Database: lms, Table: additional_readings, Purpose: Dumping data

id	title	link	topic_reference_id	content
1	Additional-11		4	this is additional

Database: lms, Table: admin_users, Purpose: Dumping data

id	name	username	password	email
4	Kabir	kabir12	qwertyuiop	kabir.behal7830@gmail.com
5	Kabir	kabir03	123456789	kabir.behal7830@gmail.com
6	nspl	nspl	123456789	nsplrtc@gmail.com

Database: lms, Table: assignments_assignments, Purpose: Dumping data

id	assignment_title	assignment_description	assignment_solution	assignment_output	assignment_author

Database: lms, Table: assignments_student, Purpose: Dumping data

id	fname	lname	username	password

Database: lms, Table: assignments_teacher, Purpose: Dumping data

id	fname	lname	username	password

Database: lms, Table: auth_group, Purpose: Dumping data

id	name
1	Admin_Group

Database: lms, Table: auth_group_permissions, Purpose: Dumping data

id	group_id	permission_id
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6
7	1	7
8	1	8
9	1	9
10	1	10
11	1	11
12	1	12
13	1	13
14	1	14
15	1	15
16	1	16
17	1	17
18	1	18
19	1	19
20	1	20
21	1	21
22	1	22
23	1	23
24	1	24
25	1	25
26	1	26
27	1	27
28	1	28
29	1	29
30	1	30
31	1	31
32	1	32
33	1	33
34	1	34
35	1	35
36	1	36
37	1	37
38	1	38
39	1	39
40	1	40
41	1	41
42	1	42
43	1	43
44	1	44
45	1	45
46	1	46
47	1	47
48	1	48
49	1	49
50	1	50
51	1	51
52	1	52
53	1	53
54	1	54
55	1	55
56	1	56
57	1	57
58	1	58
59	1	59
60	1	60
61	1	61
62	1	62
63	1	63
64	1	64
65	1	65
66	1	66
67	1	67
68	1	68
69	1	69

id	group_id	permission_id
70	1	70
71	1	71
72	1	72
73	1	73
74	1	74
75	1	75
76	1	76
77	1	77
78	1	78
79	1	79
80	1	80
81	1	81
82	1	82
83	1	83
84	1	84
85	1	85
86	1	86
87	1	87
88	1	88
89	1	89
90	1	90
91	1	91
92	1	92
93	1	93
94	1	94
95	1	95
96	1	96
97	1	97
98	1	98
99	1	99
100	1	100

Database: lms, Table: auth_permission, Purpose: Dumping data

id	name	content_type_id	codename
1	Can add log entry	1	add_logentry
2	Can change log entry	1	change_logentry
3	Can delete log entry	1	delete_logentry
4	Can view log entry	1	view_logentry
5	Can add permission	2	add_permission
6	Can change permission	2	change_permission
7	Can delete permission	2	delete_permission
8	Can view permission	2	view_permission
9	Can add group	3	add_group
10	Can change group	3	change_group
11	Can delete group	3	delete_group
12	Can view group	3	view_group
13	Can add user	4	add_user
14	Can change user	4	change_user
15	Can delete user	4	delete_user
16	Can view user	4	view_user
17	Can add content type	5	add_contenttype
18	Can change content type	5	change_contenttype
19	Can delete content type	5	delete_contenttype
20	Can view content type	5	view_contenttype
21	Can add session	6	add_session
22	Can change session	6	change_session
23	Can delete session	6	delete_session
24	Can view session	6	view_session
25	Can add assignments	7	add_assignments
26	Can change assignments	7	change_assignments
27	Can delete assignments	7	delete_assignments
28	Can view assignments	7	view_assignments
29	Can add student	8	add_student
30	Can change student	8	change_student
31	Can delete student	8	delete_student
32	Can view student	8	view_student
33	Can add teacher	9	add_teacher
34	Can change teacher	9	change_teacher
35	Can delete teacher	9	delete_teacher
36	Can view teacher	9	view_teacher
37	Can add modules	10	add_modules
38	Can change modules	10	change_modules
39	Can delete modules	10	delete_modules
40	Can view modules	10	view_modules
41	Can add module_ topics	11	add_module_topics
42	Can change module_ topics	11	change_module_topics
43	Can delete module_ topics	11	delete_module_topics
44	Can view module_ topics	11	view_module_topics
45	Can add module_ lectures	12	add_module_lectures
46	Can change module_ lectures	12	change_module_lectures
47	Can delete module_ lectures	12	delete_module_lectures
48	Can view module_ lectures	12	view_module_lectures
49	Can add lecture_ details	13	add_lecture_details
50	Can change lecture_ details	13	change_lecture_details
51	Can delete lecture_ details	13	delete_lecture_details
52	Can view lecture_ details	13	view_lecture_details
53	Can add lecture_ exercises	14	add_lecture_exercises
54	Can change lecture_ exercises	14	change_lecture_exercises
55	Can delete lecture_ exercises	14	delete_lecture_exercises
56	Can view lecture_ exercises	14	view_lecture_exercises
57	Can add courses	15	add_courses
58	Can change courses	15	change_courses
59	Can delete courses	15	delete_courses
60	Can view courses	15	view_courses
61	Can add course_contents	16	add_course_contents
62	Can change course_contents	16	change_course_contents
63	Can delete course_contents	16	delete_course_contents
64	Can view course_contents	16	view_course_contents
65	Can add course_ projects	17	add_course_projects
66	Can change course_ projects	17	change_course_projects
67	Can delete course_ projects	17	delete_course_projects
68	Can view course_ projects	17	view_course_projects
69	Can add module_ assesment	18	add_module_assesment

Database: lms, Table: auth_permission, Purpose: Dumping data

id	name	content_type_id	codename
70	Can change module_ assesment	18	change_module_assesment
71	Can delete module_ assesment	18	delete_module_assesment
72	Can view module_ assesment	18	view_module_assesment
73	Can add topic_ assignments	19	add_topic_assignments
74	Can change topic_ assignments	19	change_topic_assignments
75	Can delete topic_ assignments	19	delete_topic_assignments
76	Can view topic_ assignments	19	view_topic_assignments
77	Can add additional_ reading	20	add_additional_reading
78	Can change additional_ reading	20	change_additional_reading
79	Can delete additional_ reading	20	delete_additional_reading
80	Can view additional_ reading	20	view_additional_reading
81	Can add admin_ users	21	add_admin_users
82	Can change admin_ users	21	change_admin_users
83	Can delete admin_ users	21	delete_admin_users
84	Can view admin_ users	21	view_admin_users
85	Can add course batches	22	add_coursebatches
86	Can change course batches	22	change_coursebatches
87	Can delete course batches	22	delete_coursebatches
88	Can view course batches	22	view_coursebatches
89	Can add student courses	23	add_studentcourses
90	Can change student courses	23	change_studentcourses
91	Can delete student courses	23	delete_studentcourses
92	Can view student courses	23	view_studentcourses
93	Can add students	24	add_students
94	Can change students	24	change_students
95	Can delete students	24	delete_students
96	Can view students	24	view_students
97	Can add trainers	25	add_trainers
98	Can change trainers	25	change_trainers
99	Can delete trainers	25	delete_trainers
100	Can view trainers	25	view_trainers
101	Can add batch_ assessments	26	add_batch_assessments
102	Can change batch_ assessments	26	change_batch_assessments
103	Can delete batch_ assessments	26	delete_batch_assessments
104	Can view batch_ assessments	26	view_batch_assessments
105	Can add student_ assesments	27	add_student_assesments
106	Can change student_ assesments	27	change_student_assesments
107	Can delete student_ assesments	27	delete_student_assesments
108	Can view student_ assesments	27	view_student_assesments

Database: lms, Table: auth_user, Purpose: Dumping data

id	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active	date_joined
1	pbkdf2_sha256$390000$mix1ZL7KhzTAqxcmIZz6fb$5tBdwgodONg0QZyV+LhLdxrIXb4WAMpq53GNcsYNDqQ=	2022-11-28 09:55:18.220814	0	akash103	Akashdeep Singh		akash@example.com	0	1	2022-11-10 10:52:33.063865
9	pbkdf2_sha256$390000$Q6NWaZDRCr9RInK4wl47OZ$YMwq01gXpjVHzg7cSb9yHyWnSA8o8A0SZDzs0FwAB4w=	2022-11-21 10:39:01.530368	0	ashvid	Ashvid Kumar		ashvid@example.com	0	1	2022-11-21 10:24:08.261605
10	pbkdf2_sha256$390000$hHrlxj302YNIX3FyeZrdnA$0Sp0YeOxJlg69nA48TWeRbTXzWZQM8B8ImZ/322p9No=	2022-11-28 08:45:02.971020	0	ishleen	Ishleen Kaur		ishleen@example.com	0	1	2022-11-21 12:27:34.584760
11	pbkdf2_sha256$390000$QBZpwelGjmqwGKj0vtIXvJ$6qToTc6pkltphDlV2z8kYNgqfuLSr2Vyih8DvmVbYaY=	2022-11-28 08:40:25.619127	0	kabir117	Kabir Behal		kabir.behal7830@gmail.com	0	1	2022-11-23 12:19:44.512761
8	pbkdf2_sha256$390000$308ycKxjRVDz2yd71rkV2f$eI4SXJ6OnuiPJTRrBx9GKUsO9Q8mP34ojQVGiNFuCVY=	2022-12-15 05:40:02.219325	0	kabir03	Kabir		kabir.behal7830@gmail.com	1	1	2022-11-21 10:07:03.545467
12	pbkdf2_sha256$390000$5ttTBRZpn0yWg20TYP0z33$	2022-12-15 10:27:32.808202	0	nspl	nspl		nsplrtc@gmail.com	1	1	2022-11-28 08:48:28.429733

Database: lms, Table: auth_user, Purpose: Dumping data

id	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active	date_joined
	8s4sio8rTqGyvOFonmKkzAN1g7yqDQJN7UZXO0a/viQ=									
17	pbkdf2_sha256$390000$B2MSsmQCpZlxj0uNVwQ6nJ$u4VCgAiL4LKI8E4lDNnEkvDk2fk2pqh4hJbo+fUuMeE=	2022-12-15 10:23:23.328100	0	sunali	Sunali Kaur		sunali@example.com	1	1	2022-12-15 05:41:15.000000
18	pbkdf2_sha256$390000$wscwCtwQcmwYDuQRWis4je$sJZqRLOc3UWwSVRAdmGnuuRmY5q8M662e4Sr8aKCYkQ=		0	surbhi	Surbhi		surbhi@example.com	0	1	2022-12-15 10:26:54.542501

Database: lms, Table: auth_user_groups, Purpose: Dumping data

id	user_id	group_id
5	12	1
4	8	1

Database: lms, Table: auth_user_user_permissions, Purpose: Dumping data

id	user_id	permission_id
162	17	80
161	17	79
160	17	78
159	17	77
158	17	76
157	17	75
156	17	74
155	17	73
154	17	72
153	17	71
152	17	70
151	17	69
150	17	68
149	17	67
148	17	66
147	17	65
146	17	64
145	17	63
144	17	62
143	17	61
142	17	56
141	17	55
140	17	54
139	17	53
138	17	52
137	17	51
136	17	50
135	17	49
134	17	48
133	17	47
132	17	46
131	17	45
130	17	44
129	17	43
128	17	42
127	17	41
126	17	40
125	17	39
124	17	38
123	17	37
122	17	4

Database: lms, Table: batch_assesments, Purpose: Dumping data

id	assesment_id	batch_id
4	2	BT01

Database: lms, Table: courses, Purpose: Dumping data

id	course_name
8	Graphics
9	Web Designer

Database: lms, Table: course_batches, Purpose: Dumping data

batch_code	starting_date	ending_date	mode	timing_from	timing_to	total_students	language	course_id_id	trainer_id_id
BT01	2023-01-01	2023-02-01	offline	13	15	0	hindi	9	7
BT02	2023-03-06	2023-06-07	offline	16	18	0	hindi	9	3
BT03	2023-03-06	2023-06-07	offline	16	18	0	hindi	8	3

Database: lms, Table: course_contents, Purpose: Dumping data

id	content_type	reference_id	content_name	module_id
12	module	9	Javascript	JS12
13	project	9	E-Commerce Store	3
18	module	9	Python	PY112

Database: lms, Table: course_projects, Purpose: Dumping data

id	project_title	project_description	project_duration
1	PY Calculator	Make a calculator using python	1
3	E-Commerce Store Project	Loop Conditional statements Arrays Objects Classes Functions Clothing E-Commerce Store Data Analysis Classes Customers Customerid Name Phone number (mandatory) Email (optional) Country Constructor Products Productid Name Category [Kids, Men , Women] Regular Price 5000 Discount 500 Sale price 4500 Constructor Orders Ordered Ordernumber Orderdate Customer.Customerid (customer object) Product.Productid (product object) Quanity Price Amount Tax Netamount Constructor Analysis 1.Classes define 2.Relationship 3.Data supply – 10 customers, 50 orders 5 products Details Customer – orders Product – orders Analysis Country wise sales Product wise sales Customer comparison	1

Database: lms, Table: course_projects, Purpose: Dumping data

id	project_title	project_description	project_duration
		All results should be displayed in console	

Database: lms, Table: django_admin_log, Purpose: Dumping data

id	action_time	object_id	object_repr	action_flag	change_message	content_type_id	user_id
17	2022-12-15 05:40:27.780819	16	sunali	3		4	8
18	2022-12-15 05:44:03.637544	17	sunali	2	[{"changed": {"fields": ["Staff status", "User permissions"]}}]	4	12
16	2022-12-15 05:37:50.378847	16	sunali	2	[{"changed": {"fields": ["User permissions"]}}]	4	12
15	2022-12-15 05:34:14.698918	15	sunali	3		4	12
13	2022-12-15 05:32:26.149816	13	sunali	3		4	12
14	2022-12-15 05:33:11.110474	14	sunali	3		4	12
12	2022-11-21 10:08:34.444551	2	kabir._b	3		4	8
11	2022-11-21 10:08:24.232544	7	kabir12	3		4	8
10	2022-11-21 10:08:08.563156	3	nspl	3		4	8

Database: lms, Table: django_content_type, Purpose: Dumping data

id	app_label	model
1	admin	logentry
2	auth	permission
3	auth	group
4	auth	user
5	contenttypes	contenttype
6	sessions	session
7	Assignments	assignments
8	Assignments	student
9	Assignments	teacher
10	admin_app	modules
11	admin_app	module_topics
12	admin_app	module_lectures
13	admin_app	lecture_details
14	admin_app	lecture_exercises
15	admin_app	courses
16	admin_app	course_contents
17	admin_app	course_projects
18	admin_app	module_assesment
19	admin_app	topic_assignments
20	admin_app	additional_reading
21	admin_app	admin_users
22	admin_app	coursebatches
23	admin_app	studentcourses
24	student	students
25	trainer	trainers
26	admin_app	batch_assessments
27	admin_app	student_assesments

Database: lms, Table: django_migrations, Purpose: Dumping data

id	app	name	applied
1	Assignments	0001_initial	2022-11-10 10:43:43.085949
2	contenttypes	0001_initial	2022-11-10 10:43:43.398167
3	auth	0001_initial	2022-11-10 10:43:45.453712
4	admin	0001_initial	2022-11-10 10:43:45.902633
5	admin	0002_logentry_remove_auto_add	2022-11-10 10:43:45.918623
6	admin	0003_logentry_add_action_flag_choices	2022-11-10 10:43:45.934616
7	trainer	0001_initial	2022-11-10 10:43:46.003570
8	student	0001_initial	2022-11-10 10:43:46.090533
9	student	0002_students_delete_student_signup	2022-11-10 10:43:46.193203
10	admin_app	0001_initial	2022-11-10 10:43:47.000832
11	admin_app	0002_alter_module_lectures_module_topics_name_and_more	2022-11-10 10:43:49.212293
12	admin_app	0003_alter_module_topics_module_id	2022-11-10 10:43:50.043968
13	admin_app	0004_lecture_details	2022-11-10 10:43:50.095649
14	admin_app	0005_lecture_details_module_lecture_name_and_more	2022-11-10 10:43:50.459758
15	admin_app	0006_lecture_exercises	2022-11-10 10:43:50.709236
16	admin_app	0007_alter_lecture_exercises_lecture	2022-11-10 10:43:51.035547
17	admin_app	0008_module_topics_assignments_module_and_more	2022-11-10 10:43:51.416035
18	admin_app	0009_alter_module_lectures_module_lecture_name	2022-11-10 10:43:51.833632
19	admin_app	0010_lecture_exercises_topic_name	2022-11-10 10:43:51.993177
20	admin_app	0011_remove_lecture_exercises_topic_name	2022-11-10 10:43:52.069123
21	admin_app	0012_module_lectures_module_name	2022-11-10 10:43:52.300079
22	admin_app	0013_alter_module_lectures_module_topics_name	2022-11-10 10:43:52.824358
23	admin_app	0014_alter_module_topics_module_id	2022-11-10 10:43:53.087336
24	admin_app	0015_courses	2022-11-10 10:43:53.176914
25	admin_app	0016_coursecontents	2022-11-10 10:43:53.475575
26	admin_app	0017_rename_contenttype_coursecontents_content_type_and_more	2022-11-10 10:43:53.981471
27	admin_app	0018_rename_coursename_courses_course_name	2022-11-10 10:43:54.039305
28	admin_app	0019_rename_coursecontents_course_contents	2022-11-10 10:43:54.100308
29	admin_app	0020_course_contents_content_name	2022-11-10 10:43:54.247538
30	admin_app	0021_rename_reference_id_course_contents_reference_and_more	2022-11-10 10:43:54.673665
31	admin_app	0022_remove_course_contents_order	2022-11-10 10:43:54.742141
32	admin_app	0023_modules_course	2022-11-10 10:43:55.044495
33	admin_app	0024_course_contents_module	2022-11-10 10:43:55.311078
34	admin_app	0025_delete_module_projects	2022-11-10 10:43:55.321070
35	admin_app	0026_course_projects	2022-11-10 10:43:55.521089
36	admin_app	0027_alter_course_projects_project_description	2022-11-10 10:43:55.617666
37	admin_app	0028_remove_modules_course	2022-11-10 10:43:56.068254
38	admin_app	0029_alter_course_contents_module	2022-11-10 10:43:56.980731
39	admin_app	0030_remove_course_projects_course	2022-11-10 10:43:57.444288
40	admin_app	0031_module_lectures_module_lecture_duration_and_more	2022-11-10 10:43:57.894025
41	admin_app	0032_course_contents_module_duration	2022-11-10 10:43:58.093126
42	admin_app	0033_course_projects_project_duration	2022-11-10 10:43:58.230150
43	admin_app	0034_rename_module_topics_assignments_module_assesment_and_more	2022-11-10 10:43:58.454992
44	admin_app	0035_topic_assignments	2022-11-10 10:43:58.746078
45	admin_app	0036_additional_reading	2022-11-10 10:43:59.032753
46	admin_app	0037_additional_reading_content	2022-11-10 10:43:59.118154
47	admin_app	0038_alter_additional_reading_content	2022-11-10 10:43:59.200772
48	admin_app	0039_rename_reference_additional_reading_lecture_reference_and_more	2022-11-10 10:43:59.594410
49	admin_app	0040_alter_additional_reading_lecture_reference	2022-11-10 10:43:59.941233
50	admin_app	0041_alter_additional_reading_content_and_more	2022-11-10 10:43:59.966215
51	admin_app	0042_alter_additional_reading_content_and_more	2022-11-10 10:43:59.990202
52	admin_app	0043_alter_additional_reading_lecture_reference	2022-11-10 10:44:00.309418
53	admin_app	0044_rename_lecture_reference_additional_reading_topic_reference	2022-11-10 10:44:00.638877
54	admin_app	0045_lecture_details_file_type_and_more	2022-11-10 10:44:00.974908
55	admin_app	0046_rename_lecture_file_lecture_details_lecture_doc_and_more	2022-11-10 10:44:01.238146
56	admin_app	0047_alter_lecture_details_lecture_video	2022-11-10 10:44:01.259136
57	admin_app	0048_alter_lecture_details_lecture_doc	2022-11-10 10:44:01.277123
58	admin_app	0049_alter_lecture_details_lecture_doc_and_more	2022-11-10 10:44:01.299109
59	admin_app	0050_alter_lecture_details_lecture_doc_and_more	2022-11-10 10:44:01.325093
60	admin_app	0051_alter_lecture_details_lecture_doc_and_more	2022-11-10 10:44:01.349078
61	admin_app	0052_alter_lecture_details_lecture_doc_and_more	2022-11-10 10:44:01.371065
62	admin_app	0053_alter_lecture_details_lecture_doc	2022-11-10 10:44:01.383056
63	admin_app	0054_alter_lecture_details_lecture_content	2022-11-10 10:44:01.399047
64	admin_app	0055_admin_users	2022-11-10 10:44:01.469004
65	admin_app	0056_admin_users_email	2022-11-10 10:44:01.769822
66	admin_app	0057_coursebatches_students_trainers_studentcourses_and_more	2022-11-10 10:44:03.018190
67	admin_app	0058_alter_students_profile_pic	2022-11-10 10:44:03.035179
68	admin_app	0059_alter_studentcourses_student_id_delete_students	2022-11-10 10:44:03.315238
69	admin_app	0060_alter_coursebatches_trainer_id_delete_trainers	2022-11-10 10:44:03.637836

Database: lms, Table: django_migrations, Purpose: Dumping data

id	app	name	applied
70	contenttypes	0002_remove_content_type_name	2022-11-10 10:44:03.843143
71	auth	0002_alter_permission_name_max_length	2022-11-10 10:44:03.931805
72	auth	0003_alter_user_email_max_length	2022-11-10 10:44:04.007758
73	auth	0004_alter_user_username_opts	2022-11-10 10:44:04.025748
74	auth	0005_alter_user_last_login_null	2022-11-10 10:44:04.200222
75	auth	0006_require_contenttypes_0002	2022-11-10 10:44:04.206217
76	auth	0007_alter_validators_add_error_messages	2022-11-10 10:44:04.223206
77	auth	0008_alter_user_username_max_length	2022-11-10 10:44:04.302159
78	auth	0009_alter_user_last_name_max_length	2022-11-10 10:44:04.395540
79	auth	0010_alter_group_name_max_length	2022-11-10 10:44:04.498963
80	auth	0011_update_proxy_permissions	2022-11-10 10:44:04.531943
81	auth	0012_alter_user_first_name_max_length	2022-11-10 10:44:04.623991
82	sessions	0001_initial	2022-11-10 10:44:04.810751
83	student	0003_alter_students_username	2022-11-10 10:44:04.904937
84	student	0004_alter_students_profile_pic	2022-11-10 10:44:04.917075
85	student	0005_alter_students_profile_pic	2022-11-10 10:44:05.001022
86	student	0006_alter_students_phone_number	2022-11-10 10:44:05.092884
87	trainer	0002_trainers_profile_pic	2022-11-10 10:44:05.381707
88	trainer	0003_alter_trainers_phone_number_and_more	2022-11-10 10:44:05.621329
89	admin_app	0061_alter_coursebatches_timing_from_and_more	2022-11-17 11:52:50.211748
90	admin_app	0062_admin_users_profile	2022-11-17 11:54:44.605543
91	admin_app	0063_remove_admin_users_profile	2022-11-21 10:34:39.247496
92	admin_app	0064_batch_assessments_student_assesments	2022-11-23 11:54:40.800497
93	admin_app	0065_rename_assignment_batch_assessments_assesment_and_more	2022-11-23 12:02:41.582002
94	admin_app	0066_rename_assesment_batch_assessments_assesment_and_more	2022-11-23 12:10:32.141070
95	trainer	0004_alter_trainers_profile_pic	2022-12-15 04:30:16.710998
96	admin_app	0067_alter_modules_table	2022-12-15 06:05:44.103550
97	admin_app	0068_alter_additional_reading_table_and_more	2022-12-15 06:20:07.643021
98	admin_app	0069_alter_admin_users_table_alter_module_assesment_table_and_more	2022-12-15 06:22:35.921988
99	student	0007_alter_students_table	2022-12-15 06:24:05.265167
100	trainer	0005_alter_trainers_table	2022-12-15 06:24:05.279158
101	admin_app	0070_alter_course_contents_module_and_more	2022-12-15 10:44:41.718795
102	admin_app	0071_alter_course_contents_module_and_more	2022-12-15 10:59:44.225530
103	admin_app	0072_remove_course_contents_module_duration	2022-12-15 11:09:49.335138
104	admin_app	0073_remove_module_lectures_module_name_and_more	2022-12-16 04:49:33.323841
105	admin_app	0074_rename_module_lecture_name_lecture_details_lecture_reference	2022-12-16 05:39:57.639651
106	admin_app	0075_alter_lecture_details_lecture_reference	2022-12-16 05:40:19.424443
107	admin_app	0076_remove_coursebatches_id_and_more	2022-12-16 06:36:04.730721
108	admin_app	0077_alter_coursebatches_batch_code	2022-12-16 07:08:23.996712
109	admin_app	0078_remove_studentcourses_batch_id_and_more	2022-12-16 07:13:17.982876
110	admin_app	0079_studentcourses_batch_id	2022-12-16 07:13:59.027463

Database: lms, Table: django_session, Purpose: Dumping data

session_key	session_data	expire_date
rxzjtah3v5vr20x5ue4zgqtchort2gsy	e30:1ozZgf:pEeU9rBA7bhLjdR5NxYFhVPa4plsqLSsK4HFHWLJ-Nw	2022-12-12 08:40:25.525992
o4odixh8u62572w08zq57d64ki3rno54	e30:1ozZl8:eYAMXXiQ3HltzSrp7yt8ue56n3-CuBgRnx6FXkpTMr4	2022-12-12 08:45:02.969021
ou3og7xgca9s0d1spir69r4oc3dcqjg9	.eJxVjMsOwiAQRf-FtSFIh5dL9_0GMjCjVA0kpV0Z_92QdKHbe865bxFx30rcO69xIXERXpx-t4T5yXUAemC9N5lb3dYlyaHIg3Y5N-LX9XD_Dgr2MmoADtZAsEolN5l8BhcmJNATWPKO2ARInjHbm1JZoXMA2oNmYm85i88Xwew3Ww:1ox4Sq:Kfyun8DLaXtvl2HKdSBOlcmelCEi-t7iyLPl0Uychwc	2022-12-05 10:55:48.741031
x2fm7d0tkngi0st5vs3tqihmvd8qd9jv	.eJxVjE0OwiAUBu_C2hB-hIJL956BfPAetmrapLQr490NSRe6nZnMWyTs25j2xmuaSFyENuL0CzPKk-du6IH5vsiyzNs6ZdkTedgmbwvx63q0f4MRbexfT1VFFQDt3WCdskwAKRuowHmXfUXkcK45KzYFIVfYIWJgo7UJUXy-FGU4iQ:1p4hBE:y5iNzxPdT0QdwDhp22QvL9csruSDjhL0nZ29m5IZjCY	2022-12-26 11:41:08.426455
gqai8ttwicjq32bosuoazyl704can00l	.eJxVjE0OwiAUBu_C2hB-hIJL956BfPAetmrapLQr490NSRe6nZnMWyTs25j2xmuaSFyENuL0CzPKk-du6IH5vsiyzNs6ZdkTedgmbwvx63q0f4MRbexfT1VFFQDt3WCdskwAKRuowHmXfUXkcK45KzYFIVfYIWJgo7UJUXy-FGU4iQ:1ozcqL:TDFi7NjakjxtcqQKqMagSQYfD9Dvkctw00Mw8foz9p4	2022-12-12 12:02:37.667793
zujbsh8spqlibfrqahyw4afdty1pwwhz	.eJxVjE0OwiAUBu_C2hB-hIJL956BfPAetmrapLQr490NSRe6nZnMWyTs25j2xmuaSFyENuL0CzPKk-du6IH5vsiyzNs6ZdkTedgmbwvx63q0f4MRbexfT1VFFQDt3WCdskwAKRuowHmXfUXkcK45KzYFIVfYIWJgo7UJUXy-FGU4iQ:1p5ISe:YJDEHpXeZs561EkbeUGOBq8jBSJ2oNYnzmn5jnOdbaI	2022-12-29 10:27:32.810202

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
1	Lecture-1 Introduction	Lecture 1	1		
3	Lecture-1 (Dictionaries	Lecture 2	1		
4	Javascript Lecture-1 (Introduction)	Javascript Client Side Language Javascript is a scripting language Javascript was designed to add interactivity to HTML pages Javascript is lightweight language because it is executed by browsers Javascript is open license Two methods to write javascript 1.Internal javascript <!DOCTYPE html> <html lang="en"> <head> <meta charset="utf-8" /> <title></title> <script> document.write("hello"); </script> </head> <body> </body> </html> 2.External javavascript Script.js document.write("hello");	5		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		index.html
		<!DOCTYPE
		html>
		
		<html
		lang="en">
		  <head>
		  <meta
		charset="utf-8
		" />
		
		<title></title>
		
		  <script type
		="text/javascri
		pt" src="Script.
		js"></script>
		
		  </head>
		  <body>
		
		  </body>
		</html>
		
		<!DOCTYPE
		html>
		
		<html
		lang="en">
		  <head>
		  <meta
		charset="utf-8
		" />
		
		<title></title>
		
		  <script>
		
		  var x =
		10;
		  var y =
		20;
		
		  var z = x
		+ y;
		
		  document
		.write(z);
		
		  </script>
		
		  </head>
		  <body>			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		</body> </html>			
5	Lecture-1 variables	Variables Most of the time, a JavaScript application needs to work with information. Here are two examples: An online shop – the information might include goods being sold and a shopping cart. A chat application – the information might include users, messages, and much more. Variables are used to store this information. Variable A variable is a “named storage” for data. Variable Naming Rules Variable names are case sensitive. AGE, Age and age are different variables Variable names can have letters, numbers, underscore and $ Variable names must begin with letter or _ Different Types of Variables We can declare variables to store data by using the var, let, or const keywords. let – is a modern variable declaration. var – is an old	6		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		type variable declaration. const – is like let, but the value of the variable can’t be changed.
		
		let
		To create a variable in JavaScript, use the let keyword.
		The statement below creates (in other words: declares) a variable with the name “message”:
		
		let message;
		Now, we can put some data into it by using the assignment operator =:
		
		let message;
		message = 'Hello'; // store the string 'Hello' in the variable named message
		The string is now saved into the memory area associated with the variable.
		We can access it using the variable name:
		
		let message;
		message = 'Hello!';
		alert(message); // shows the variable content
		
		To be concise, we can combine the variable declaration and assignment into a single line:
		let message = 'Hello!'; // define the variable and assign the value			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		alert(message); // Hello!
		
		We can also declare multiple variables in one line:
		let user = 'John', age = 25, message = 'Hello';
		
		That might seem shorter, but we don’t recommend it. For the sake of better readability, please use a single line per variable.
		
		The multiline variant is a bit longer, but easier to read:
		let user = 'John';
		let age = 25;
		let message = 'Hello';
		
		Var
		
		The var declaration is similar to let. Most of the time we can replace let by var or vice-versa and expect things to work:
		var message = "Hi";
		alert(message); // Hi
		But internally var is a very different beast that originates from very old times. It’s generally not used in modern scripts, but still lurks in the old ones.
		it’s important to understand differences			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		when migrating old scripts from var to let, to avoid odd errors.
		
		“var” has no block scope
		Variables, declared with var, are either function-scoped or global-scoped. They are visible through blocks.
		For instance:
		if (true) {
		  var test = true; // use "var" instead of "let"
		}
		
		alert(test); // true, the variable lives after if
		As var ignores code blocks, we’ve got a global variable test.
		
		If we used let test instead of var test, then the variable would only be visible inside if:
		if (true) {
		  let test = true; // use "let"
		}
		
		alert(test); // ReferenceError: test is not defined
		
		The same thing for loops: var cannot be block- or loop-local:
		for (var i = 0; i < 10; i++) {
		  var one = 1;
		  // ...
		}
		
		alert(i); // 10, "i" is visible			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		after loop, it's a global variable
		alert(one); // 1, "one" is visible
		after loop, it's a global variable
		
		If a code block is inside a function, then var becomes a function-level variable:
		function sayHi() {
		  if (true) {
		  var phrase = "Hello";
		  }
		
		  alert(phrase); // works
		}
		
		sayHi();
		alert(phrase); // ReferenceError: phrase is not defined
		
		“var” tolerates redeclarations
		
		If we declare the same variable with let twice in the same scope, that’s an error:
		let user;
		let user; // SyntaxError: 'user' has already been declared
		
		With var, we can redeclare a variable any number of times. If we use var with an already-declared variable, it’s just ignored:
		var user = "Pete";
		
		var user = "John"; // this "var" does nothing (already declared)			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		// ...it doesn't trigger an error
		
		alert(user); // John
		
		“var” variables can be declared below their use
		var declarations are processed when the function starts (or script starts for globals).
		
		In other words, var variables are defined from the beginning of the function, no matter where the definition is (assuming that the definition is not in the nested function).
		So this code:
		function sayHi() {
		  phrase = "Hello";
		
		  alert(phrase);
		
		  var phrase;
		}
		sayHi();
		...Is technically the same as this (moved var phrase above):
		function sayHi() {
		  var phrase;
		
		  phrase = "Hello";
		
		  alert(phrase);
		}
		sayHi();
		...Or even as this (remember, code blocks are ignored):
		
		function sayHi() {
		  phrase = "Hello"; // (*)			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		if (false) {
		  var phrase;
		}
		
		alert(phrase);
		}
		
		sayHi();
		People also call such behavior “hoisting” (raising), because all var are “hoisted” (raised) to the top of the function.
		So in the example above, if (false) branch never executes, but that doesn’t matter. The var inside it is processed in the beginning of the function, so at the moment of (*) the variable exists.
		Declarations are hoisted, but assignments are not.
		That’s best demonstrated with an example:
		function sayHi() {
		  alert(phrase);
		
		  var phrase = "Hello";
		}
		
		sayHi();
		The line var phrase = "Hello" has two actions in it:
		Variable declaration var
		Variable assignment =.
		The declaration is processed at the start of function execution (“hoisted”), but the assignment always works at the place where it appears. So			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		the code works essentially like this:
		function sayHi() {
		  var phrase; // declaration works at the start...
		
		  alert(phrase); // undefined
		
		  phrase = "Hello"; // ...assignment - when the execution reaches it.
		}
		
		sayHi();
		Because all var declarations are processed at the function start, we can reference them at any place. But variables are undefined until the assignments.
		In both examples above, alert runs without an error, because the variable phrase exists. But its value is not yet assigned, so it shows undefined.
		
		const
		once value has been initialized, we cannot change it
		const rate=5;
		rate=6;
		it will give error			

6	Lecture-1 DataTypes		7	doc/Javascript_Lecture_3_-_Data_Types_VWRUcjh.doc	
7	Lecture-1 Type Conversions	Type Conversions Most of the time, operators and functions automatically convert the values given to them to the right type. There are also cases when we need to	8		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		explicitly convert a value to the expected type.
		
		String Conversion String conversion happens when we need the string form of a value.
		let value = true;
		alert(typeof value); // boolean
		value = String(value); // now value is a string "true"
		alert(typeof value); // string
		String conversion is mostly obvious. A false becomes "false", null becomes "null", etc.
		
		Numeric Conversion Numeric conversion happens in mathematical functions and expressions automatically.
		For example, when division / is applied to non-numbers
		alert("6" / "2"); // 3, strings are converted to numbers
		We can use the Number(value) function to explicitly convert a value to a number:
		let str = "123";
		alert(typeof str); // string
		
		let num = Number(str); // becomes a number 123
		alert(typeof num); // number
		
		If the string is			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		not a valid number, the result of such a conversion is NaN. For instance:
		let age = Number("india");
		alert(age); // NaN, conversion failed
		
		Numeric conversion rules:
		Value Becomes...
		undefined□NaN
		null□0
		true and false 1 and 0
		string Whitespaces (includes spaces, tabs \t, newlines \n etc.) from the start and end are removed. If the remaining string is empty, the result is 0. Otherwise, the number is “read” from the string. An error gives NaN.
		Examples:
		alert(Number(" 123 ")); // 123
		alert(Number("123z")); // NaN (error reading a number at "z")
		alert(Number(true)); // 1
		alert(Number(false)); // 0
		alert(Number(null)); // 0
		let d;
		alert(Number(d)); // NaN
		
		Please note that null and undefined behave differently			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		here: null becomes zero while undefined becomes NaN. Boolean Conversion Boolean conversion rule: Values that are intuitively “empty”, like 0, an empty string, null, undefined, and NaN, become false. Other values become true. For instance: alert(Boolean(1)); // true alert(Boolean(0)); // false
		
		alert(Boolean("hello")); // true alert(Boolean("")); // false Please note: the string with zero "0" is true Some languages (namely PHP) treat "0" as false. But in JavaScript, a non-empty string is always true. alert(Boolean("0")); // true alert(Boolean(" ")); // spaces, also true (any non-empty string is true)			

8	Lecture-1 operators	Basic operators, maths We know many operators from school. They are things like addition +, multiplication *, subtraction -, and so on. Terms: “unary”,	9		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		“binary”, “operand” Before we move on, let’s grasp some common terminology. An operand – is what operators are applied to. For instance, in the multiplication of 5 * 2 there are two operands: the left operand is 5 and the right operand is 2. Sometimes, people call these “arguments” instead of “operands”. An operator is unary if it has a single operand. For example, the unary negation - reverses the sign of a number: let x = 1; x = -x; alert(x); // -1, unary negation was applied
		
		An operator is binary if it has two operands. The same minus exists in binary form as well: let x = 1, y = 3; alert(y - x); // 2, binary minus subtracts values
		
		Maths The following math operations are supported: Addition +, Subtraction -, Multiplication *, Division /, Remainder %, Exponentiation			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		**.
		
		Remainder %
		The remainder operator %, despite its appearance, is not related to percents.
		The result of a % b is the remainder of the integer division of a by b.
		For instance:
		alert(5 % 2); // 1, a remainder of 5 divided by 2
		alert(8 % 3); // 2, a remainder of 8 divided by 3
		
		Exponentiation **
		The exponentiation operator a ** b raises a to the power of b.
		In school maths, we write that as ab.
		For instance:
		alert(2 ** 2); // 2² = 4
		alert(2 ** 3); // 2³ = 8
		
		String concatenation with binary +
		If the binary + is applied to strings, it merges (concatenates) them:
		
		let s = "my" + "string";
		alert(s); // mystring
		Note that if any of the operands is a string, then the other one is converted to a string too.
		
		For example:
		alert('1' + 2); // "12"
		alert(2 + '1'); // "21"			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		See, it doesn’t matter whether the first operand is a string or the second one.
		
		Here’s a more complex example: alert(2 + 2 + '1'); // "41" and not "221"
		
		Here, operators work one after another. The first + sums two numbers, so it returns 4, then the next + adds the string 1 to it, so it’s like 4 + '1' = '41'. alert('1' + 2 + 2); // "122" and not "14"
		
		Here, the first operand is a string, the compiler treats the other two operands as strings too. The 2 gets concatenated to '1', so it’s like '1' + 2 = "12" and "12" + 2 = "122".
		
		The binary + is the only operator that supports strings in such a way. Other arithmetic operators work only with numbers and always convert their operands to numbers.
		
		Here’s the demo for subtraction and division: alert(6 - '2'); // 4, converts '2' to a number alert('6' / '2'); // 3, converts both operands to numbers			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Operator precedence
		Precedence Name□Sign
		...□...□...
		14□unary plus +
		14□unary negation□-
		13 exponentiation **
		12 multiplication *
		12□division□/
		11□addition□+
		11□subtraction -
		...□...□...
		2□assignment =
		...□...□...
		
		Assignment
		Let’s note that an assignment = is also an operator. It is listed in the precedence table with the very low priority of 2. That’s why, when we assign a variable, like x = 2 * 2 + 1, the calculations are done first and then the = is evaluated, storing the result in x.
		let x = 2 * 2 + 1;
		
		alert(x); // 5
		
		Chaining assignments
		
		Another interesting feature is the ability to chain assignments:
		let a, b, c;
		
		a = b = c = 2 + 2;			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		alert(a); // 4
		alert(b); // 4
		alert(c); // 4
		
		Chained assignments evaluate from right to left. First, the rightmost expression 2 + 2 is evaluated and then assigned to the variables on the left: c, b and a. At the end, all the variables share a single value.
		Once again, for the purposes of readability it’s better to split such code into few lines:
		c = 2 + 2;
		b = c;
		a = c;
		That’s easier to read, especially when eye-scanning the code fast.
		
		Modify-in-place
		We often need to apply an operator to a variable and store the new result in that same variable.
		For example:
		let n = 2;
		n = n + 5;
		alert(n); //7
		
		This notation can be shortened using the operators += and *=:
		let n = 2;
		n += 5; // now n = 7 (same as n = n + 5)
		n *= 2; // now n = 14 (same as n = n * 2)
		
		alert(n); // 14
		Short “modify-and-assign” operators exist for all			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		arithmetical operators
		
		Such operators have the same precedence as a normal assignment, so they run after most other calculations:
		
		let n = 2;
		n *= 3 + 5; // right part evaluated first, same as n *= 8
		
		alert(n); // 16
		
		Increment/decrement
		Increasing or decreasing a number by one is among the most common numerical operations.
		So, there are special operators for it:
		Increment ++ increases a variable by 1:
		
		let counter = 2;
		counter++;
		 // works the same as counter = counter + 1, but is shorter
		alert(counter); // 3
		
		Decrement -- decreases a variable by 1:
		let counter = 2;
		counter--;
		// works the same as counter = counter - 1, but is shorter
		alert(counter); // 1
		
		Important:
		Increment/decrement can only be applied to variables.
		Trying to use it on a value like 5++ will give			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		an error.
		
		The operators ++ and -- can be placed either before or after a variable.
		When the operator goes after the variable, it is in “postfix form”: counter++.
		
		The “prefix form” is when the operator goes before the variable: ++counter.
		
		Both of these statements do the same thing: increase counter by 1.
		Is there any difference?
		Yes, but we can only see it if we use the returned value of ++/--.
		
		Let’s clarify. As we know, all operators return a value. Increment/decrement is no exception. The prefix form returns the new value while the postfix form returns the old value (prior to increment/decrement).
		To see the difference, here’s an example:
		let counter = 1;
		let a = ++counter;
		
		alert(a); // 2
		
		The prefix form ++counter increments counter and returns the new value, 2.			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		So, the alert shows 2.
		Now, let’s use the postfix form:
		
		let counter = 1;
		let a = counter++; // (*) changed ++counter to counter++
		
		alert(a); // 1
		In the line (*), the postfix form counter++ also increments counter but returns the old value (prior to increment). So, the alert shows 1.
		
		To summarize:
		If the result of increment/decrement is not used, there is no difference in which form to use:
		let counter = 0;
		counter++;
		++counter;
		alert(counter); // 2, the lines above did the same
		
		Increment/decrement among other operators
		
		The operators ++/-- can be used inside expressions as well. Their precedence is higher than most other arithmetical operations.
		For instance:
		let counter = 1;
		alert(2 * ++counter); // 4
		
		Compare with:
		let counter =			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		1;
		alert(2 *
		counter++); //
		2, because
		counter++
		returns the
		"old" value
		
		Though
		technically
		okay, such
		notation
		usually makes
		code less
		readable. One
		line does
		multiple things
		– not good.
		
		While reading
		code, a fast
		“vertical” eye-
		scan can easily
		miss
		something like
		counter++ and
		it won’t be
		obvious that
		the variable
		increased.
		
		We advise a
		style of “one
		line – one
		action”:
		let counter =
		1;
		alert(2 *
		counter);
		counter++;			

9	Lecture-1 Dialogue Boxex	Dialog boxes
		
		Alert
		<!DOCTYPE
		html>
		
		<html
		lang="en">
		  <head>
		  <meta
		charset="utf-8
		" />
		
		<title></title>
		
		  <script>
		
		  var age =
		30;
		
		  if (age >=
		18)
		  {
		
		alert("you can
		vote");
		
		  }
			10		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		else
		{
		document.write("you cannot vote");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		
		Prompt
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		var age = prompt("enter your age");
		
		if (age >= 18)
		{
		
		alert("you can vote");
		
		}
		
		else
		{
		
		alert("you cannot vote");
		
		}
		
		</script>			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		</head>
		<body>
		
		</body>
		</html>
		
		Confirm
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		var r = confirm("Are you sure to cancel");
		
		if (r==true) {
		
		alert("you press ok");
		
		}
		
		else {
		
		alert("you press cancel");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>			

10	Lecture-1 Comparisions	Basic operators, maths	11		
		We know many			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		operators from school. They are things like addition +, multiplication *, subtraction -, and so on.
		
		Terms: “unary”, “binary”, “operand”
		Before we move on, let’s grasp some common terminology.
		An operand – is what operators are applied to. For instance, in the multiplication of 5 * 2 there are two operands: the left operand is 5 and the right operand is 2. Sometimes, people call these “arguments” instead of “operands”.
		An operator is unary if it has a single operand. For example, the unary negation - reverses the sign of a number:
		let x = 1;
		x = -x;
		alert(x); // -1, unary negation was applied
		
		An operator is binary if it has two operands. The same minus exists in binary form as well:
		let x = 1, y = 3;
		alert(y - x); // 2, binary minus subtracts values			

		Maths			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		The following math operations are supported: Addition +, Subtraction -, Multiplication *, Division /, Remainder %, Exponentiation **.
		
		Remainder % The remainder operator %, despite its appearance, is not related to percents. The result of a % b is the remainder of the integer division of a by b. For instance: alert(5 % 2); // 1, a remainder of 5 divided by 2 alert(8 % 3); // 2, a remainder of 8 divided by 3
		
		Exponentiation ** The exponentiation operator a ** b raises a to the power of b. In school maths, we write that as ab. For instance: alert(2 ** 2); // 2² = 4 alert(2 ** 3); // 2³ = 8
		
		String concatenation with binary + If the binary + is applied to strings, it merges (concatenates) them:
		
		let s = "my" + "string"; alert(s); // mystring Note that if any of the operands is a			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		string, then the other one is converted to a string too.
		
		For example:
		alert('1' + 2);
		// "12"
		alert(2 + '1');
		// "21"
		See, it doesn’t matter whether the first operand is a string or the second one.
		
		Here’s a more complex example:
		alert(2 + 2 + '1'); // "41" and not "221"
		
		Here, operators work one after another. The first + sums two numbers, so it returns 4, then the next + adds the string 1 to it, so it’s like 4 + '1' = '41'.
		alert('1' + 2 + 2); // "122" and not "14"
		
		Here, the first operand is a string, the compiler treats the other two operands as strings too. The 2 gets concatenated to '1', so it’s like '1' + 2 = "12" and "12" + 2 = "122".
		
		The binary + is the only operator that supports strings in such a way. Other arithmetic operators work only with numbers and always convert their operands to numbers.
		
		Here’s the demo for			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		subtraction and division:
		alert(6 - '2'); // 4, converts '2' to a number
		alert('6' / '2'); // 3, converts both operands to numbers
		
		Operator precedence
		Precedence Name□Sign
		...□...□...
		14□unary plus +
		14□unary negation□-
		13 exponentiation **
		12 multiplication *
		12□division□/
		11□addition□+
		11□subtraction -
		...□...□...
		2□assignment =
		...□...□...
		
		Assignment
		Let’s note that an assignment = is also an operator. It is listed in the precedence table with the very low priority of 2. That’s why, when we assign a variable, like x = 2 * 2 + 1, the calculations are done first and then the = is evaluated, storing the result in x.
		let x = 2 * 2 + 1;
		
		alert(x); // 5			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Chaining assignments
		
		Another interesting feature is the ability to chain assignments:
		let a, b, c;
		
		a = b = c = 2 + 2;
		
		alert(a); // 4
		alert(b); // 4
		alert(c); // 4
		
		Chained assignments evaluate from right to left. First, the rightmost expression 2 + 2 is evaluated and then assigned to the variables on the left: c, b and a. At the end, all the variables share a single value.
		Once again, for the purposes of readability it’s better to split such code into few lines:
		c = 2 + 2;
		b = c;
		a = c;
		That’s easier to read, especially when eye-scanning the code fast.
		
		Modify-in-place
		We often need to apply an operator to a variable and store the new result in that same variable.
		For example:
		let n = 2;
		n = n + 5;
		alert(n); //7
		
		This notation can be shortened using the operators += and *=:			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		let n = 2;
		n += 5; // now n = 7 (same as n = n + 5)
		n *= 2; // now n = 14 (same as n = n * 2)
		
		alert(n); // 14
		Short “modify-and-assign” operators exist for all arithmetical operators
		
		Such operators have the same precedence as a normal assignment, so they run after most other calculations:
		
		let n = 2;
		n *= 3 + 5; // right part evaluated first, same as n *= 8
		
		alert(n); // 16
		
		Increment/decrement
		Increasing or decreasing a number by one is among the most common numerical operations.
		So, there are special operators for it:
		Increment ++ increases a variable by 1:
		
		let counter = 2;
		counter++;
		// works the same as counter = counter + 1, but is shorter
		alert(counter); // 3
		
		Decrement -- decreases a variable by 1:
		let counter = 2;
		counter--;
		// works the same as counter =			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		counter - 1, but is shorter alert(counter); // 1 Important: Increment/decrement can only be applied to variables. Trying to use it on a value like 5++ will give an error. The operators ++ and -- can be placed either before or after a variable. When the operator goes after the variable, it is in “postfix form”: counter++. The “prefix form” is when the operator goes before the variable: ++counter. Both of these statements do the same thing: increase counter by 1. Is there any difference? Yes, but we can only see it if we use the returned value of ++/--. Let’s clarify. As we know, all operators return a value. Increment/decrement is no exception. The prefix form returns the new value while the postfix form returns the old value (prior to increment/decrement). To see the difference, here’s an example:			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		let counter = 1;
		let a = ++counter;
		
		alert(a); // 2
		
		The prefix form ++counter increments counter and returns the new value, 2. So, the alert shows 2.
		Now, let’s use the postfix form:
		
		let counter = 1;
		let a = counter++; // (*) changed ++counter to counter++
		
		alert(a); // 1
		In the line (*), the postfix form counter++ also increments counter but returns the old value (prior to increment). So, the alert shows 1.
		
		To summarize:
		If the result of increment/decrement is not used, there is no difference in which form to use:
		let counter = 0;
		counter++;
		++counter;
		alert(counter); // 2, the lines above did the same
		
		Increment/decrement among other operators
		
		The operators ++/-- can be used inside expressions as well. Their precedence is			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		higher than most other arithmetical operations. For instance:
		let counter = 1;
		alert(2 * ++counter); // 4
		
		Compare with:
		let counter = 1;
		alert(2 * counter++); // 2, because counter++ returns the "old" value
		
		Though technically okay, such notation usually makes code less readable. One line does multiple things – not good.
		
		While reading code, a fast “vertical” eye-scan can easily miss something like counter++ and it won’t be obvious that the variable increased.
		
		We advise a style of “one line – one action”:
		let counter = 1;
		alert(2 * counter);
		counter++;			

11	Lecture-1 Decision Making Statements	Conditional Statements Conditional statements allow different sections of code or actions to be executed depending on a condition being met Conditional operators > <	12		

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		>=
		<=
		! not
		!=
		==
		===
		
		If
		<!DOCTYPE
		html>
		
		<html
		lang="en">
		  <head>
		  <meta
		charset="utf-8
		" />
		
		<title></title>
		
		  <script>
		
		  var age =
		10;
		
		  if (age >=
		18)
		  {
		  docume
		nt.write("you
		can vote");
		  }
		
		  </script>
		
		  </head>
		  <body>
		
		  </body>
		</html>
		
		If Else
		
		<!DOCTYPE
		html>
		
		<html
		lang="en">
		  <head>
		  <meta
		charset="utf-8
		" />
		
		<title></title>
		
		  <script>
		
		  var age =
		10;
		
		  if (age >=				

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		18)
		{
		document.write("you can vote");
		}
		
		else
		{
		document.write("you cannot vote");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		
		Nested if
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		var age = 10;
		var country = "india";
		
		if (age >= 18)
		{
		
		if(country=="india")
		{
		
		document.write("you can vote");
		
		}
		
		else
		{			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		docu
		ment.write("onl
		y indians can
		vote");
		
		}
		
		}
		
		else
		{
		docume
		nt.write("you
		cannot vote");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		If else if
		<!DOCTYPE
		html>
		
		<html
		lang="en">
		<head>
		<meta
		charset="utf-8
		" />
		
		<title></title>
		
		<script>
		
		var
		weekday = 13;
		
		if
		(weekday ==
		1)
		{
		docume
		nt.write("mond
		ay");
		}
		else if
		(weekday ==
		2)
		{
		docume
		nt.write("tuesd
		ay");
		}
		
		else if
		(weekday ==			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		3)
		{
		document.write("wednesday");
		}
		else
		{
		document.write("enter value between 1-3");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		
		Switch
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		var weekday = 2;
		
		switch (weekday) {
		
		case 1:
		document.write("monday");
		break;
		
		case 2:
		document.write("tuesday");
		break;
		
		case 3:			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		document.write("wednesday");
		break;
		
		default:
		document.write("enter value between 1-3");
		break;
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		
		Multiple Case Group
		
		switch (browser) {
		case 'Edge':
		alert("You've got the Edge!");
		break;
		
		case 'Chrome':
		case 'Firefox':
		case 'Safari':
		case 'Opera':
		alert('Okay we support these browsers too');
		break;
		
		default:
		alert('We hope that this page looks ok!');
		}			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Conditional operator ‘?’
		Sometimes, we need to assign a variable depending on a condition.
		For instance:
		let accessAllowed;
		let age = prompt('How old are you?', '');
		
		if (age > 18) {
		
		accessAllowed = true;
		} else {
		
		accessAllowed = false;
		}
		
		alert(accessAllowed);
		The so-called “conditional” or “question mark” operator lets us do that in a shorter and simpler way.
		The operator is represented by a question mark ?.
		Sometimes it’s called “ternary”, because the operator has three operands. It is actually the one and only operator in JavaScript which has that many.
		The syntax is:
		let result = condition ? value1 : value2;
		The condition is evaluated: if it’s truthy then value1 is returned, otherwise – value2.
		For example:
		let accessAllowed = (age > 18) ? true : false;			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Technically, we can omit the parentheses around age > 18. The question mark operator has a low precedence, so it executes after the comparison >. This example will do the same thing as the previous one:
		// the comparison operator "age > 18" executes first anyway
		// (no need to wrap it into parentheses)
		let accessAllowed = age > 18 ? true : false;
		But parentheses make the code more readable, so we recommend using them.
		
		Please note:
		In the example above, you can avoid using the question mark operator because the comparison itself returns true/false:
		// the same
		let accessAllowed = age > 18;
		
		Multiple ‘?’
		A sequence of question mark operators ? can return a value that depends on more than one condition.
		For instance:
		let age = prompt('age?', 18);
		
		let message = (age < 3) ? 'Hi, baby!' :
		(age < 18) ?			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		'Hello!' :
		  (age < 100) ?
		'Greetings!' :
		  'What an unusual age!';
		
		alert(message);
		It may be difficult at first to grasp what’s going on. But after a closer look, we can see that it’s just an ordinary sequence of tests:
		1.The first question mark checks whether age < 3.
		2.If true – it returns 'Hi, baby!'. Otherwise, it continues to the expression after the colon “":"”, checking age < 18.
		3.If that’s true – it returns 'Hello!'. Otherwise, it continues to the expression after the next colon “":"”, checking age < 100.
		4.If that’s true – it returns 'Greetings!'. Otherwise, it continues to the expression after the last colon “":"”, returning 'What an unusual age!'.
		Here’s how this looks using if..else:
		if (age < 3) {
		  message = 'Hi, baby!';
		} else if (age < 18) {
		  message = 'Hello!';
		} else if (age < 100) {
		  message = 'Greetings!';
		} else {			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		message = 'What an unusual age!';
		}
		
		Convert String to Number
		var a = parseInt("10");
		
		Assignment
		
		Online Billing
		
		Coffee – 50
		Tea – 25
		Shake 40
		
		Customer name
		Item
		Quantity
		Get price using if / switch and print amount
		Amount
		
		Answer
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		var itemname="Milk";
		var quantity=3;
		var price;
		var amount;
		
		if (itemname == "Coffee")
		{
		price=120;
		}
		if (itemname == "Tea")
		{
		price=60;
		}
		if (itemname == "Milk")
		{
		price=30;
		}
		
		amount= quantity*price;
		document .write(amount);
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>			

12	Lecture-2 For Loop	Loop Loop is repeated execution of code until condition is being met 3 elements 1.Initial value 2. Condition/final value 3.Increment For loop <!DOCTYPE html>	15		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		<html
		lang="en">
		  <head>
		  <meta
		charset="utf-8
		" />
		
		<title></title>
		
		  <script>
		
		  let no = 5;
		
		  for (let i =
		1; i <= 10;
		i++) {
		
		  docume
		nt.write(no * i);
		  docume
		nt.write("
		");
		
		  }
		
		  </script>
		
		  </head>
		  <body>
		
		  </body>
		</html>
		
		Nested loop
		
		<!DOCTYPE
		html>
		
		<html
		lang="en">
		  <head>
		  <meta
		charset="utf-8
		" />
		
		<title></title>
		
		  <script>
		
		  for (let no
		= 5; no <= 8;
		no++)			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		{
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write(no * i);
		document.write("
		");
		
		}
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		Printing in html table
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		document.write('<table border="1">');
		
		for (let no = 5; no <= 8; no++) {
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write("<tr>");
		
		document.write("<t			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		d>"+ no
		+"</td>");
		docu
		ment.write("<t
		d>"+ i
		+"</td>");
		docu
		ment.write("<t
		d>"+ no *
		i+"</td>");
		
		docu
		ment.write("</
		tr>");
		
		}
		
		}
		
		document.wri
		te('</table>');
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>			

13	Lecture-3 While Loop	Loop Loop is repeated execution of code until condition is being met 3 elements 1. Initial value 2. Condition/final value 3. Increment For loop <!DOCTYPE html> <html lang="en"> <head> <meta charset="utf-8 " /> <title></title> <script> let no = 5;	16		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		for (let i = 1; i <= 10; i++) {
		
		document.write(no * i);
		document.write("
		");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		
		Nested loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		for (let no = 5; no <= 8; no++)
		{
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write(no * i);
		document.write("
		");
		
		}			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		Printing in html table
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		document.write('<table border="1">');
		
		for (let no = 5; no <= 8; no++) {
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write("<tr>");
		
		document.write("<td>"+ no +"</td>");
		document.write("<td>"+ i +"</td>");
		document.write("<td>"+ no * i+"</td>");
		
		document.write("</tr>");			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		}
		
		}
		
		document.write('</table>');
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>			

14	Lecture-4 Do While Loop	Loop
		Loop is repeated execution of code until condition is being met
		
		3 elements
		1.Initial value
		2. Condition/final value
		3.Increment
		
		For loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		let no = 5;
		
		for (let i = 1; i <= 10; i++) {
		
		document.write(no * i);
		document.write("
		");
		
		}
			17		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		
		Nested loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		for (let no = 5; no <= 8; no++)
		{
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write(no * i);
		document.write("
		");
		
		}
		
		}
		
		</script>
		
		</head>
		<body>			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		</body>
		</html>
		Printing in html table
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		document.write('<table border="1">');
		
		for (let no = 5; no <= 8; no++) {
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write("<tr>");
		
		document.write("<td>"+ no +"</td>");
		document.write("<td>"+ i +"</td>");
		document.write("<td>"+ no * i+"</td>");
		
		document.write("</tr>");
		
		}
		
		}
		
		document.write('</table>');
		
		</script>
		
		</head>			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		<body> </body> </html>			

15	Lecture-5 For in Loop	Loop
		Loop is repeated execution of code until condition is being met
		
		3 elements
		1.Initial value
		2. Condition/final value
		3.Increment
		
		For loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		let no = 5;
		
		for (let i = 1; i <= 10; i++) {
		
		document.write(no * i);
		document.write("
		");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
			25		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		</html>
		
		Nested loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		for (let no = 5; no <= 8; no++)
		{
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write(no * i);
		document.write("
		");
		
		}
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		Printing in html table
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		<script>
		
		document.write('<table border="1">');
		
		for (let no = 5; no <= 8; no++) {
		for (let i = 1; i <= 10; i = i + 1) {
		document.write("<tr>");
		document.write("<td>"+ no +"</td>");
		document.write("<td>"+ i +"</td>");
		document.write("<td>"+ no * i+"</td>");
		document.write("</tr>");
		}
		}
		document.write('</table>');
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>			

16	Lecture-2 Jumping statements	Loop Loop is repeated execution of code until condition is being met 3 elements 1.Initial value	13		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		2. Condition/final value
		3.Increment
		
		For loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		let no = 5;
		
		for (let i = 1; i <= 10; i++) {
		
		document.write(no * i);
		document.write("
		");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		
		Nested loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		<title></title>
		
		<script>
		
		for (let no = 5; no <= 8; no++)
		{
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write(no * i);
		document.write("
		");
		
		}
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		Printing in html table
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		document.write('<table border="1">');
		
		for (let no = 5; no <= 8; no++) {
		
		for (let i			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		= 1; i <= 10; i = i + 1) {
		
		docu ment.write("<t r>");
		
		docu ment.write("<t d>"+ no +"</td>");
		docu ment.write("<t d>"+ i +"</td>");
		docu ment.write("<t d>"+ no * i+"</td>");
		
		docu ment.write("</ tr>");
		
		}
		
		}
		
		document.wri te('</table>');
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>			

17	Lecture-1 Functions	Loop Loop is repeated execution of code until condition is being met 3 elements 1. Initial value 2. Condition/final value 3. Increment For loop <!DOCTYPE html> <html lang="en"> <head> <meta charset="utf-8	18		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		" />
		<title></title>
		
		<script>
		
		let no = 5;
		
		for (let i = 1; i <= 10; i++) {
		
		document.write(no * i);
		document.write("
		");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		
		Nested loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		for (let no = 5; no <= 8; no++)
		{
		
		for (let i = 1; i <= 10; i = i + 1) {			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		document.write(no * i);
		document.write("
		");
		
		}
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		Printing in html table
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		document.write('<table border="1">');
		
		for (let no = 5; no <= 8; no++) {
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write("<tr>");
		
		document.write("<td>"+ no +"</td>");
		document.write("<td>"+ i +"</td>");			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		document.write("<td>"+ no * i+"</td>");
		
		document.write("</tr>");
		
		}
		
		}
		
		document.write('</table>');
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>			

18	Lecture-1 Intro to arrays	Loop
		Loop is repeated execution of code until condition is being met
		
		3 elements
		1. Initial value
		2. Condition/final value
		3. Increment
		
		For loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		let no = 5;
		
		for (let i = 1; i <= 10; i++) {
		
		document.write(no * i);
			19		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		document.write("
		");
		
		}
		
		</script>
		
		</head>
		<body>
		
		</body>
		</html>
		
		Nested loop
		
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		for (let no = 5; no <= 8; no++)
		{
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write(no * i);
		document.write("
		");
		
		}
		
		}
		
		</script>			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		</head>
		<body>
		
		</body>
		</html>
		Printing in html table
		<!DOCTYPE html>
		
		<html lang="en">
		<head>
		<meta charset="utf-8" />
		
		<title></title>
		
		<script>
		
		document.write('<table border="1">');
		
		for (let no = 5; no <= 8; no++) {
		
		for (let i = 1; i <= 10; i = i + 1) {
		
		document.write("<tr>");
		
		document.write("<td>"+ no +"</td>");
		document.write("<td>"+ i +"</td>");
		document.write("<td>"+ no * i+"</td>");
		
		document.write("</tr>");
		
		}
		
		}
		
		document.write('</table>');			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		</script> </head> <body> </body> </html>			

19	Lecture-2 Detailed Array	Get last elements with “at”
		A recent addition
		let fruits = ["Apple", "Orange", "Plum"];
		
		alert(fruits[fruits.length-1]); // Plum
		
		alert(fruits.at(0)); // Apple
		it is same like alert(fruits[0]);
		
		alert(fruits.at(-1)); // Plum
		In other words, arr.at(i):
		•is exactly the same as arr[i], if i >= 0.
		•for negative values of i, it steps back from the end of the array.
		Methods pop/push, shift/unshift
		A queue is one of the most common uses of an array. In computer science, this means an ordered collection of elements which supports two operations:
		•push appends an element to the end.
		•shift get an element from the beginning,
			20		

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		advancing the queue, so that the 2nd element becomes the 1st. Arrays support both operations. There’s another use case for arrays – the data structure named stack. It supports two operations: •□push adds an element to the end. •□pop takes an element from the end. So new elements are added or taken always from the “end”. A stack is usually illustrated as a pack of cards: new cards are added to the top or taken from the top: For stacks, the latest pushed item is received first, that’s also called LIFO (La st-In-First-Out) principle. For queues, we have FIFO (Firs t-In-First-Out). Arrays in JavaScript can work both as a queue and as a stack. They allow you to add/remove elements, both to/from the beginning or the end. In computer science, the data structure that allows this, is called deque. Methods that			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		work with the end of the array:
		pop
		Extracts the last element of the array and returns it:
		let fruits = ["Apple", "Orange", "Pear"];
		
		alert(fruits.pop()); // remove "Pear" and alert it
		
		alert(fruits); // Apple, Orange
		Both fruits.pop() and fruits.at(-1) return the last element of the array, but fruits.pop() also modifies the array by removing it.
		push
		Append the element to the end of the array:
		let fruits = ["Apple", "Orange"];
		
		fruits.push("Pear");
		
		alert(fruits); // Apple, Orange, Pear
		The call fruits.push(...) is equal to fruits[fruits.length] = ...
		Methods that work with the beginning of the array:
		shift
		Extracts the first element of the array and returns it:
		let fruits = ["Apple", "Orange", "Pear"];
		
		alert(fruits.shift()); // remove Apple and alert it			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		alert(fruits); // Orange, Pear
		unshift
		Add the element to the beginning of the array:
		let fruits = ["Orange", "Pear"];
		
		fruits.unshift('Apple');
		
		alert(fruits); // Apple, Orange, Pear
		Methods push and unshift can add multiple elements at once:
		let fruits = ["Apple"];
		
		fruits.push("Orange", "Peach");
		fruits.unshift("Pineapple", "Lemon");
		
		// ["Pineapple", "Lemon", "Apple", "Orange", "Peach"]
		alert(fruits);
		Performance
		Methods push/pop run fast, while shift/unshift are slow.
		Why is it faster to work with the end of an array than with its beginning? Let’s see what happens during the execution:
		fruits.shift(); // take 1 element from the start
		It’s not enough to take and remove the element with the index 0. Other elements need to be renumbered as well.
		The shift operation must do 3 things:
		1.Remove the element with			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		the index 0.
		2.Move all elements to the left, renumber them from the index 1 to 0, from 2 to 1 and so on.
		3.Update the length property.
		The more elements in the array, the more time to move them, more in-memory operations.
		The similar thing happens with unshift: to add an element to the beginning of the array, we need first to move existing elements to the right, increasing their indexes.
		And what’s with push/pop? They do not need to move anything. To extract an element from the end, the pop method cleans the index and shortens length.
		The actions for the pop operation:
		fruits.pop(); // take 1 element from the end
		The pop method does not need to move anything, because other elements keep their indexes.
		That’s why it’s blazingly fast.
		The similar thing with the push method.
		Loops
		One of the oldest ways to cycle array items is the for loop over			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		indexes:
		let arr =
		["Apple",
		"Orange",
		"Pear"];
		
		for (let i = 0; i
		< arr.length;
		i++) {
		  alert(arr[i]);
		}
		But for arrays
		there is
		another form of
		loop, for..of:
		let fruits =
		["Apple",
		"Orange",
		"Plum"];
		
		// iterates over
		array elements
		for (let fruit of
		fruits) {
		  alert(fruit);
		}
		
		Technically,
		because arrays
		are objects, it
		is also possible
		to use for..in:
		let arr =
		["Apple",
		"Orange",
		"Pear"];
		
		for (let key in
		arr) {
		  alert(arr[key]
		); // Apple,
		Orange, Pear
		}
		
		We should use
		for of instead
		of for..in
		
		A word about
		“length”
		The length
		property
		automatically
		updates when
		we modify the
		array. To be
		precise, it is
		actually not
		the count of
		values in the
		array, but the
		greatest
		numeric index
		plus one.
		For instance, a
		single element
		with a large
		index gives a			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		big length:
		let fruits = [];
		fruits[123] = "Apple";
		
		alert(fruits.length); // 124
		alert(fruits[2]); // undefined
		
		Another interesting thing about the length property is that it’s writable.
		If we increase it manually, nothing interesting happens. But if we decrease it, the array is truncated. The process is irreversible, here’s the example:
		let arr = [1, 2, 3, 4, 5];
		
		arr.length = 2; // truncate to 2 elements
		alert(arr); // [1, 2]
		
		arr.length = 5; // return length back
		alert(arr[5]); // undefined: the values do not return
		So, the simplest way to clear the array is:
		arr.length = 0;.
		
		Multidimensional arrays
		Arrays can have items that are also arrays. We can use it for multi dimensional arrays, for example to store matrices:
		let matrix = [
		  [1, 2, 3],
		  [4, 5, 6],
		  [7, 8, 9]
		];
		
		alert(			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		matrix[1][1]); // 5, the central element
		toString
		Arrays have their own implementation of toString method that returns a comma-separated list of elements.
		For instance:
		let arr = [1, 2, 3];
		
		alert(arr); // 1,2,3
		alert(String(arr)); // 1,2,3
		it will return as string (check in console)
		
		Don’t compare arrays with ==
		Arrays in JavaScript, unlike some other programming languages, shouldn’t be compared with operator ==.
		The strict comparison === is even simpler, as it doesn’t convert types.
		So, if we compare arrays with ==, they are never the same, unless we compare two variables that reference exactly the same array.
		For example:
		alert([] == []); // false
		alert([0] == [0]); // false
		These arrays are technically different objects. So they aren’t equal. The == operator doesn’t do item-by-item comparison.			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Comparison with primitives may give seemingly strange results as well:
		alert(0 == []);
		// true
		
		alert('0' == []
		); // false
		
		Array methods
		Arrays provide a lot of methods. To make things easier, in this chapter they are split into groups.
		Add/remove items
		We already know methods that add and remove items from the beginning or the end:
		•arr.push(...items) – adds items to the end,
		•arr.pop() – extracts an item from the end,
		•arr.shift() – extracts an item from the beginning,
		•arr.unshift(...items) – adds items to the beginning.
		splice
		How to delete an element from the array?
		The arrays are objects, so we can try to use delete:
		let arr = ["I",
		"go", "home"];
		
		delete arr[1]; //
		remove "go"
		
		alert(arr[1]);
		// undefined
		
		// now arr =
		["I", , "home"];
		alert(
		arr.length); //
		3
		The element			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		was removed, but the array still has 3 elements, we can see that arr.length == 3.
		That’s natural, because delete obj.key removes a value by the key. It’s all it does. Fine for objects. But for arrays we usually want the rest of elements to shift and occupy the freed place. We expect to have a shorter array now
		
		So, special methods should be used.
		The arr.splice method is a swiss army knife for arrays. It can do everything: insert, remove and replace elements.
		The syntax is: arr.splice(start[, deleteCount, elem1, ..., elemN])
		
		It modifies arr starting from the index start: removes deleteCount elements and then inserts elem1, ..., elemN at their place. Returns the array of removed elements.
		This method is easy to grasp by examples.
		Let’s start with the deletion:
		let arr = ["I", "study", "JavaScript"];
		
		arr.splice(1, 1); // from index 1			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		remove 1 element
		
		alert(arr); // ["I", "JavaScript"] Easy, right? Starting from the index 1 it removed 1 element.
		In the next example we remove 3 elements and replace them with the other two:
		let arr = ["I", "study", "JavaScript", "right", "now"];
		
		// remove 3 first elements and replace them with another
		arr.splice(0, 3, "Let's", "dance");
		
		alert(arr) // now ["Let's", "dance", "right", "now"]
		
		Here we can see that splice returns the array of removed elements:
		let arr = ["I", "study", "JavaScript", "right", "now"];
		
		// remove 2 first elements
		let removed = arr.splice(0, 2);
		
		alert(removed); // "I", "study" <-- array of removed elements
		The splice method is also able to insert the elements without any removals. For that we need to set deleteCount to 0:
		let arr = ["I",			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		"study",
		"JavaScript"];
		
		// from index 2
		// delete 0
		// then insert
		"complex" and
		"language"
		arr.splice(2, 0,
		"complex",
		"language");
		
		alert(arr); //
		"I", "study",
		"complex",
		"language",
		"JavaScript"
		
		Negative
		indexes
		allowed
		Here and in
		other array
		methods,
		negative
		indexes are
		allowed. They
		specify the
		position from
		the end of the
		array, like
		here:
		let arr = [1, 2,
		5];
		
		// from index -1
		(one step from
		the end)
		// delete 0
		elements,
		// then insert 3
		and 4
		arr.splice(-1, 0,
		3, 4);
		
		alert(arr); //
		1,2,3,4,5
		slice
		The method
		arr.slice is
		much simpler
		than similar-
		looking
		arr.splice.
		The syntax is:
		arr.slice([start],
		[end])
		It returns a
		new array
		copying to it all
		items from
		index start to
		end (not
		including end).
		Both start and
		end can be
		negative, in
		that case			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		position from array end is assumed.
		It’s similar to a string method str.slice, but instead of substrings it makes subarrays.
		For instance:
		let arr = ["t", "e", "s", "t"];
		
		alert(arr.slice(1, 3)); // e,s (copy from 1 to 3)
		
		alert(arr.slice(-2)); // s,t (copy from -2 till the end)
		We can also call it without arguments: arr.slice() creates a copy of arr. That’s often used to obtain a copy for further transformations that should not affect the original array.
		concat
		The method arr.concat creates a new array that includes values from other arrays and additional items.
		The syntax is: arr.concat(arg1, arg2...)
		
		It accepts any number of arguments – either arrays or values.
		The result is a new array containing items from arr, then arg1, arg2 etc.
		If an argument argN is an array, then all its elements are copied. Otherwise, the argument itself is copied.			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		For instance:
		let arr = [1, 2];
		
		// create an array from: arr and [3,4]
		alert(arr.concat([3, 4])); // 1,2,3,4
		
		// create an array from: arr and [3,4] and [5,6]
		alert(arr.concat([3, 4], [5, 6])); // 1,2,3,4,5,6
		
		// create an array from: arr and [3,4], then add values 5 and 6
		alert(arr.concat([3, 4], 5, 6)); // 1,2,3,4,5,6
		Iterate: forEach
		The arr.forEach method allows to run a function for every element of the array.
		The syntax:
		arr.forEach(function(item, index, array) {
		  // ... do something with item
		});
		For instance, this shows each element of the array:
		// for each element call alert
		let arr= ["Bilbo", "Gandalf", "Nazgul"];
		arr.forEach(alert);
		
		or
		
		["Bilbo", "Gandalf", "Nazgul"].forEach(alert);
		
		And this code is more elaborate about their			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		positions in the target array:
		
		let arr= ["Bilbo", "Gandalf", "Nazgul"];
		
		function showarray (item, index, array)
		{
		alert(`${item} is at index ${index} in ${array}`);
		}
		
		arr.forEach(showarray);
		
		or
		
		let arr= ["Bilbo", "Gandalf", "Nazgul"];
		arr.forEach((item, index, array) =>
		{
		
		alert(`${item} is at index ${index} in ${array}`);
		});
		
		The result of the function (if it returns any) is thrown away and ignored.
		Searching in array
		Now let’s cover methods that search in an array.
		indexOf/lastIndexOf and includes
		The methods arr.indexOf and arr.includes have the similar syntax and do essentially the same as their string counterparts, but operate on items instead of characters:			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		•arr.indexOf(item, from) – looks for item starting from index from, and returns the index where it was found, otherwise -1.
		•arr.includes(item, from) – looks for item starting from index from, returns true if found.
		Usually these methods are used with only one argument: the item to search. By default, the search is from the beginning.
		For instance:
		let arr = [1, 2, 3];
		
		alert(arr.indexOf(1)); // 1
		alert(arr.indexOf(2)); // 2
		alert(arr.indexOf(null)); // -1
		alert(arr.includes(1)); // true
		
		If we want to check if item exists in the array, and don’t need the exact index, then arr.includes is preferred.
		
		The method arr.lastIndexOf is the same as indexOf, but looks for from right to left.
		let fruits = ['Apple', 'Orange', 'Apple']
		
		alert(fruits.indexOf('Apple')); // 0 (first Apple)
		alert(fruits.lastIndexOf('Apple')); // 2 (last			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Apple)
		
		The includes method handles NaN correctly
		A minor, but noteworthy feature of includes is that it correctly handles NaN, unlike indexOf:
		const arr = [NaN];
		alert(arr.indexOf(NaN)); // -1 (wrong, should be 0)
		alert(arr.includes(NaN));// true (correct)
		That’s because includes was added to JavaScript much later and uses the more up to date comparison algorithm internally.
		find and findIndex/findLastIndex
		Imagine we have an array of objects. How do we find an object with the specific condition?
		Here the arr.find(fn) method comes in handy.
		The syntax is:
		let result = arr.find(function(item, index, array) {
		  // if true is returned, item is returned and iteration is stopped
		  // for falsy scenario returns undefined
		});
		The function is called for elements of the array, one after another:
		•item is the element.
		•index is its			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		index.
		•□array is the array itself.
		If it returns true, the search is stopped, the item is returned. If nothing found, undefined is returned.
		For example, we have an array of users, each with the fields id and name. Let’s find the one with id == 1:
		let users = [
		  {id: 1, name: "John"},
		  {id: 2, name: "Pete"},
		  {id: 3, name: "Mary"}
		];
		
		let user = users.find(item => item.id == 1);
		
		alert(user.name); // John
		
		In real life arrays of objects is a common thing, so the find method is very useful.
		Note that in the example we provide to find the function item => item.id == 1 with one argument. That’s typical, other arguments of this function are rarely used.
		The arr.findIndex method has the same syntax, but returns the index where the element was found instead of the element itself.			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		The value of -1 is returned if nothing is found.
		The arr.findLastIndex method is like findIndex, but searches from right to left, similar to lastIndexOf.
		Here’s an example:
		let users = [
		  {id: 1, name: "John"},
		  {id: 2, name: "Pete"},
		  {id: 3, name: "Mary"},
		  {id: 4, name: "John"}
		];
		
		// Find the index of the first John
		alert(users.findIndex(user => user.name == 'John')); // 0
		
		// Find the index of the last John
		alert(users.findLastIndex(user => user.name == 'John')); // 3
		
		filter
		The find method looks for a single (first) element that makes the function return true.
		If there may be many, we can use arr.filter(fn).
		The syntax is similar to find, but filter returns an array of all matching elements:
		let results = arr.filter(function (item, index, array) {
		  // if true item is pushed to results and the iteration continues			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		// returns empty array if nothing found
		});
		
		For instance:
		let users = [
		{id: 1, name: "John"},
		{id: 2, name: "Pete"},
		{id: 3, name: "Mary"}
		];
		
		// returns array of the first two users
		let someUsers = users.filter(item => item.id < 3);
		
		alert(someUsers.length); // 2
		
		console.log(someUsers);
		//check console for array returned
		
		Transform an array
		Let’s move on to methods that transform and reorder an array.
		map
		The arr.map method is one of the most useful and often used.
		It calls the function for each element of the array and returns the array of results.
		The syntax is:
		let result = arr.map(function(item, index, array) {
		// returns the new value instead of item
		});
		For instance, here we transform each element into its length:
		let arr=["Bilbo",			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		"Gandalf", "Nazgul"]; let lengths = arr.map(item => item.length); alert(lengths); // 5,7,6
		
		sort(fn)
		The call to arr.sort() sorts the array in place, changing its element order.
		It also returns the sorted array, but the returned value is usually ignored, as arr itself is modified.
		For instance:
		let arr = [1, 2, 15];
		
		// the method reorders the content of arr
		arr.sort();
		
		alert(arr); // 1, 15, 2
		Did you notice anything strange in the outcome?
		The order became 1, 15, 2. Incorrect.
		But why?
		The items are sorted as strings by default.
		Literally, all elements are converted to strings for comparisons.
		For strings, lexicographic ordering is applied and indeed "2" > "15".
		To use our own sorting order, we need to supply a function as the argument of arr.sort().
		The function should compare two arbitrary			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		values and return:
		function compare(a, b) {
		  if (a > b) return 1; // if the first value is greater than the second
		  if (a == b) return 0; // if values are equal
		  if (a < b) return -1; // if the first value is less than the second
		}
		For instance, to sort as numbers:
		function compareNumeric(a, b) {
		  if (a > b) return 1;
		  if (a == b) return 0;
		  if (a < b) return -1;
		}
		
		let arr = [1, 2, 15];
		
		arr.sort(compareNumeric);
		
		alert(arr); // 1, 2, 15
		Now it works as intended.
		Let’s step aside and think what’s happening. The arr can be array of anything, right? It may contain numbers or strings or objects or whatever. We have a set of some items. To sort it, we need an ordering function that knows how to compare its elements. The default is a string order.
		The arr.sort(fn) method			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		implements a generic sorting algorithm. We don’t need to care how it internally works (an optimized quicksort or Timsort most of the time). It will walk the array, compare its elements using the provided function and reorder them, all we need is to provide the fn which does the comparison.
		
		Use localeCompare for strings Remember strings comparison algorithm? It compares letters by their codes by default.
		For many alphabets, it’s better to use str.localeCompare method to correctly sort letters, such as Ö.
		For example, let’s sort a few countries in German:
		let countries = ['Österreich', 'Andorra', 'Vietnam'];
		
		alert(countries.sort((a, b) => a > b ? 1 : -1)); // Andorra, Vietnam, Österreich (wrong)
		
		alert(countries.sort((a, b) => a.localeCompare(b))); // Andorra,Österreich,Vietnam (correct!)
		reverse			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		The method arr.reverse reverses the order of elements in arr.
		For instance:
		let arr = [1, 2, 3, 4, 5];
		arr.reverse();
		
		alert(arr); // 5,4,3,2,1
		It also returns the array arr after the reversal.
		split and join
		Here’s the situation from real life. We are writing a messaging app, and the person enters the comma-delimited list of receivers: John, Pete, Mary. But for us an array of names would be much more comfortable than a single string. How to get it?
		The str.split(delim) method does exactly that. It splits the string into an array by the given delimiter delim.
		In the example below, we split by a comma followed by space:
		let names = 'Bilbo, Gandalf, Nazgul';
		
		let arr = names.split(', ');
		
		for (let name of arr) {
		  alert(`A message to ${name}.`); // A message to Bilbo (and other names)
		}
		The split			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		method has an optional second numeric argument – a limit on the array length. If it is provided, then the extra elements are ignored. In practice it is rarely used though:
		let arr = 'Bilbo, Gandalf, Nazgul, Saruman'.split(', ', 2);
		
		alert(arr); // Bilbo, Gandalf
		
		Split into letters
		The call to split(s) with an empty s would split the string into an array of letters:
		let str = "test";
		
		alert(str.split('')); // t,e,s,t
		
		The call arr.join(glue) does the reverse to split. It creates a string of arr items joined by glue between them.
		For instance:
		let arr = ['Bilbo', 'Gandalf', 'Nazgul'];
		
		let str = arr.join(';'); // glue the array into a string using ;
		
		alert(str); // Bilbo;Gandalf;Nazgul
		reduce/reduceRight
		When we need to iterate over an array – we			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		can use forEach, for or for..of. When we need to iterate and return the data for each element – we can use map. The methods arr.reduce and arr.reduceRight also belong to that breed, but are a little bit more intricate. They are used to calculate a single value based on the array. The syntax is: let value = arr.reduce(function(accumulator, item, index, array) { // ... }, [initial]); The function is applied to all array elements one after another and “carries on” its result to the next call. Arguments: •accumulator – is the result of the previous function call, equals initial the first time (if initial is provided). •item – is the current array item. •index – is its position. •array – is the array. As function is applied, the result of the previous function call is passed to the next one as the first argument. So, the first argument is essentially the accumulator that stores the combined result of all previous			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		executions.
		And at the end it becomes the result of reduce.
		Sounds complicated? The easiest way to grasp that is by example.
		Here we get a sum of an array in one line:
		let arr = [1, 2, 3, 4, 5];
		
		let result = arr.reduce((sum, current) => sum + current, 0);
		
		alert(result); // 15
		
		let arr = [1, 2, 3, 4, 5];
		
		let result = arr.reduce((sum, current) => sum + current, 100);
		
		alert(result); // 115
		
		The function passed to reduce uses only 2 arguments, that’s typically enough.
		Let’s see the details of what’s going on.
		1.On the first run, sum is the initial value (the last argument of reduce), equals 0, and current is the first array element, equals 1. So the function result is 1.
		2.On the second run, sum = 1, we add the second array element			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		(2) to it and return.
		3.□On the 3rd run, sum = 3 and we add one more element to it, and so on…
		The calculation flow:
		Or in the form of a table, where each row represents a function call on the next array element:
		
		□sum□current result
		the first call□0 1□1
		the second call 1□2□3
		the third call□3 3□6
		the fourth call 6□4□10
		the fifth call 10□5□15
		Here we can clearly see how the result of the previous call becomes the first argument of the next one.
		We also can omit the initial value:
		let arr = [1, 2, 3, 4, 5];
		
		// removed initial value from reduce (no 0)
		let result = arr.reduce((sum, current) => sum + current);
		
		alert(result);
		// 15
		
		The result is the same. That’s because if there’s no initial, then reduce takes the first element of the array as the initial value and starts the iteration from			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		the 2nd element.
		The calculation table is the same as above, minus the first row.
		But such use requires an extreme care.
		If the array is empty, then reduce call without initial value gives an error.
		Here’s an example:
		let arr = [];
		
		// Error: Reduce of empty array with no initial value
		// if the initial value existed, reduce would return it for the empty arr.
		arr.reduce((sum, current) => sum + current);
		So it’s advised to always specify the initial value.
		The method arr.reduceRight does the same, but goes from right to left.
		Array.isArray
		Arrays do not form a separate language type.
		They are based on objects.
		So typeof does not help to distinguish a plain object from an array:
		alert(typeof {}); // object
		alert(typeof []); // object (same)
		
		…But arrays are used so often that there’s a special method for that: Array.isArray(value).
		It returns true			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		if the value is an array, and false otherwise.
		alert(Array.isArray({})); // false
		
		alert(Array.isArray([])); // true
		
		Most methods support “thisArg” Almost all array methods that call functions – like find, filter, map, with a notable exception of sort, accept an optional additional parameter thisArg.
		That parameter is not explained in the sections above, because it’s rarely used.
		But for completeness we have to cover it.
		Here’s the full syntax of these methods:
		arr.find(func, thisArg);
		arr.filter(func, thisArg);
		arr.map(func, thisArg);
		// ...
		// thisArg is the optional last argument
		The value of thisArg parameter becomes this for func.
		For example, here we use a method of army object as			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		a filter, and thisArg passes the context:
		let army = {
		  minAge: 18,
		  maxAge: 27,
		  canJoin(user)
		{
		  return user.age >= this.minAge && user.age < this.maxAge;
		  }
		};
		
		let users = [
		  {age: 16},
		  {age: 20},
		  {age: 23},
		  {age: 30}
		];
		
		// find users, for who army.canJoin returns true
		let soldiers = users.filter(army.canJoin, army);
		
		alert(soldiers.length); // 2
		alert(soldiers[0].age); // 20
		alert(soldiers[1].age); // 23
		
		If in the example above we used users.filter(army.canJoin), then army.canJoin would be called as a standalone function, with this=undefined, thus leading to an instant error.
		A call to users.filter(army.canJoin, army) can be replaced with users.filter(user => army.canJoin(user)), that does the same. The latter is used more often, as it’s a bit easier to understand for most people.			

20	Lecture-1 Intro to Objects	JavaScript	21		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		Objects JavaScript is an object-based language. Everything is an object in JavaScript. Here, we don't create class to get the object. But, we direct create objects. All JavaScript values, except primitives, are objects. JavaScript Primitives A primitive value is a value that has no properties or methods. A primitive data type is data that has a primitive value. JavaScript defines 5 types of primitive data types: •□string •□number •□boolean •□null •□undefined Objects are Variables JavaScript variables can contain single values: Example let person = "John Doe"; Objects are variables too. But objects can contain many values. Object values are written as name : value pairs (name and value separated by a colon). Example let person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		A JavaScript object is a collection of named values It is a common practice to declare objects with the const keyword. Example const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}; document.write(person.firstName); document.write(person.lastName); document.write(person.age); document.write(person.eyeColor); _____ _____ _____ _____ _____ _____ Object Properties The named values, in JavaScript objects, are called properties. Property Value firstName John lastName Doe age 50 eyeColor blue Object Methods Methods are actions that can be performed on objects. Object properties can be both primitive values, other objects, and functions. An object method is an object property containing a function definition. Property Value				

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		firstName□John lastName□Doe age□50 eyeColor□blue fullName function() {return this.firstName + " " + this.lastName; } JavaScript objects are containers for named values, called properties and methods. Creating a JavaScript Object by different ways With JavaScript, you can define and create your own objects. There are different ways to create new objects: •□Create a single object, using an object literal. •□Create a single object, with the keyword new. •□Define an object constructor, and then create objects of the constructed type. •□Create an object using Object.create() . _____ _____ _____ Using an Object Literal This is the easiest way to create a JavaScript Object. Using an object literal, you both define and create an object in one statement. An object			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		literal is a list of name:value pairs (like age:50) inside curly braces {}.
		The following example creates a new JavaScript object with four properties:
		Example
		const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
		Spaces and line breaks are not important. An object definition can span multiple lines:
		Example
		const person = {
		  firstName: "John",
		  lastName: "Doe",
		  age: 50,
		  eyeColor: "blue"
		};
		This example creates an empty JavaScript object, and then adds 4 properties:
		Example
		const person = {};
		person.firstName = "John";
		person.lastName = "Doe";
		person.age = 50;
		person.eyeColor = "blue";
		
		Using the JavaScript Keyword new
		The following example create a new JavaScript object using new Object(), and then adds 4 properties:
		Example
		const person =			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		new Object();
		person.firstName = "John";
		person.lastName = "Doe";
		person.age = 50;
		person.eyeColor = "blue";
		The examples above do exactly the same.
		But there is no need to use new Object().
		For readability, simplicity and execution speed, use the object literal method.
		_____
		_____
		_____
		JavaScript Objects are addressed by Reference
		Objects are addressed by reference, not by value.
		If person is an object, the following statement will not create a copy of person:
		const x = person; // Will not create a copy of person.
		The object x is not a copy of person. It is person. Both x and person are the same object.
		Any changes to x will also change person, because x and person are the same object.
		Example
		const person = {
		  firstName:"John",
		lastName:"Doe",
		  age:50,
		eyeColor:"blue"
		}
		
		const x =			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		person;
		x.age = 10;
		// Will change both x.age and person.age
		
		JavaScript Object Properties
		Properties are the most important part of any JavaScript object.
		_____
		_____
		_____
		JavaScript Properties
		Properties are the values associated with a JavaScript object.
		A JavaScript object is a collection of unordered properties.
		Properties can usually be changed, added, and deleted.
		
		Accessing JavaScript Properties
		The syntax for accessing the property of an object is:
		objectName.property // person.age
		document.write(person.firstName);
		document.write(person.lastName);
		document.write(person.age);
		document.write(person.eyeColor);
		
		or
		objectName["property"] // person["age"]
		document.write(person["firstName"]);			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		document.write(person["lastName"]);
		document.write(person["age"]);
		document.write(person["eyeColor"]);
		
		or
		objectName[expression] // x = "age";
		person[x]
		The expression must evaluate to a property name.
		Example 1
		person.firstname + " is " + person.age + " years old.";
		Example 2
		person["firstName"] + " is " + person["age"] + " years old.";
		
		JavaScript for...in Loop
		The JavaScript for...in statement loops through the properties of an object.
		Syntax
		for (let variable in object) {
		  // code to be executed
		}
		The block of code inside of the for...in loop will be executed once for each property.
		Looping through the properties of an object:
		Example
		const person = { fname:"John", lname:"Doe", age: 25 };
		for (let x in person)
		 {
		  document.write(person[x]);
		  document.write("
		");
		}			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		}Adding New Properties You can add new properties to an existing object by simply giving it a value. Assume that the person object already exists - you can then give it new properties: Example const person = { fname:" John", lname:" Doe", age: 25 }; person.nationality = "indian"; for (let x in person) { document.write(person[x]); document.write(" "); } Deleting Properties The delete keyword deletes a property from an object: Example const person = { firstName: "John", lastName: "Doe", age: 50, eyeColor: "blue" }; delete person.age; or delete person["age"]; Example const person = { fname:" John", lname:" Doe", age: 25 }; delete person.age;			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		for (let x in person) {
		  document.write(person[x]);
		  document.write("
		");
		}
		The delete keyword deletes both the value of the property and the property itself.
		
		Nested Objects
		Values in an object can be another object:
		Example
		
		let myObj = {
		  name:"John",
		  age:30,
		  cars: {
		  car1:"Ford",
		  car2:"BMW",
		  car3:"Fiat"
		  }
		}
		
		document.write(myObj.cars.car2);
		You can access nested objects using the dot notation or the bracket notation:
		Example
		myObj.cars.car2;
		or:
		Example
		myObj.cars["car2"];
		or:
		Example
		myObj["cars"]["car2"];
		
		Nested Arrays and Objects
		Values in objects can be arrays, and values in arrays can be objects:
		Example
		const myObj = {
		  name: "John",
		  age: 30,
		  cars: [			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		{name:"Ford", models:["Fiesta", "Focus", "Mustang"]},
		
		{name:"BMW", models:["320", "X3", "X5"]},
		
		{name:"Fiat", models:["500", "Panda"]}
		]
		}
		To access arrays inside arrays, use a for-in loop for each array:
		Example
		
		const myObj = {
		name: "John",
		age: 30,
		cars: [
		
		{name:"Ford", models:["Fiesta", "Focus", "Mustang"]},
		
		{name:"BMW", models:["320", "X3", "X5"]},
		
		{name:"Fiat", models:["500", "Panda"]}
		]
		}
		
		for (let i in myObj.cars) {
		document.write(" " + myObj.cars[i].name + " ");
		for (let j in myObj.cars[i].models)
		{
		document.write(myObj.cars[i].models[j]+"
		");
		}
		}			

21	Lecture-2 Object-This	Object methods, "this" Objects are usually created to represent entities of the real world, like users, orders	22		

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		and so on:
		let user = {
		  name: "John",
		  age: 30
		};
		Method examples
		For a start, let’s teach the user to say hello:
		let user = {
		  name: "John",
		  age: 30
		};
		
		user.sayHi = function() {
		
		alert("Hello!");
		};
		
		user.sayHi(); // Hello!
		Here we’ve just used a Function Expression to create a function and assign it to the property user.sayHi of the object.
		Then we can call it as user.sayHi(). The user can now speak!
		A function that is a property of an object is called its method.
		So, here we’ve got a method sayHi of the object user.
		Of course, we could use a pre-declared function as a method, like this:
		let user = {
		  // ...
		};
		
		// first, declare function sayHi() {
		
		alert("Hello!");
		}
		
		// then add as a method
		user.sayHi = sayHi;			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		user.sayHi(); // Hello!
		
		Method shorthand
		There exists a shorter syntax for methods in an object literal:
		// these objects do the same
		
		user = {
		  sayHi:
		function() {
		
		alert("Hello");
		  }
		};
		
		// method shorthand looks better, right?
		user = {
		  sayHi()
		  { // same as "sayHi:
		function(){...}"
		
		alert("Hello");
		  }
		};
		
		user.sayHi(); // Hello!As demonstrated, we can omit "function" and just write sayHi().
		“this” in methods
		It’s common that an object method needs to access the information stored in the object to do its job.
		For instance, the code inside user.sayHi() may need the name of the user.
		To access the object, a method can use the this keyword.
		The value of this is the object “before dot”, the one used to call the method.			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		For instance:
		let user = {
		  name: "John",
		  age: 30,
		
		  sayHi() {
		    // "this" is the "current object"
		
		alert(this.name);
		  }
		
		};
		
		user.sayHi(); // John
		Here during the execution of user.sayHi(), the value of this will be user.
		Technically, it’s also possible to access the object without this, by referencing it via the outer variable:
		let user = {
		  name: "John",
		  age: 30,
		
		  sayHi() {
		    alert(user.name); // "user" instead of "this"
		  }
		
		};
		...But such code is unreliable. If we decide to copy user to another variable, e.g. admin = user and overwrite user with something else, then it will access the wrong object.
		That’s demonstrated below:
		let user = {
		  name: "John",
		  age: 30,
		
		  sayHi() {
		    alert(user.name); // leads to an			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		error
		  }
		
		};
		
		let admin =
		user;
		user = null; //
		overwrite to
		make things
		obvious
		
		admin.sayHi();
		// TypeError:
		Cannot read
		property
		'name' of null
		If we used
		this.name
		instead of
		user.name
		inside the
		alert, then the
		code would
		work.
		“this” is not
		bound
		In JavaScript,
		keyword this
		behaves unlike
		most other
		programming
		languages. It
		can be used in
		any function,
		even if it’s not
		a method of an
		object.
		There’s no
		syntax error in
		the following
		example:
		function
		sayHi() {
		  alert(
		this.name);
		}
		The value of
		this is
		evaluated
		during the run-
		time,
		depending on
		the context.
		For instance,
		here the same
		function is
		assigned to
		two different
		objects and
		has different
		“this” in the
		calls:
		let user = {
		name: "John"
		};
		let admin = {
		name: "Admin"			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		};
		
		function sayHi() {
		  alert(this.name);
		}
		
		// use the same function in two objects
		user.f = sayHi;
		admin.f = sayHi;
		
		// these calls have different this
		// "this" inside the function is the object "before the dot"
		user.f(); // John (this == user)
		admin.f(); // Admin (this == admin)
		
		admin['f'](); // Admin (dot or square brackets access the method – doesn't matter)
		The rule is simple: if obj.f() is called, then this is obj during the call of f. So it’s either user or admin in the example above.
		Calling without an object: this == undefined
		We can even call the function without an object at all:
		function sayHi() {
		  alert(this);
		}
		
		sayHi(); // undefined
		In this case this is undefined in strict mode. If we try to access this.name, there will be an error.			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		The consequences of unbound this If you come from another programming language, then you are probably used to the idea of a "bound this", where methods defined in an object always have this referencing that object. In JavaScript this is “free”, its value is evaluated at call-time and does not depend on where the method was declared, but rather on what object is “before the dot”. The concept of run-time evaluated this has both pluses and minuses. On the one hand, a function can be reused for different objects. On the other hand, the greater flexibility creates more possibilities for mistakes. Arrow functions have no “this” Arrow functions are special: they don’t have their “own” this. If we reference this from such a function, it’s taken from the outer “normal” function. For instance, here arrow() uses this from the outer user.sayHi() method:			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		let user = {
		  firstName: "Ilya",
		  sayHi() {
		  let arrow = () => alert(this.firstName);
		  arrow();
		  }
		};
		
		user.sayHi(); // Ilya			

22	Lecture-3 Object from Array	JavaScript Object Methods
		Example
		const person = {
		  firstName: "John",
		  lastName: "Doe",
		  id: 5566,
		  fullName: function() {
		  return this.firstName + " " + this.lastName;
		  }
		};
		_____
		_____
		_____
		The this Keyword
		In a function definition, this refers to the "owner" of the function.
		In the example above, this is the person object that "owns" the fullName function.
		In other words, this.firstName means the firstName property of this object.
		
		JavaScript Methods
		JavaScript methods are actions that can be performed on objects.
		A JavaScript method is a property containing a function definition.
			23		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		Property□Value firstName□John lastName□Doe age□50 eyeColor□blue fullName function() {return this.firstName + " " + this.lastName; } Methods are functions stored as object properties. _____ _____ _____ Accessing Object Methods You access an object method with the following syntax: objectName.m ethodName() You will typically describe fullName() as a method of the person object, and fullName as a property. The fullName property will execute (as a function) when it is invoked with (). This example accesses the fullName() method of a person object: Example name = person .fullName(); If you access the fullName property, without (), it will return the function definition: Example name = person .fullName; Adding a Method to an Object Adding a new method to an object is easy: Example const person =			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		{
		firstName:
		"John",
		lastName:
		"Doe",
		id: 5566,
		};
		
		person.fullNam
		e=function() {
		return
		this.firstName
		+ " " +
		this.lastName;
		};
		
		document.writ
		e(person.fullNa
		me());
		
		Example :
		using typeof to
		check property
		type
		const person =
		{
		firstName:
		"John",
		lastName:
		"Doe",
		id: 5566,
		};
		
		person.fullNam
		e=function() {
		return
		this.firstName
		+ " " +
		this.lastName;
		};
		
		for (let x in
		person)
		{
		
		if (typeof(per
		son[x])=="func
		tion")
		{
		document.
		write(person[x]
		());
		document.
		write("
		");
		}
		else
		{
		document
		.write(person[x
		]);
		document
		.write("
		");
		
		}
		
		}			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Output
		John
		Doe
		5566
		John Doe
		
		Using Built-In Methods
		This example uses the toUpperCase() method of the String object, to convert a text to uppercase:
		let message = "Hello world!";
		let x = message.toUpperCase();
		document.write(x);
		
		The value of x, after execution of the code above will be:
		HELLO WORLD!
		Example
		person.name = function () {
		  return (this.firstName + " " + this.lastName).toUpperCase();
		};
		
		JavaScript Display Objects
		____
		How to Display JavaScript Objects?
		Displaying a JavaScript			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		object will output [object Object]. Example
		const person = {
		  name: "John",
		  age: 30,
		  city: "New York"
		};
		
		document.write(person);
		Some common solutions to display JavaScript objects are:
		•Displaying the Object Properties by name
		•Displaying the Object Properties in a Loop
		•Displaying the Object using Object.values()
		•Displaying the Object using JSON.stringify()
		
		Displaying Object Properties
		The properties of an object can be displayed as a string:
		Example
		
		<!DOCTYPE html>
		 <html lang="en">
		  <head>
		  <meta charset="utf-8" />
		
		<title></title>
		  </head>
		  <body>
		  nspl
		<script>
		
		const person = {
		  name: "John",
		  age: 30,			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		city: "New York"
		};
		
		document.getElementById("demo").innerHTML = person.name + "," + person.age + "," + person.city;
		</script>
		  </body>
		</html>
		_____
		_____
		_____
		Displaying the Object in a Loop
		The properties of an object can be collected in a loop:
		Example
		
		<!DOCTYPE html>
		<html lang="en">
		  <head>
		  <meta charset="utf-8" />
		
		<title></title>
		  </head>
		  <body>
		
		nspl
		  <script>
		
		const person = {
		  name: "John",
		  age: 30,
		  city: "New York"
		};
		
		let txt = "";
		for (let x in person) {
		txt += person[x] + " ";
		};
		document.getElementById("demo").innerHTML = txt;
		
		</script>
		
		  </body>
		</html>			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		You must use person[x] in the loop. person.x will not work (Because x is a variable).
		
		Using Object.values()
		Any JavaScript object can be converted to an array using Object.values():
		const person = {
		  name: "John",
		  age: 30,
		  city: "New York"
		};
		
		const myArray = Object.values(person);
		myArray is now a JavaScript array, ready to be displayed:
		Example
		<!DOCTYPE html>
		 <html lang="en">
		  <head>
		  <meta charset="utf-8" />
		
		<title></title>
		  </head>
		  <body>
		
		nspl
		
		<script>
		 const person = {
		  name: "John",
		  age: 30,
		  city: "New York"
		};
		const myArray = Object.values(person);
		document.getElementById("demo").innerHTML = myArray;
		</script>
		  </body>			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		</html>
		
		Using JSON.stringify()
		Any JavaScript object can be stringified (converted to a string) with the JavaScript function JSON.stringify():
		const person = {
		  name: "John",
		  age: 30,
		  city: "New York"
		};
		
		let myString = JSON.stringify(person);
		myString is now a JavaScript string, ready to be displayed:
		Example
		
		<!DOCTYPE html>
		 <html lang="en">
		  <head>
		  <meta charset="utf-8" />
		
		<title></title>
		  </head>
		  <body>
		
		  nspl
		
		<script>
		
		const person = {
		  name: "John",
		  age: 30,
		  city: "New York"
		};
		
		let myString = JSON.stringify(person);
		document.getElementById("demo").innerHTM			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		L = myString;
		</script>
		    </body>
		</html>
		
		The result will be a string following the JSON notation:
		{"name":"John"
		,"age":50,"city"
		:"New York"}
		JSON.stringify()
		is included in
		JavaScript and
		supported in all
		major
		browsers.
		
		Stringify Dates
		JSON.stringify
		converts dates
		into strings:
		Example
		
		<!DOCTYPE
		html>
		
		<html
		lang="en">
		    <head>
		        <meta
		charset="utf-8
		" />
		
		<title></title>
		    </head>
		    <body>
		
		        <div
		id="demo">
		nspl
		
		<script>
		const person =
		{
		  name: "John",
		  today: new
		Date()
		};
		
		let myString =
		JSON.stringify(
		person);
		document.getE
		lementById("de
		mo").innerHTM
		L = myString;
		
		</script>
		
		    </body>
		</html>
		_____
		_____
		_____			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Stringify Functions
		JSON.stringify will not stringify functions:
		Example
		const person = {
		  name: "John",
		  age: function () {return 30;}
		};
		
		let myString = JSON.stringify(person);
		document.getElementById("demo").innerHTML = myString;
		This can be "fixed" if you convert the functions into strings before stringifying.
		Example
		const person = {
		  name: "John",
		  age: function () {return 30;}
		};
		person.age = person.age().toString();
		
		let myString = JSON.stringify(person);
		document.getElementById("demo").innerHTML = myString;
		output
		{"name":"John","age":"30"}
		
		Stringify Arrays
		It is also possible to stringify JavaScript arrays:
		Example
		const arr = ["John",
		"Peter", "Sally", "Jane"];
		
		let myString = JSON.stringify(arr);
		document.getElementById("demo").innerHTML = myString;
		The result will			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video
		be a string following the JSON notation: ["John","Peter", "Sally","Jane"]			
23	Lecture-4 Object Methods	Object.keys, values, entries In the previous chapter we saw methods map.keys(), map.values(), map.entries(). These methods are generic, there is a common agreement to use them for data structures. If we ever create a data structure of our own, we should implement them too. They are supported for: •Map •Set •Array Plain objects also support similar methods, but the syntax is a bit different. Object.keys, values, entries For plain objects, the following methods are available: • Object.keys(obj) – returns an array of keys. •Object.values(obj) – returns an array of values. •Object.entries(obj) – returns an array of [key, value] pairs. Please note the distinctions (compared to map for example): Map Object Call syntax map.keys() Object.keys(obj), but not obj.keys()	24		

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Returns iterable□“real” Array
		The first difference is that we have to call Object.keys(obj), and not obj.keys(). Why so? The main reason is flexibility. Remember, objects are a base of all complex structures in JavaScript. So we may have an object of our own like data that implements its own data.values() method. And we still can call Object.values(data) on it.
		The second difference is that Object.* methods return “real” array objects, not just an iterable. That’s mainly for historical reasons.
		For instance:
		let user = {
		  name: "John",
		  age: 30
		};
		•□Object.keys(user) = ["name", "age"]
		•□Object.values(user) = ["John", 30]
		•□Object.entries(user) = [["name","John"], ["age",30]]
		Here’s an example of using Object.values to loop over property values:
		let user = {
		  name: "John",
		  age: 30
		};
		
		// loop over			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		values
		for (let value of
		Object.values(
		user)) {
		alert(value); //
		John, then 30
		}
		
		//complete
		array
		console.log(Obj
		ect.entries(use
		r)) ;
		Object.keys/val
		ues/entries
		ignore
		symbolic
		properties
		Just like a
		for..in loop,
		these methods
		ignore
		properties that
		use Symbol(...)
		as keys.
		Usually that’s
		convenient.
		But if we want
		symbolic keys
		too, then
		there’s a
		separate
		method Object.
		getOwnPropert
		ySymbols that
		returns an
		array of only
		symbolic keys.
		Also, there
		exist a method
		Reflect.ownKey
		s(obj) that
		returns all
		keys.
		Transforming
		objects
		Objects lack
		many methods
		that exist for
		arrays, e.g.
		map, filter and
		others.
		If we’d like to
		apply them,
		then we can
		use
		Object.entries
		followed by Obj
		ect.fromEntries
		:
		1.Use Object.
		entries(obj) to
		get an array of
		key/value pairs
		from obj.
		2.Use array
		methods on
		that array, e.g.
		map, to			

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		transform these key/value pairs.
		3.Use Object.fromEntries(array) on the resulting array to turn it back into an object.
		For example, we have an object with prices, and would like to double them:
		let prices = {
		  banana: 1,
		  orange: 2,
		  meat: 4,
		};
		
		let doublePrices =
		 Object.fromEntries(
		  // convert prices to array, map each key/value pair into another pair
		  // and then fromEntries gives back the object
		  Object.entries(prices).map(entry =>
		[entry[0], entry[1] * 2])
		);
		
		alert(doublePrices.meat); // 8
		It may look difficult at first sight, but becomes easy to understand after you use it once or twice.
		We can make powerful chains of transforms this way.
		Tasks
		Sum the properties
		There is a salaries object with arbitrary number of salaries.
		Write the function sumSalaries(salaries) that returns			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		the sum of all salaries using Object.values and the for..of loop.
		If salaries is empty, then the result must be 0.
		For instance:
		let salaries = {
		  "John": 100,
		  "Pete": 300,
		  "Mary": 250
		};
		
		alert(sumSalaries(salaries));
		// 650
		
		solution
		function sumSalaries(salaries)
		{
		
		  let sum = 0;
		  for (let salary of Object.values(salaries)) {
		    sum += salary;
		  }
		
		  return sum; // 650
		}
		
		let salaries = {
		  "John": 100,
		  "Pete": 300,
		  "Mary": 250
		};
		
		alert(sumSalaries(salaries));
		// 650
		Or, optionally, we could also get the sum using Object.values and reduce:
		// reduce loops over array of salaries,
		// adding them up
		// and returns the result
		function sumSalaries(salaries)
		{
		  return Object.values(salaries).reduce((a, b) => a + b, 0) // 650
		}			

Database: lms, Table: lecture_details, Purpose: Dumping data

id	lecture_name	lecture_content	lecture_reference_id	lecture_doc	lecture_video

		Count properties
		Write a function count(obj) that returns the number of properties in the object:
		let user = {
		  name: 'John',
		  age: 30
		};
		
		alert(count(user)); // 2
		Try to make the code as short as possible.
		P.S. Ignore symbolic properties, count only “regular” ones.
		solution
		function count(obj) {
		  return Object.keys(obj).length;
		}			

Database: lms, Table: lecture_exercises, Purpose: Dumping data

id	exercise_name	exercise_content	lecture_id
2	Exercise-1	do this	3
4	sorting of array	sort the following array in ascending order of names arr = ["Karan","Akash","Alli","Sumita","Mehak","Ella"]	20

Database: lms, Table: modules, Purpose: Dumping data

module_id	module_name	duration
PY112	Python	4
JS12	Javascript	4
CC21	Cloud Computing	5
CS07	C#	7
CS08	C++	7

Database: lms, Table: modules_topics, Purpose: Dumping data

module_topic_id	module_topics_name	module_id_id
1	Python For Beginers	PY112
6	Introduction	JS12
4	Python For InterMediate	PY112
5	Python For Advanced	PY112
7	Variables	JS12
8	DataTypes	JS12
9	Type Conversions	JS12
10	Operators	JS12
11	Dialogue Boxes	JS12
12	Comparsions	JS12
13	Conditional Statements	JS12
14	Loops	JS12
15	Functions/Methods	JS12
16	Arrays	JS12
17	Objects	JS12

Database: lms, Table: modules_topics_lectures, Purpose: Dumping data

id	module_lecture_name	module_topics_name_id	module_lecture_duration
1	Lecture-1 Introduction	1	1
4	Lecture-2 Arrays	1	1
3	Lecture-1 (Dictionaries)	4	1
5	Javascript Lecture-1 (Introduction)	6	30
6	Lecture-1 variables	7	30
7	Lecture-1 DataTypes	8	30
8	Lecture-1 Type Conversions	9	30
9	Lecture-1 operators	10	30
10	Lecture-1 Dialogue Boxex	11	30
11	Lecture-1 Comparisions	12	20
12	Lecture-1 Decision Making Statements	13	40
13	Lecture-2 Jumping statements	13	15
14	Lecture-1 Introduction to loops	14	5
15	Lecture-2 For Loop	14	15
16	Lecture-3 While Loop	14	15
17	Lecture-4 Do While Loop	14	15
18	Lecture-1 Functions	15	45
19	Lecture-1 Intro to arrays	16	15
20	Lecture-2 Detailed Array	16	90
21	Lecture-1 Intro to Objects	17	15
22	Lecture-2 Object-This	17	1
23	Lecture-3 Object from Array	17	45
24	Lecture-4 Object Methods	17	45
25	Lecture-5 For in Loop	14	15
26	Lecture-6 JSON-Methods	14	5

Database: lms, Table: module_assesments, Purpose: Dumping data

module_assesment_id	module_assesment_title	module_assesment_description	module_id
1	Basic Test	Debug the following code no = Input(int("Enter Number of students")) Print(no)	PY112
2	Assesment-2	second assesment	PY112

3	Snake Game	Snake Game
		HTML
		
		 Nokia 3310 snake
		
		<button class="top">top</button>
		<button class="bottom">bottom</button>
		<button class="left">left</button>
		<button class="right">right</button>
		
		<button class="playAgain">play Again</button>
		
		CSS
		
		body {
		background: rgb(212, 211, 211);
		}
		.grid {
		width: 200px;
		height: 200px;
		border: 1px solid red;
		margin: 0 auto;
		display: flex;
		flex-wrap: wrap;
		}
		.grid div {
		width: 20px;
		height: 20px;
		/*border:1px black solid;
		box-sizing:border-box*/
		}
		.snake {
		background: blue;
		}
		.apple {
		background: yellow;
		border-radius: 20px;
		}
		.popup {
		background: rgb(32, 31, 31);
		width: 100px;
		height: 100px;
		position: fixed;
		top: 100px;
		left: 100px;
		display: flex;
		justify-content: center;
		align-items: center;
		}
		
		Javascript
		let grid = document.querySelector(".grid")
		let popup = document.querySelector(".popup");
		let playAgain = document.querySelector(".playAgain");
			JS12

Database: lms, Table: module_assesments, Purpose: Dumping data

module_assesment_id	module_assesment_title	module_assesment_description	module_id

		let scoreDisplay = document.querySelector(".scoreDisplay")
		let left = document.querySelector(".left")
		let bottom = document.querySelector(".bottom")
		let right = document.querySelector(".right")
		let up = document.querySelector(".top")
		let width=10;
		let currentIndex = 0
		let appleIndex=0
		let currentSnake=[2,1,0]
		let direction =1
		let score = 0
		let speed = 0.8
		let intervalTime =0
		let interval =0
		
		document.addEventListener("DOMContentLoaded",function(){
		document.addEventListener("keyup",control)
		createBoard()
		startGame()
		playAgain.addEventListener("click", replay);
		})
		
		//createboard function
		function createBoard(){
		popup.style.display = "none";
		for(let i=0;i<100;i++){
		let div =document.createElement("div")
		grid.appendChild(div)
		}
		}
		
		//startgame function
		function startGame(){
		let squares =document.querySelectorAll(".grid div")
		randomApple(squares)
		//random apple
		direction =1
		scoreDisplay.innerHTML=score
		intervalTime=1000
		currentSnake =[2,1,0]
		currentIndex = 0
		currentSnake.forEach(index=>squares[index].classList.add("snake"))
		interval = setInterval(moveOutcome,intervalTime)
		}
		
		function moveOutcome (){
		let squares =document.querySelectorAll(".grid div")
		if(checkForHits(squares)){
		alert("you hit something")
		popup.style.display="flex"
		return clearInterval(interval)
		}else{
		moveSnake(squares)
		}
		}
		
		function moveSnake(squares){
		let tail = currentSnake.pop()
		squares[tail].classList.remove("snake")
		currentSnake.unshift(currentSnake[0]+direction)
		// movement ends here
		eatApple(squares,tail)
		squares[currentSnake[0]].classList.add("snake")
		}
		
		function checkForHits(squares){
		if(
		(currentSnake[0] + width >=(width*width) && direction === width) ||
		(currentSnake[0] % width ===width -1 && direction ===1) ||	

Database: lms, Table: module_assesments, Purpose: Dumping data

module_assesment_id	module_assesment_title	module_assesment_description	module_id

		(currentSnake[0] % width === 0 && direction === -1) ||
		(currentSnake[0] - width <= 0 && direction === -width) ||
		squares[currentSnake[0] + direction].classList.contains("snake")
		){
		return true
		}else{
		return false
		}
		}
		
		function eatApple(squares,tail){
		if(squares[currentSnake[0]].classList.contains("apple")){
		squares[currentSnake[0]].classList.remove("apple")
		squares[tail].classList.add("snake")
		currentSnake.push(tail)
		randomApple(squares)
		score++
		scoreDisplay.textContent = score
		clearInterval(interval)
		intervalTime = intervalTime *speed
		interval = setInterval(moveOutcome,intervalTime)
		}
		}
		
		function randomApple(squares){
		do{
		appleIndex =Math.floor(Math.random() * squares.length)
		}while(squares[appleIndex].classList.contains("snake"))
		squares[appleIndex].classList.add("apple")
		}
		
		function control(e){
		if (e.keycode===39){
		direction = 1 // right
		}else if (e.keycode===38){
		direction = -width //if we press the up arrow, the snake will go ten divs up
		}else if (e.keycode===37){
		direction = -1 // left, the snake will go left one div
		}else if (e.keycode===40){
		direction = +width // down the snake head will instantly appear 10 divs below from
		the current div
		}
		}
		
		up.addEventListener("click",()=>direction= -width)
		bottom.addEventListener("click",()=>direction= +width)
		left.addEventListener("click",()=>direction= -1)
		right.addEventListener("click",()=>direction= 1)
		
		function replay() {
		grid.innerHTML=""
		createBoard()
		startGame()
		popup.style.display = "none";
		}	

Database: lms, Table: students, Purpose: Dumping data

id	name	father_name	email	country_code	phone_number	gender	date_of_birth	enrollment_date	enrollment_time	profile_pic	username	password	source
1	Akashdeep Singh	Gurveer Singh	akash@example.com	+91	7867886567	Male	2002-11-18	2022-11-10	16:21	student-profile/face1.jfif	akash103	123456789	instagram
2	Ashvid Kumar	Rohan Kumar	ashvid@example.com	+91	7879797989	Male	1997-04-06	2022-05-05	16:21	student-profile/face1_j2Ti2Yl.jfif	ashvid	qwertyuiop	instagram

Database: lms, Table: student_assesments, Purpose: Dumping data

id	assesment_id	student_id

Database: lms, Table: student_courses, Purpose: Dumping data

id	course_id_id	student_id_id	batch_id_id
7	9	1	BT01

Database: lms, Table: topic_assignments, Purpose: Dumping data

id	assignment_title	assignment_description	topic_id
1	Assignment-1	Do this assignment	1
2	Assignment-2	Do this assignment	1
3	Assignment-3	Do this assignment	1
4	Clothing E-Commerce Store	Loop Conditional statements Arrays Objects Classes Functions Clothing E-Commerce Store Data Analysis Classes Customers Customerid Name Phone number (mandatory) Email (optional) Country Constructor Products Productid Name Category [Kids, Men , Women] Regular Price 5000 Discount 500 Sale price 4500 Constructor Orders Ordered Ordernumber Orderdate Customer.Customerid (customer object) Product.Productid (product object) Quanity Price Amount Tax Netamount Constructor Analysis 1.Classes define 2.Relationship 3.Data supply – 10 customers , 50 orders 5 products Details Customer – orders Product – orders	16

Database: lms, Table: topic_assignments, Purpose: Dumping data

id	assignment_title	assignment_description	topic_id
		Analysis Country wise sales Product wise sales Customer comparison All results should be displayed in console	

Database: lms, Table: trainers, Purpose: Dumping data

id	trainer_code	name	gender	date_of_birth	country_code	phone_number	email	username	password	profile_pic
2	ISH01	Ishleen Kaur	female	2000-08-14	+91	7234234234	ishleen@example.com	ishleen	enter	trainer-profile/default-profile.jpg
3	KB45	Kabir Behal	male	2000-12-20	+91	7355017830	kabir.behal7830@gmail.com	kabir117	123456789	trainer-profile/default-profile.jpg
7	SUN1	Sunali Kaur	female	2000-08-14	+91	7234234234	sunali@example.com	sunali	123456789	