program counter (line num)	call stack	micro queue	promises	macro queue	event reg	console output
5	[main()]	[]	{promise@5}	[]	{}	
8	[main()]	[]	{promise@5, promise@8}	[]	{}	
22	[main()]	[]	{promise@5, promise@8}	[]	{ ev1.run:function@22 }	
26	[main()]	[]	{promise@5, promise@8}	[]	{ ev1.run:function@22, ev2.run:function@26 }	
30	[main()]	[]	{promise@5, promise@8}	[function@26(0)]	{ ev1.run:function@22, ev2.run:function@26 }	
eof	[]	[]	{promise@5, promise@8}	[function@26(0)]	{ ev1.run:function@22, ev2.run:function@26 }	
26	[function@26(0)]	[]	{promise@5, promise@8}	[]	{ ev1.run:function@22, ev2.run:function@26 }	
27	[function@26(0)]	[]	{promise@5, promise@8}	[]	{ ev1.run:function@22, ev2.run:function@26 }	data 0 received by ev2
28	[function@26(0)]	[promise@5.resolve(0) = (0) => foo(0)]	{promise@5, promise@8}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2"
eof	[]		{promise@5, promise@8}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2"
28	[promise@5.resolve(0)=(0) => foo(0), foo(0)]	[]	{promise@5, promise@8}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2"
11	[promise@5.resolve(0)=(0) => foo(0), foo(0)]	[]	{promise@5, promise@8}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2"
12	[promise@5.resolve(0)=(0) => foo(0), foo(0)]	[]	{promise@5, promise@8, promise@12}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2"
16	[promise@5.resolve(0)=(0) => foo(0), foo(0)]	[]	{promise@5, promise@8, promise@12}	[function@22[0]]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2"
28	[promise@5.resolve(0)=(0) => foo(0), foo(0)]	[]	{promise@5, promise@8, promise@12}	[function@22[0]]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2"
eof	[]	[]	{promise@5, promise@8, promise@12}	[function@22[0]]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2"
22	[function@22(1)]	[]	{promise@5, promise@8, promise@12}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2"
23	[function@22(1)]	[]	{promise@5, promise@8, promise@12}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2", "data 1 received by ev1"
24	[function@22(1)]	[promise@8.resolve(0)=(0)=> foo(1)]	{promise@5, promise@8, promise@12}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2", "data 1 received by ev1"
eof	[]	[promise@8.resolve(0)=(0)=> foo(1)]	{promise@5, promise@8, promise@12}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2", "data 1 received by ev1"
24	[promise@8.resolve(0)=(0)=> foo(1)]	[]	{promise@5, promise@8, promise@12}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2", "data 1 received by ev1"
11	[promise@8.resolve(0)=(0)=> foo(1)]	[]	{promise@5, promise@8, promise@12}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2", "data 1 received by ev1"
12	[promise@8.resolve(0)=(0)=> foo(1)]	[]	{promise@5, promise@8, promise@12, promise@12}	[]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2", "data 1 received by ev1"
18	[promise@8.resolve(0)=(0)=> foo(1)]	[]	{promise@5, promise@8, promise@12, promise@12}	[function@28[0]]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2", "data 1 received by ev1"
24	[promise@8.resolve(0)=(0)=> foo(1)]	[]	{promise@5, promise@8, promise@12, promise@12}	[function@28[0]]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2", "data 1 received by ev1"
eof	[]	[]	{promise@5, promise@8, promise@12, promise@12}	[function@28[0]]	{ ev1.run:function@22, ev2.run:function@26 }	"data 0 received by ev2", "data 1 received by ev1"

the function repeats until we print "data 11 received by ev1"

The execution demonstrates how the events, promises, and callback functions interplay in a JavaScript environment. Each emit leads to the registration of new tasks in the macrotask queue, eventually resolving promises and triggering the next event, creating a cycle until the stopping condition is met. The console output reflects these transitions, logging data reception by ev1 and ev2 until x exceeds 10. Microtask queue is used to handle tasks that need to be executed as soon as possible, update state or handle promise resolution and promises are always processed in the microtask queue. The macrotask queue has larger, less urgen tasks to settle such as I/O operations, timers and user interactions, hence handling the functions in the program.