PHASE 1

WEEK 2

DAY 3

ELBRUS/BOOTCAMP

План

1. Асинхронность (asynchrony)
   1.1. Таймеры — setTimeout, setInterval (timers)
   1.2. Event Loop
   1.3. Асинхронные методы модуля FS
2. Rest-параметр, объект arguments*

Таймеры

setTimeout

Откладывает вызов callback-функции как минимум на указанное количество миллисекунд.

```
function printText(message) {
 console.log(message);
}

setTimeout(printText, 1000, "До скорой встречи");
```

setInterval

Ставит вызов callback-функции на повтор — каждое повторение как минимум через указанное количество миллисекунд.

```
function printText(message) {
 console.log(message);
}

setInterval(printText, 1000, "И снова здравствуй");
```

Очистка таймера

`clearTimeout`, `clearInterval` — функции для отмены запланированного таймаута или интервала.

```
const id = setInterval(printText, 1000);

clearInterval(id);
```

Очистка таймера: применение

```
const people = ["Василий", "Алёна", "Максим", "Филат"];
const intervalId = setInterval(playLottery, 1000, people);

function playLottery(players) {
 const i = Math.floor(Math.random() * players.length);
 console.log(`Победитель лотереи: ${players[i]}`);
}

function stopLottery(id) {
 clearInterval(id);
}
```

Callback

Callback – функция обратного вызова. Функция, которая передана в другую функцию.

Что делать с callback-функциями?

- передавать их в параметрах, но не вызывать.
- передаём callback-функции, чтобы они были вызваны позже.

Пример callback-функции

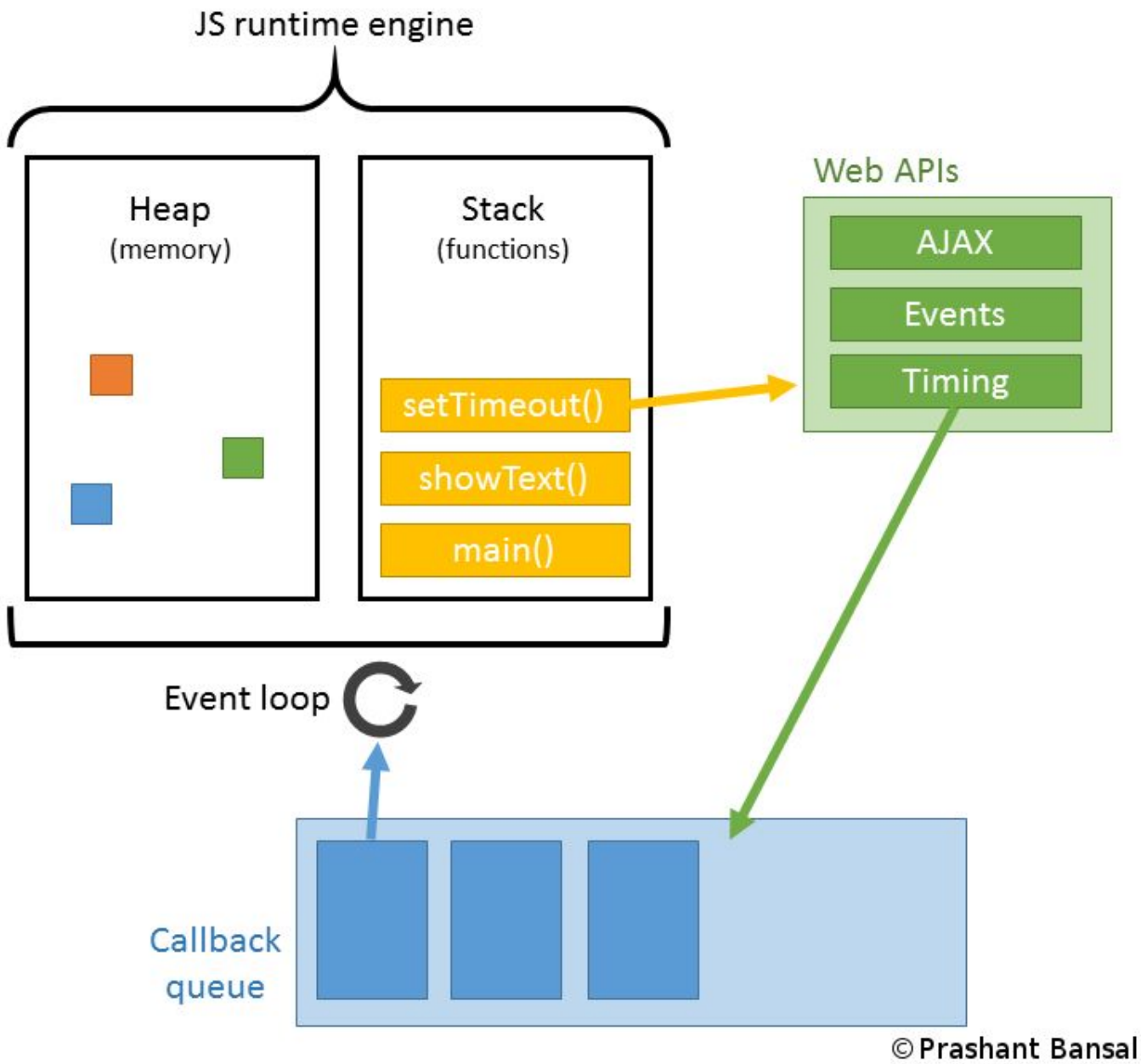
```
const fs = require('fs')

fs.readFile('./data/users.txt', 'utf-8', (err, data) => {
 // обработка ошибки (в Node.js обычно это первый параметр)
 if (err) throw new Error('File not found!')

 // получаем данные из файла, с помощью callback
 console.log(data);
})
```

Event loop

цикл событий


JS runtime engine
Heap
(memory)
Stack
(functions)
setTimeout()
showText()
main()
Web APIs
AJAX
Events
Timing
Event loop
Callback
queue
© Prashant Bansal

Пример

```
// В каком порядке сработают логи?

console.log("Привет");

setTimeout(function() {
 console.log("Ты мне нравишься");
}, 5000);

console.log("JavaScript");
```

Пример посложнее

```
console.log('Step 1');

setTimeout(() => {
  console.log('Step 2');
}, 0);

console.log('Step 3');

setTimeout(() => {
  console.log('Step 4');
}, 4000);

setTimeout(() => {
  console.log('Step 5');
}, 0);
```

Визуализатор работы Event loop

Полезные ссылки:

- http://latentflip.com/loupe/
- https://www.jsv9000.app/
- https://dev.to/lydiahallie/javascript-visualized-event-loop-3dif

Асинхронные методы FS

Некорректная запись в файл

```
const fs = require("fs");

fs.writeFile(`file.txt`, `1\n`, () => {});
fs.appendFile(`file.txt`, `2\n`, () => {});
fs.appendFile(`file.txt`, `3\n`, () => {});
fs.appendFile(`file.txt`, `Ой...\n`, () => {});
```

Корректная запись в файл

```
const fs = require("fs");

// добро пожаловать в ад callback-функций :)
fs.writeFile(`file.txt`, `1\n`, () => {
 fs.appendFile(`file.txt`, `2\n`, () => {
   fs.appendFile(`file.txt`, `3\n`, () => {
     fs.appendFile(`file.txt`, `4\n`, () => {
     });
   });
 });
});
```

Чтение нескольких файлов

```
const fs = require("fs");

/* Как вывести содержимое файлов в нужном порядке,
читая асинхронно и не попадая в callback hell? */
for (let i = 0; i < 3; i += 1) {
 fs.readFile(`./files/${i + 1}.txt`, "utf8", (err, data) => {
   console.log(data);
 });
}
```

“Синхронизация” callback-функций

```
const fs = require("fs");

const fileCount = 3;
const content = [];
let callbackCount = 0;

for (let i = 0; i < fileCount; i += 1) {
  fs.readFile(`./files/${i + 1}.txt`, "utf8", (err, data) => {
    // замыкание — callback-функция имеет доступ к значению i на момент своего создания
    content[i] = data;
    callbackCount += 1;

    if (callbackCount === fileCount) { // если этот callback последний
      console.log(content.join("\n")); // ...вывести содержимое
    }
  });
}
```

Rest-параметры, объект arguments

Больше аргументов, чем ждали

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

sum(1, 2); // 3
sum(1, 2, 3, 4, 5); // будет ли ошибка?
```

Объект arguments

```
function sum(a, b) {
  /* arguments - объект со всеми аргументами (реальными значениями
  переданными в параметры). Доступен в любой функции, но лучше не
  использовать — предпочтителен синтаксис rest-параметра. */
  // return arguments;
}

sum(1, 2); // { '0': 1, '1': 2 }
sum(1, 2, 3, 4, 5); // { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5 }
```

Rest-параметр (всё, что осталось)

```
function sum(a, b, ...args) {
  /* args — массив всех аргументов, переданных после
  именованных параметров. */
  return args;
}

sum(1, 2); // []
sum(1, 2, 3, 4, 5); // [3, 4, 5]
```