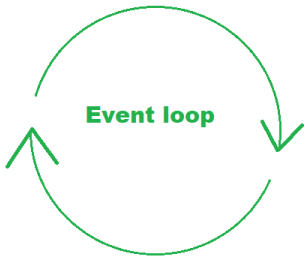
Event Looping

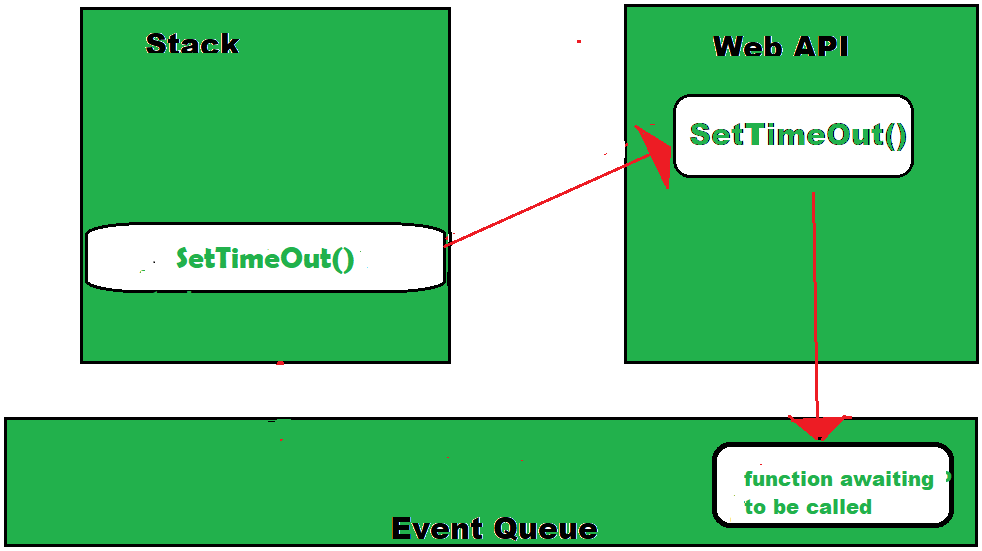
Introduction

Event loop is responsible for executing the code, collecting and processing events, and executing queued sub-tasks.

Event loop is something that pulls stuff out of the queue and places it onto the function execution stack whenever the function stack becomes empty.



Event looping looks like



let's say we have an example to understand the event loop concepts.

```
function foo(b) {
  const a = 10;
  return a + b + 11;
}

function bar(x) {
  const y = 3;
  return foo(x * y);
}

const baz = bar(7);
```

The flow of above code will be

1. When calling **bar,** first frame is created containing reference to **bar's** argument and local variables.
2. When **bar** calls **foo**, a second frame is created and pushed on the top of the first one, containing references to **foo's** arguments and local variables.

3. When **foo** returns, the top frame element is popped out of the stack(leaving only **bar's** call frame.
4. When **bar** returns, the stack is empty.

Heap

Objects are allocated in heap which is just a name to denote a large(mostly unstructured) region of memory.

Queue

A JavaScript runtime uses a message queue, which is a list of messages to be processed. Each message has an associated function that gets called to handle the message.

Pros

A very interesting property of event loop model is that JS, unlike a lot of other languages, never blocks. Handling I/O us typically performed via events and callback, so when the application is waiting for an indexedDB query to return or an XHR request to return, it can still process other things like user input.