12 }
13
14 const personal1 = new Persona("Alice", 25)
```

```
obj&Classes > retoFinal > index.js > ...
1  /*
2  Requerimientos del reto:
3
4  1. El usuario debe poder ingresar su usuario y contraseña
5  2. El sistema debe ser capaz de validar si el usuario y
6  contraseña ingresados por el usuario existen en la base de
7  datos
8  3. Si el usuario y contraseña son correctos, el sistema
9  debe mostrar un mensaje de bienvenida y mostrar el
timeline del usuario.
10 4. Si el usuario y contraseña son incorrectos, el sistema
11 debe mostrar un mensaje de error y no mostrar ningun
12 timeline.
13 */
14
```

```
78.Clase y 79 > JS proyectoredsocialjs > signin
1  const usersDatabase=[
2  {
3    username:"andres",
4    password:"123",
5  },
6  {
7    username:"caro",
8    password:"456",
9  },
10 {
11   username:"mariana",
12   password:"789",
13 },
14 ];
15 const usersTimeline=[
16 {
17   username:"Estefany",
18   timeline:"Me encata Javascript!",
19 },
20 {
21   username:"Oscar",
22   timeline:"Bebeloper es lo mejor!",
23 },
24 {
25   username:"Mariana",
26   timeline:"A mi me gusta mÁs el cafÁO que el tÁO",
27 },
28 {
29   username:"Andres",
30   timeline:"Q hoy no quiero trabajar",
31 },
32 ];
33
34 ];
35
36 const username=prompt("CuÁl es tu usuario?");
37 const password=prompt("CuÁl es tu contraseÁa?");
38 function usuarioExiste(username,password){
39   for(let i=0;i<usersDatabase.length;i++){
40     if(usersDatabase[i].username==username&&usersDatabase[i].password==password){
41       return true;
42     }
43   }
44   return false;
45 }
46
47 function signIn(username,password){
48   if(usuarioExiste(username,password)){
49     alert(`Bienvenido a tu cuenta ${username}`);
50     console.log(usersTimeline);
51   }
52 }
```

```
    }
    else{
        alert("Uups, usuario o contraseña incorrectos!");
    }
signIn(username,password);
```