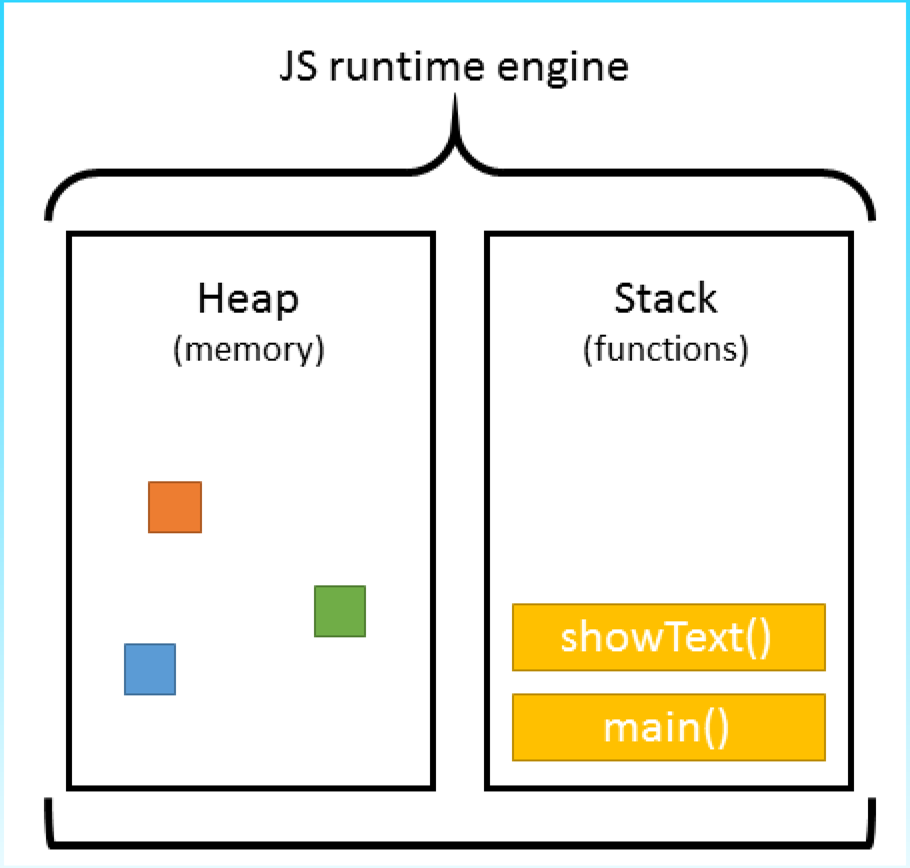
Обещание

Talk about:

1. Event loop
2. Callback, async code
3. Promise

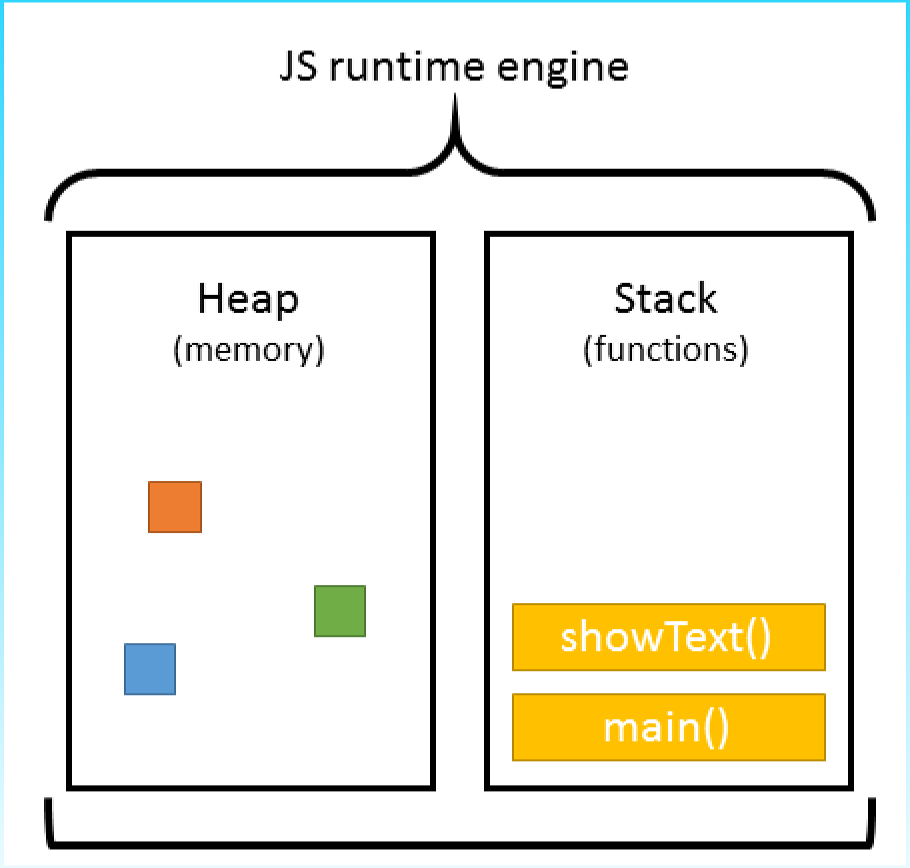
А начиналось всё так...
DATE
2010
SCENE
9
TAKE
2

Стек, Куча



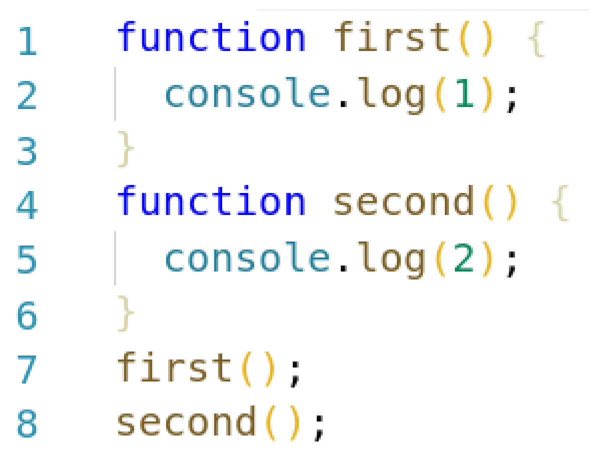
Куча (Heap) - это ссылка на определённую неструктурированную область памяти, в которой размещаются объекты

Стек, Куча



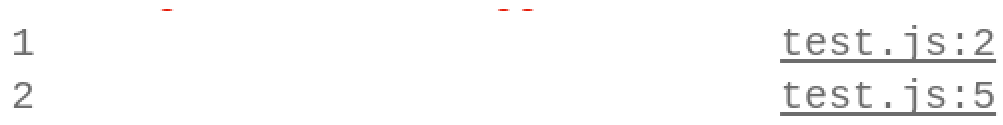
Куча (Heap) - это ссылка на определённую неструктурированную область памяти, в которой размещаются объекты

Стек (Steak) – структура данных, представляющая из себя список контекстов организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).



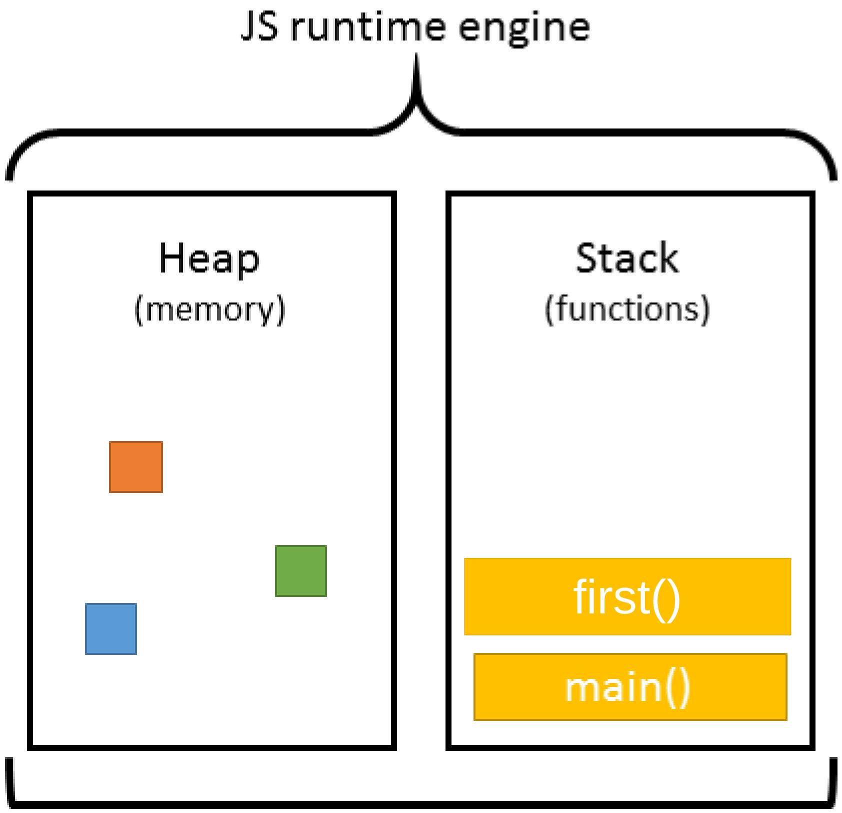
```
1  function first() {
2    console.log(1);
3  }
4  function second() {
5    console.log(2);
6  }
7  first();
8  second();
```

Console:

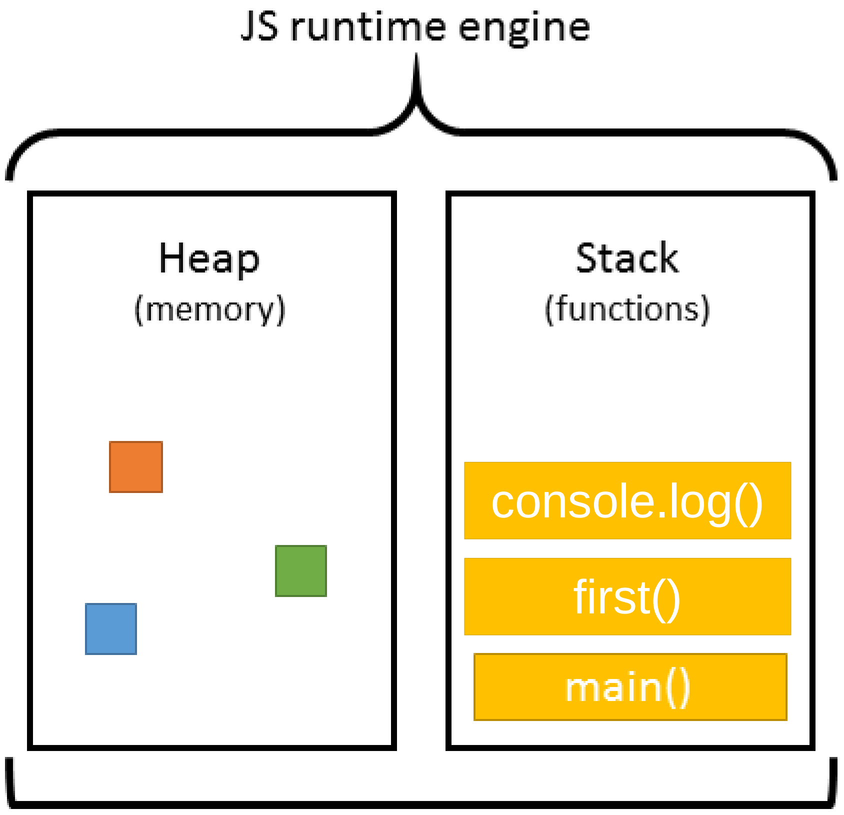


```
1                                   test.js:2
2                                   test.js:5
```

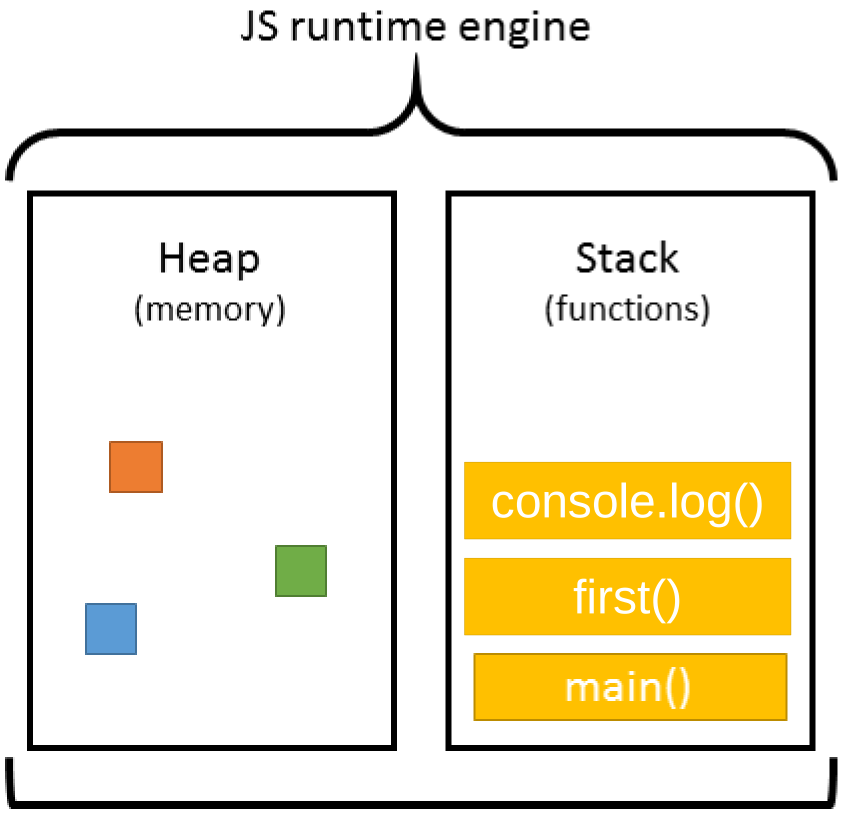
```
function first() {
  console.log(1);
}
function second() {
  console.log(2);
}
first();
second();
```



```
function first() {
  console.log(1);
}
function second() {
  console.log(2);
}
first();
second();
```



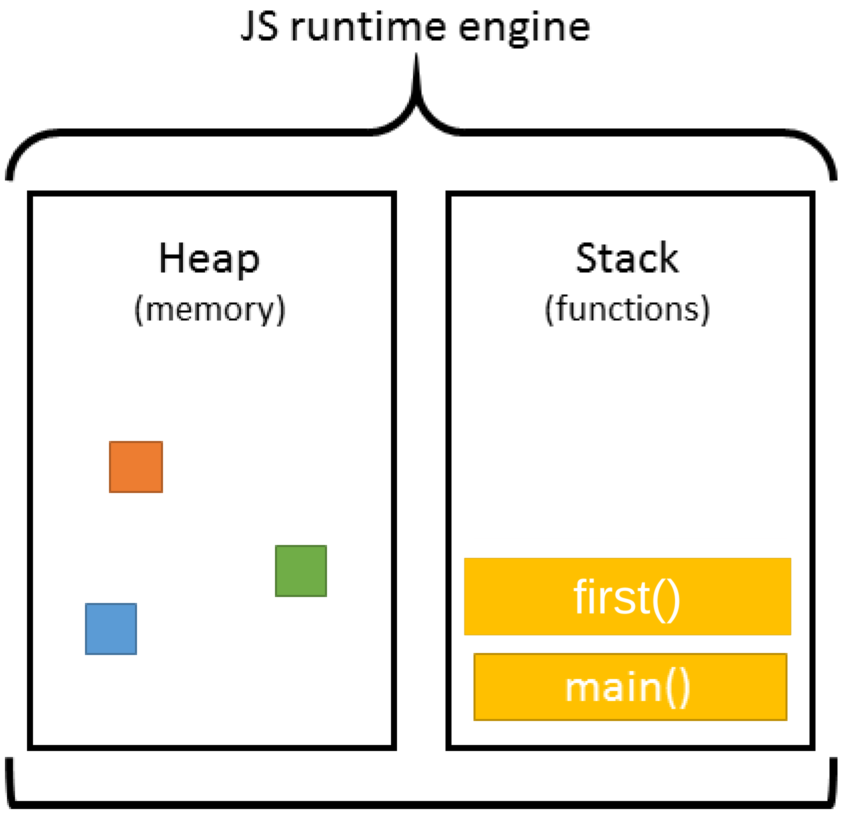
```
function first() {
  console.log(1);
}
function second() {
  console.log(2);
}
first();
second();
```



Console:

```
1                                   test.js:2
```

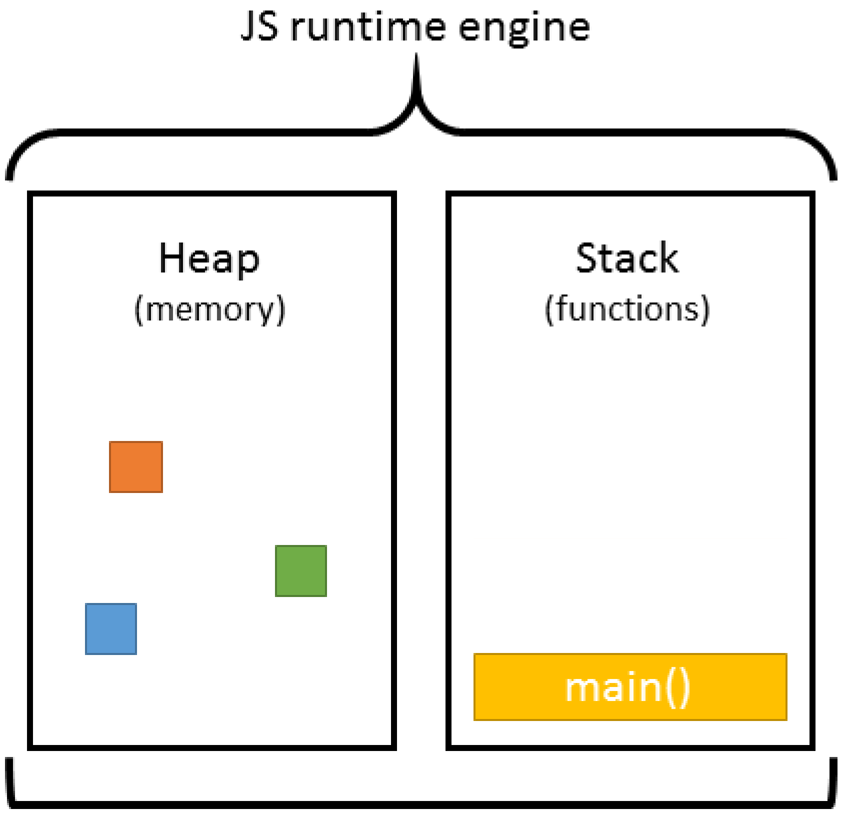
```javascript
function first() {
  console.log(1);
}
function second() {
  console.log(2);
}
first();
second();
```



Console:

```
1                                    test.js:2
```

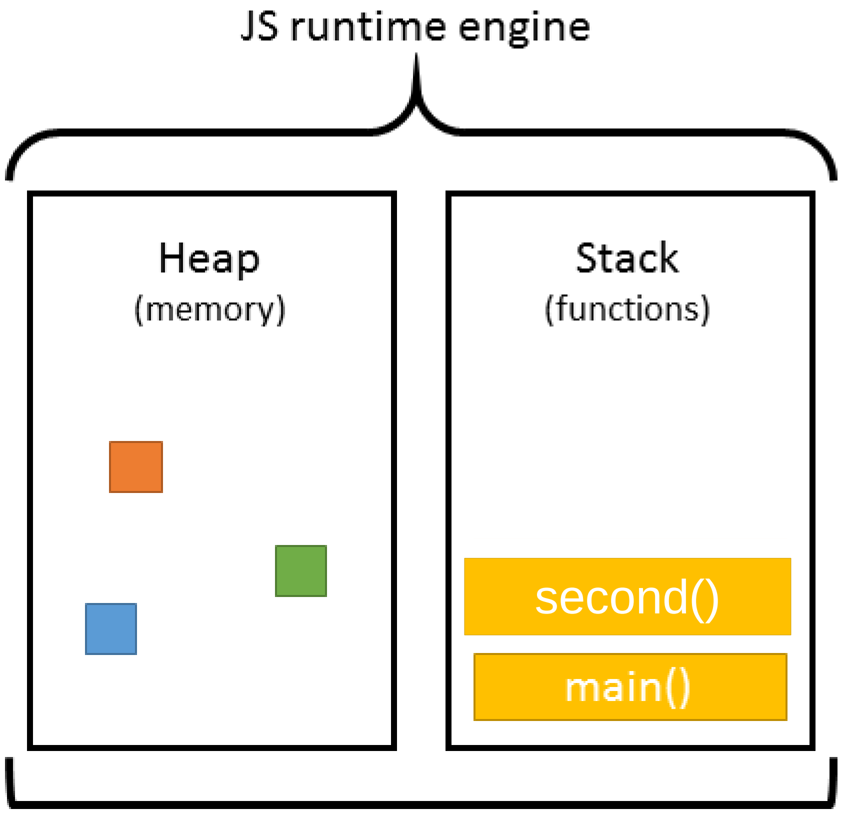
```
function first() {
  console.log(1);
}
function second() {
  console.log(2);
}
first();
second();
```



Console:

```
1                                    test.js:2
```

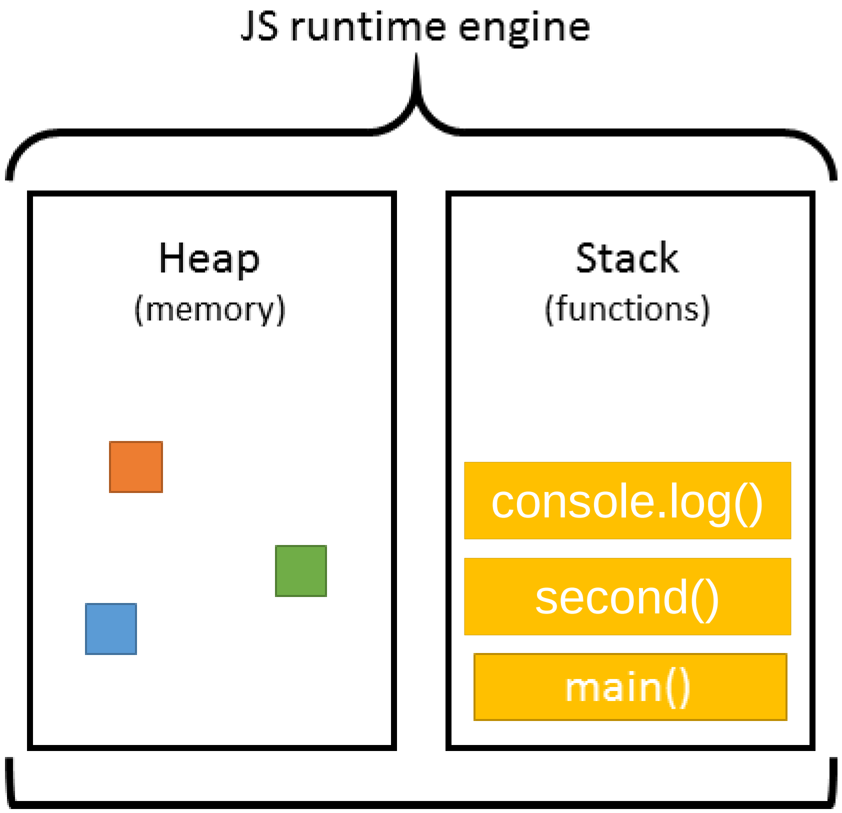
```javascript
function first() {
  console.log(1);
}
function second() {
  console.log(2);
}
first();
second();
```



Console:

```
1                                              test.js:2
```

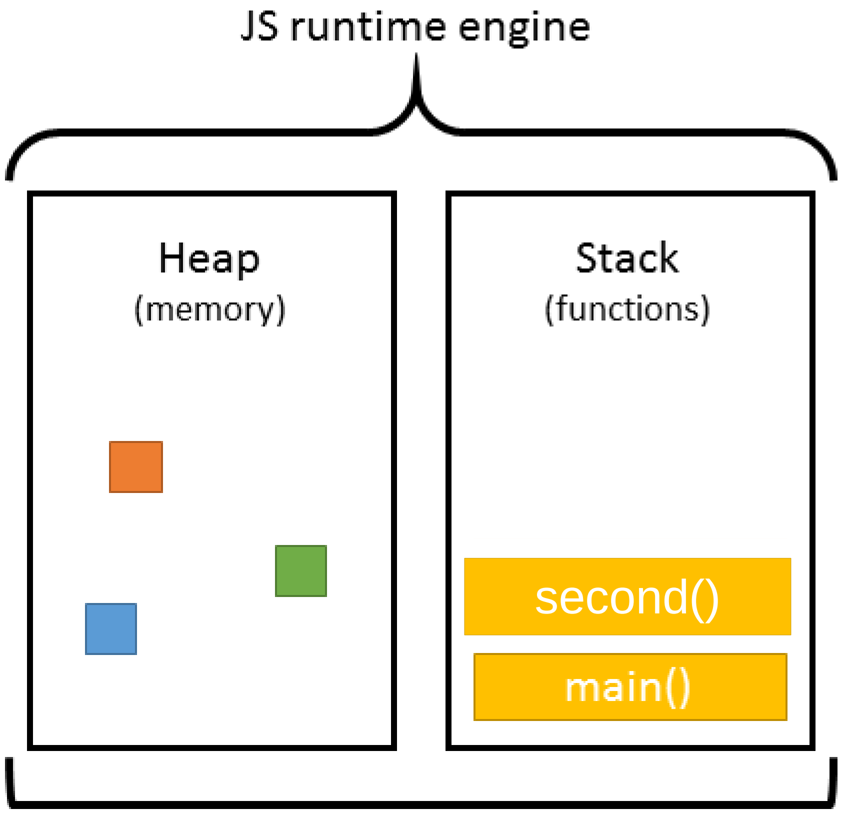
```
1  function first() {
2    console.log(1);
3  }
4  function second() {
5    console.log(2);
6  }
7  first();
8  second();
```



Console:

```
1                     test.js:2
2                     test.js:5
```

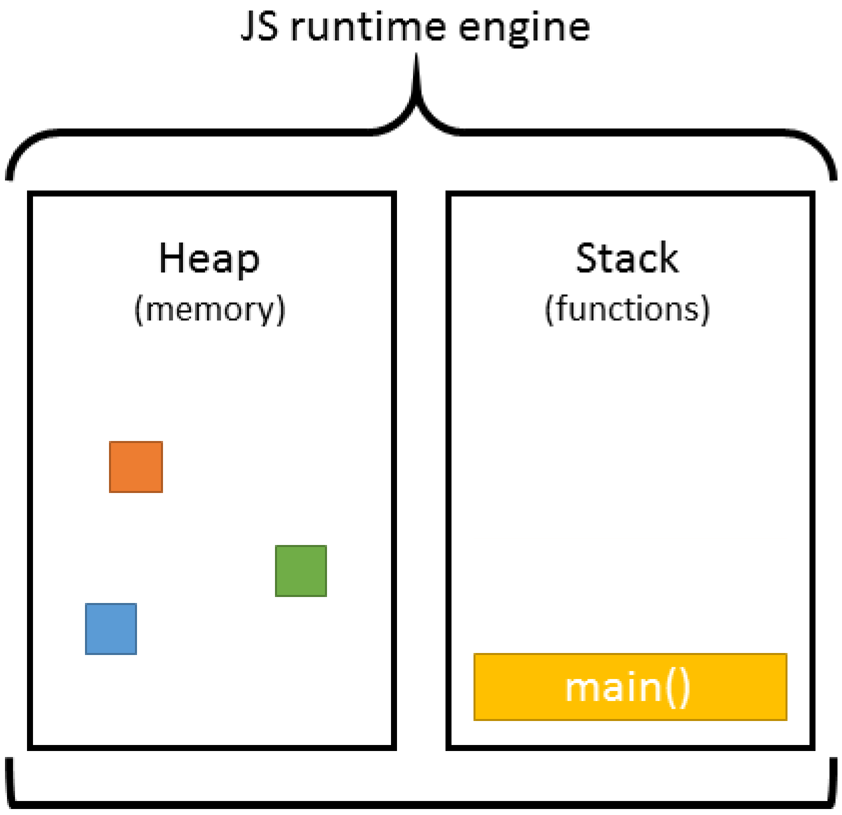
```
1  function first() {
2    console.log(1);
3  }
4  function second() {
5    console.log(2);
6  }
7  first();
8  second();
```



Console:

```
1                         test.js:2
2                         test.js:5
```

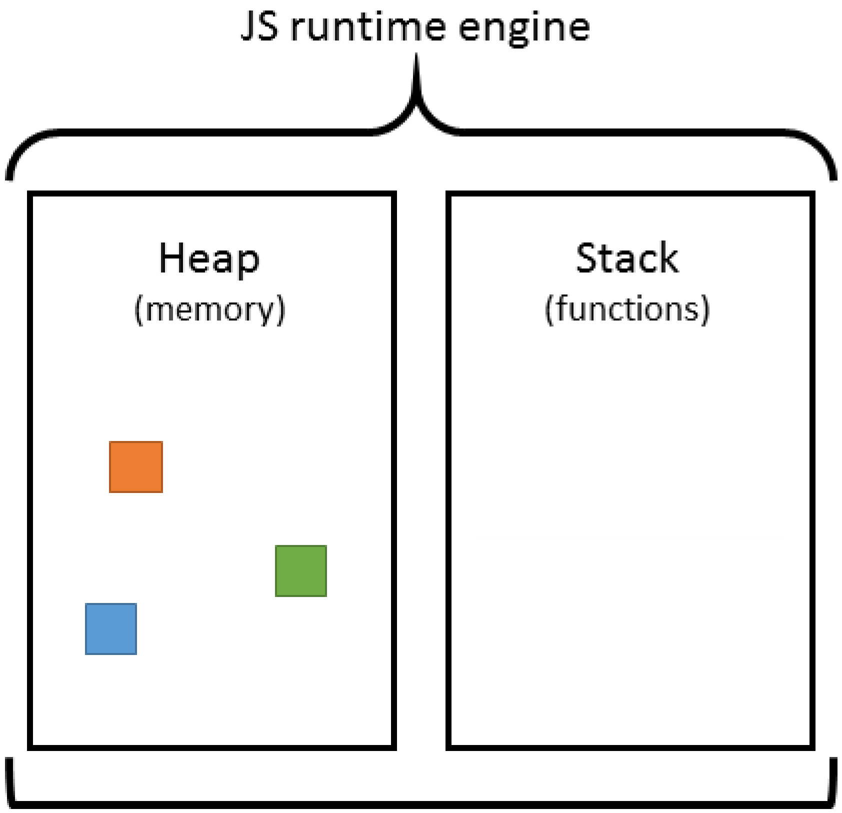
```javascript
function first() {
  console.log(1);
}
function second() {
  console.log(2);
}
first();
second();
```



Console:

```
1                                 test.js:2
2                                 test.js:5
```

```
function first() {
  console.log(1);
}
function second() {
  console.log(2);
}
first();
second();
```



Console:

```
1                                   test.js:2
2                                   test.js:5
```

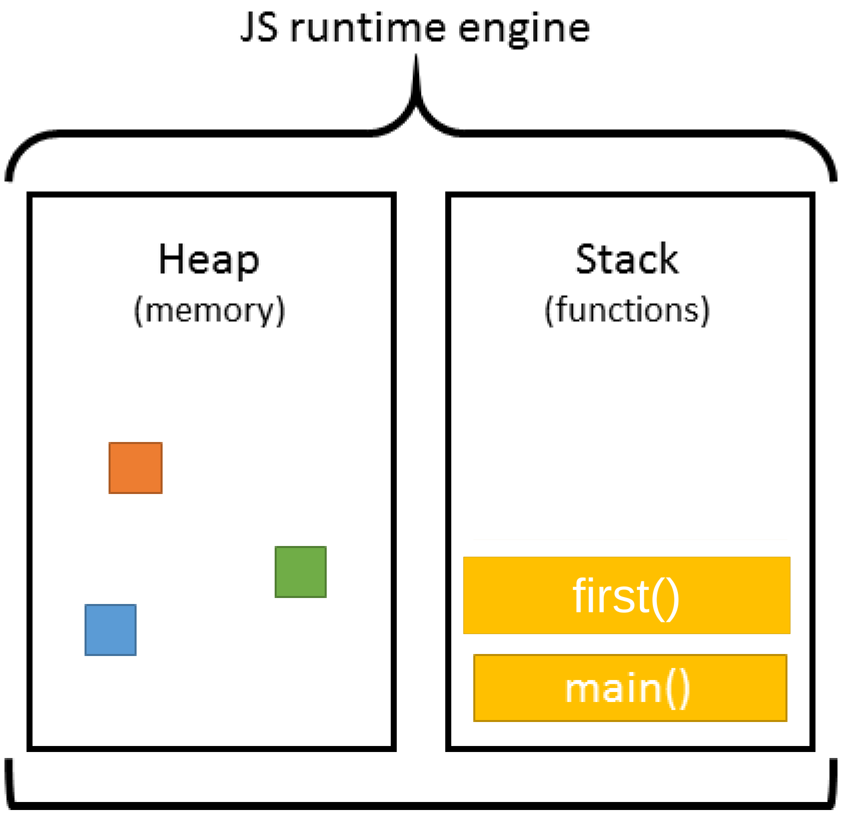
```javascript
1  function first() {
2    console.log(1);
3  }
4  function second() {
5    console.log(2);
6  }
7  first();
8  second();
```

```javascript
function first() {
  console.log(1);
  for (var i = 0; i < 100000000; i++) {
    Math.pow(1e10000, i);
  }
}
function second() {
  console.log(2);
}
first();
second();
```

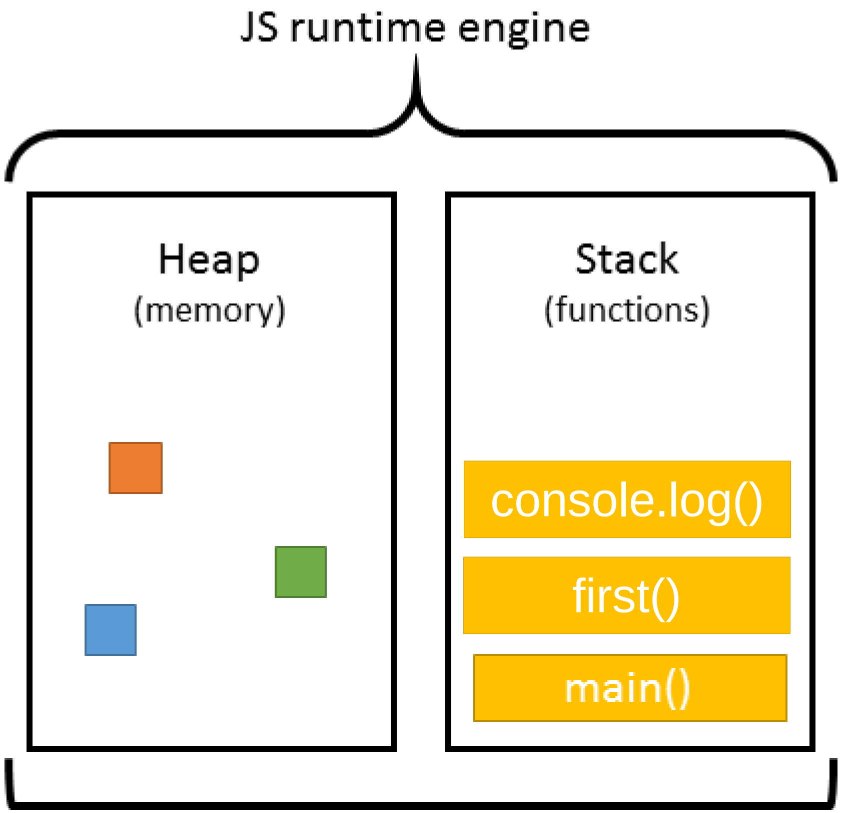
Console:

```
1                                    test.js:2
2                                    test.js:5
```

```
function first() {
  console.log(1);
  for (var i = 0; i < 100000000; i++) {
    Math.pow(1e10000, i);
  }
}
function second() {
  console.log(2);
}
first();
second();
```



```
1   function first() {
2     console.log(1);
3     for (var i = 0; i < 100000000; i++) {
4       Math.pow(1e10000, i);
5     }
6   }
7   function second() {
8     console.log(2);
9   }
10  first();
11  second();
```



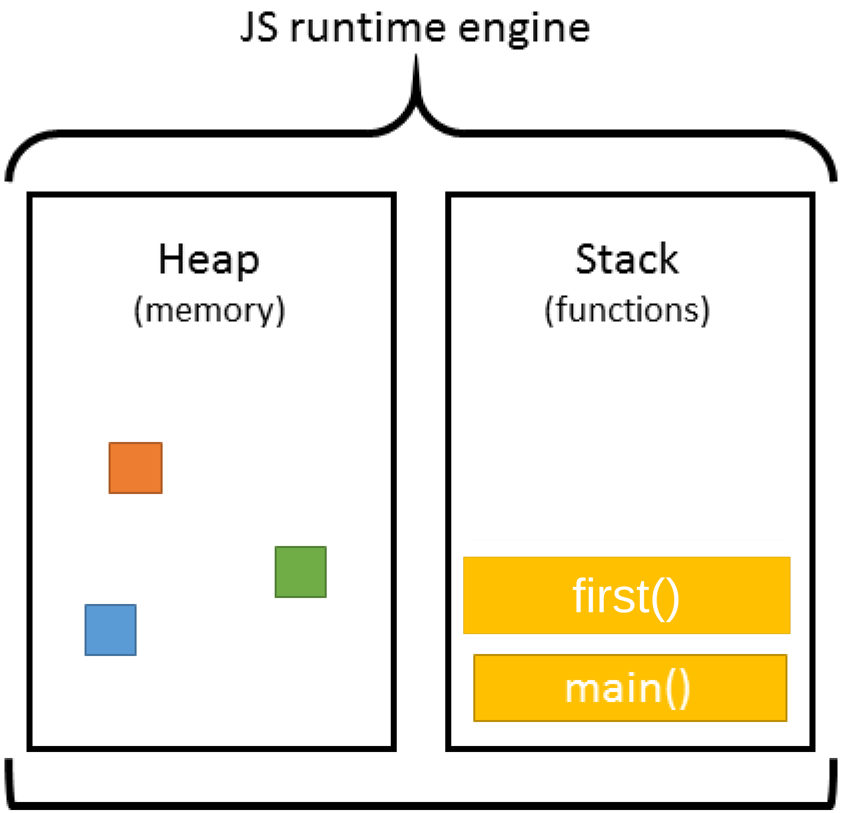
Console:

```
1                                   test.js:2
```

```
function first() {
  console.log(1);
  for (var i = 0; i < 100000000; i++) {
    Math.pow(1e10000, i);
  }
}
function second() {
  console.log(2);
}
first();
second();
```

Console:

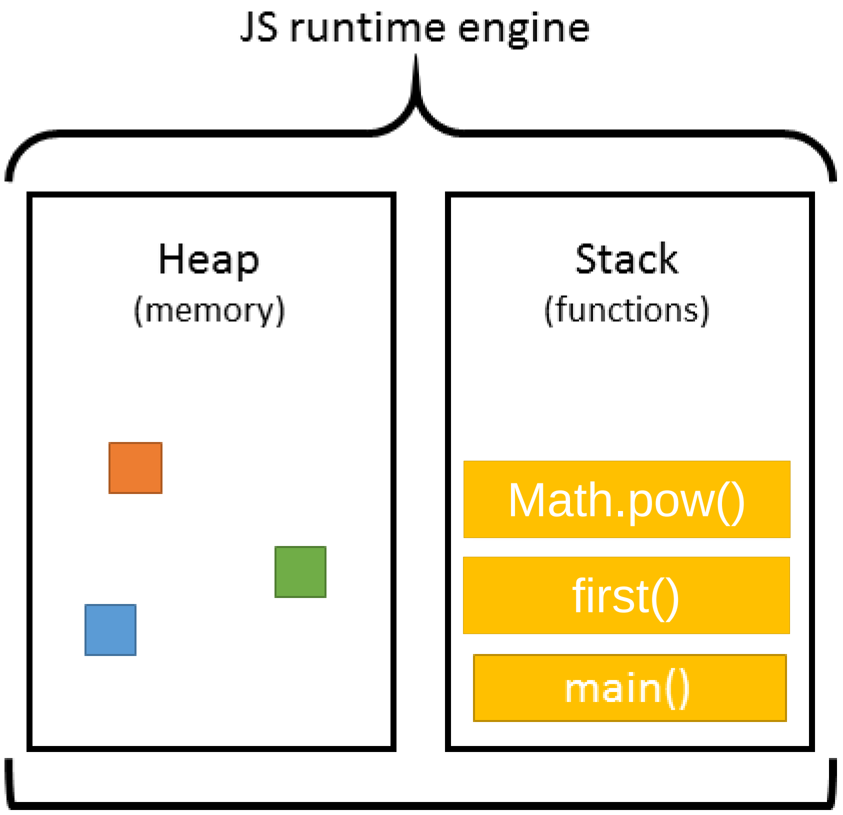
```
1                         test.js:2
```

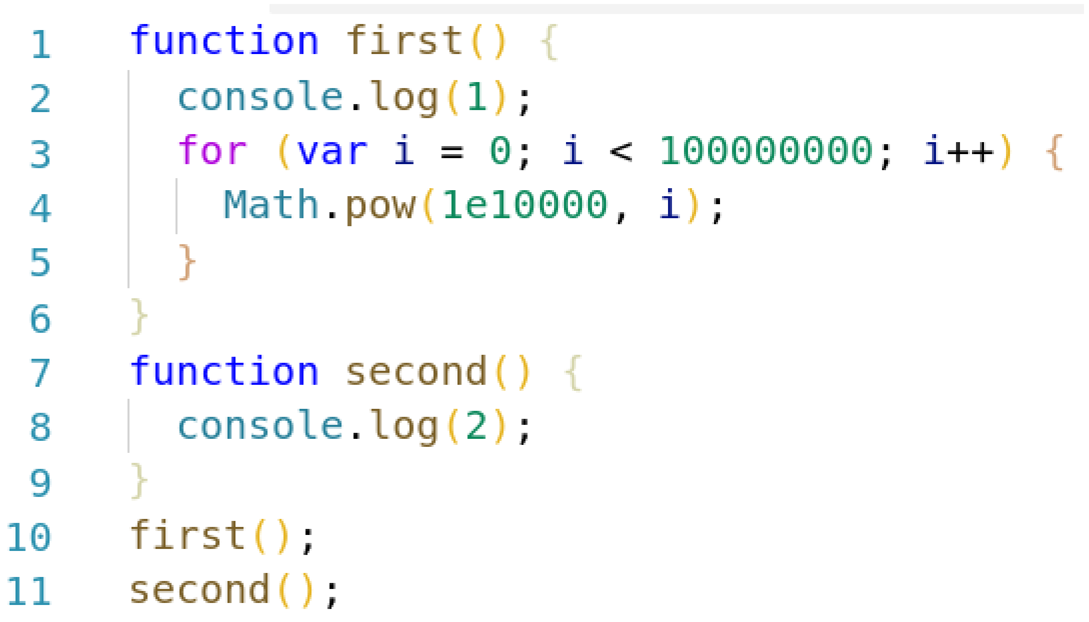


```javascript
function first() {
  console.log(1);
  for (var i = 0; i < 100000000; i++) {
    Math.pow(1e10000, i);
  }
}
function second() {
  console.log(2);
}
first();
second();
```

Console:

```
1                                   test.js:2
```

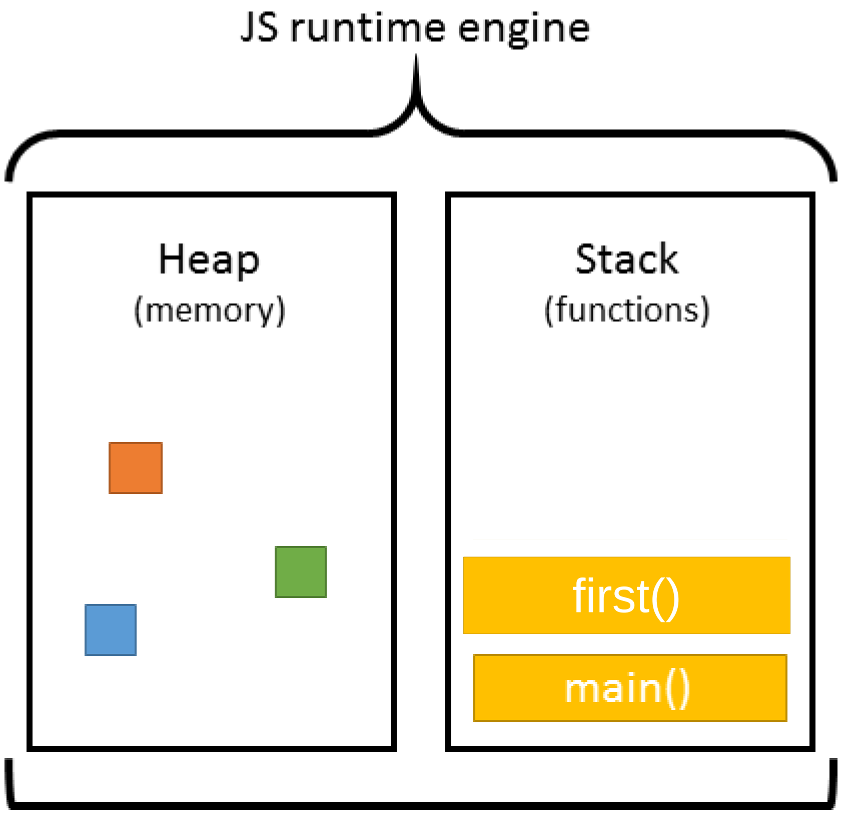




```
function first() {
  console.log(1);
  for (var i = 0; i < 100000000; i++) {
    Math.pow(1e10000, i);
  }
}
function second() {
  console.log(2);
}
first();
second();
```

Console:

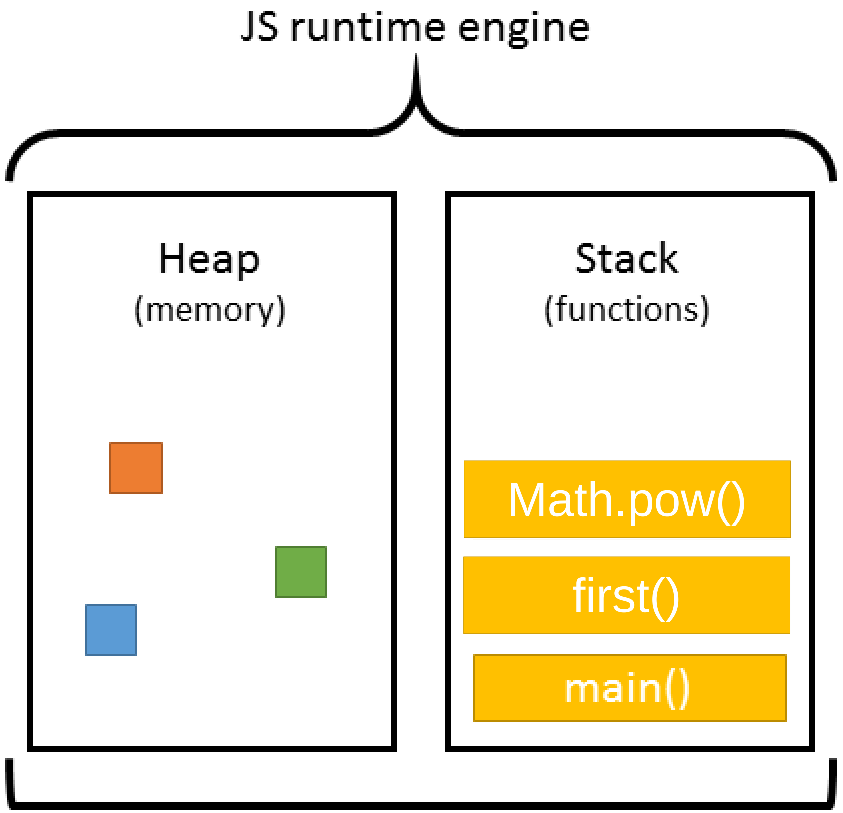
```
1                                    test.js:2
```



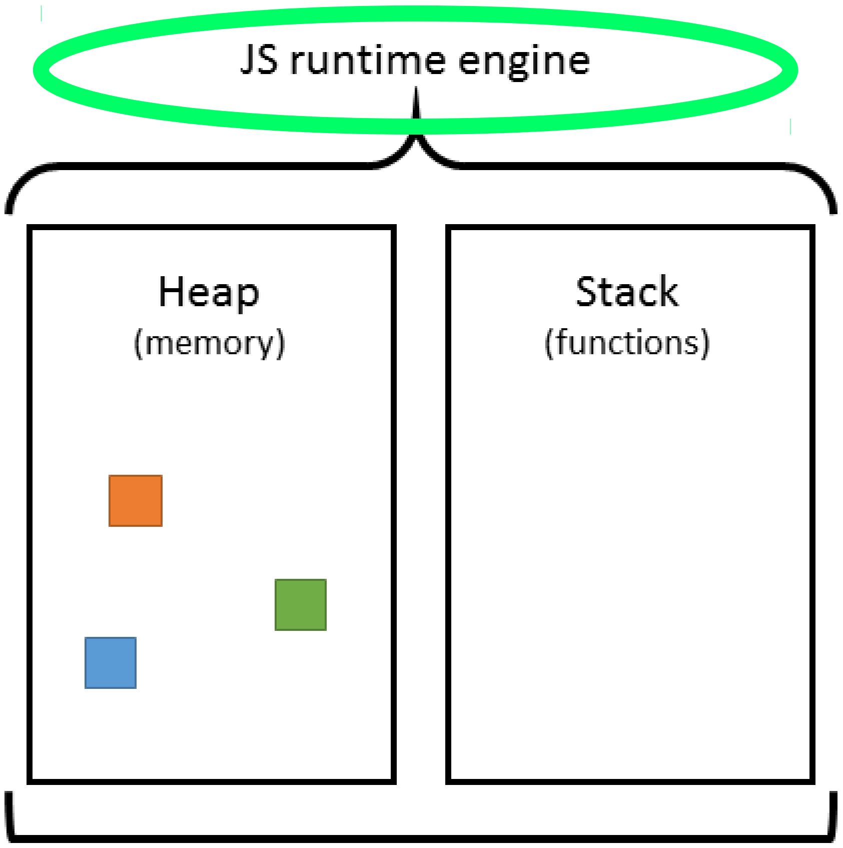
```
function first() {
  console.log(1);
  for (var i = 0; i < 100000000; i++) {
    Math.pow(1e10000, i);
  }
}
function second() {
  console.log(2);
}
first();
second();
```

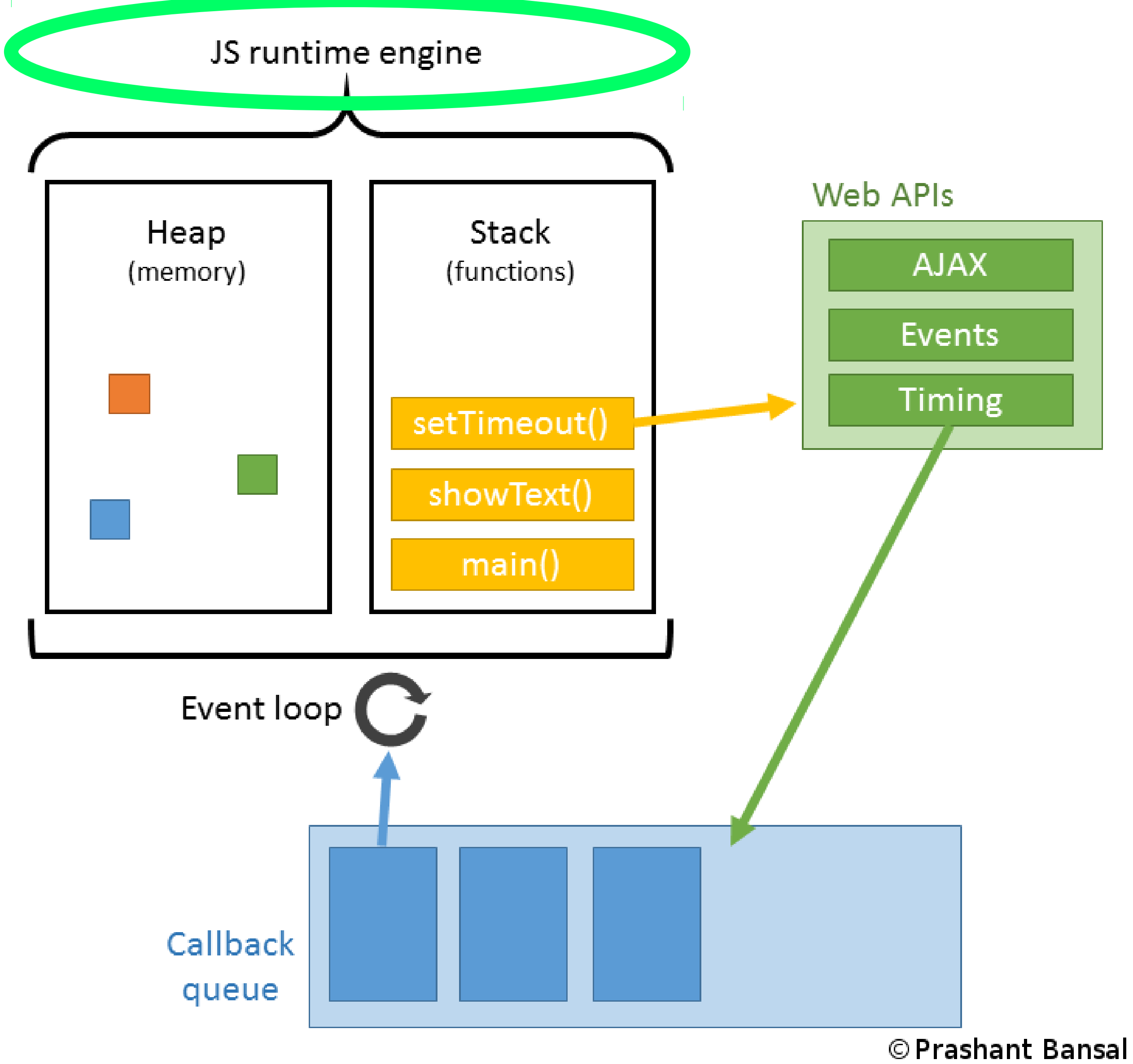
Console:

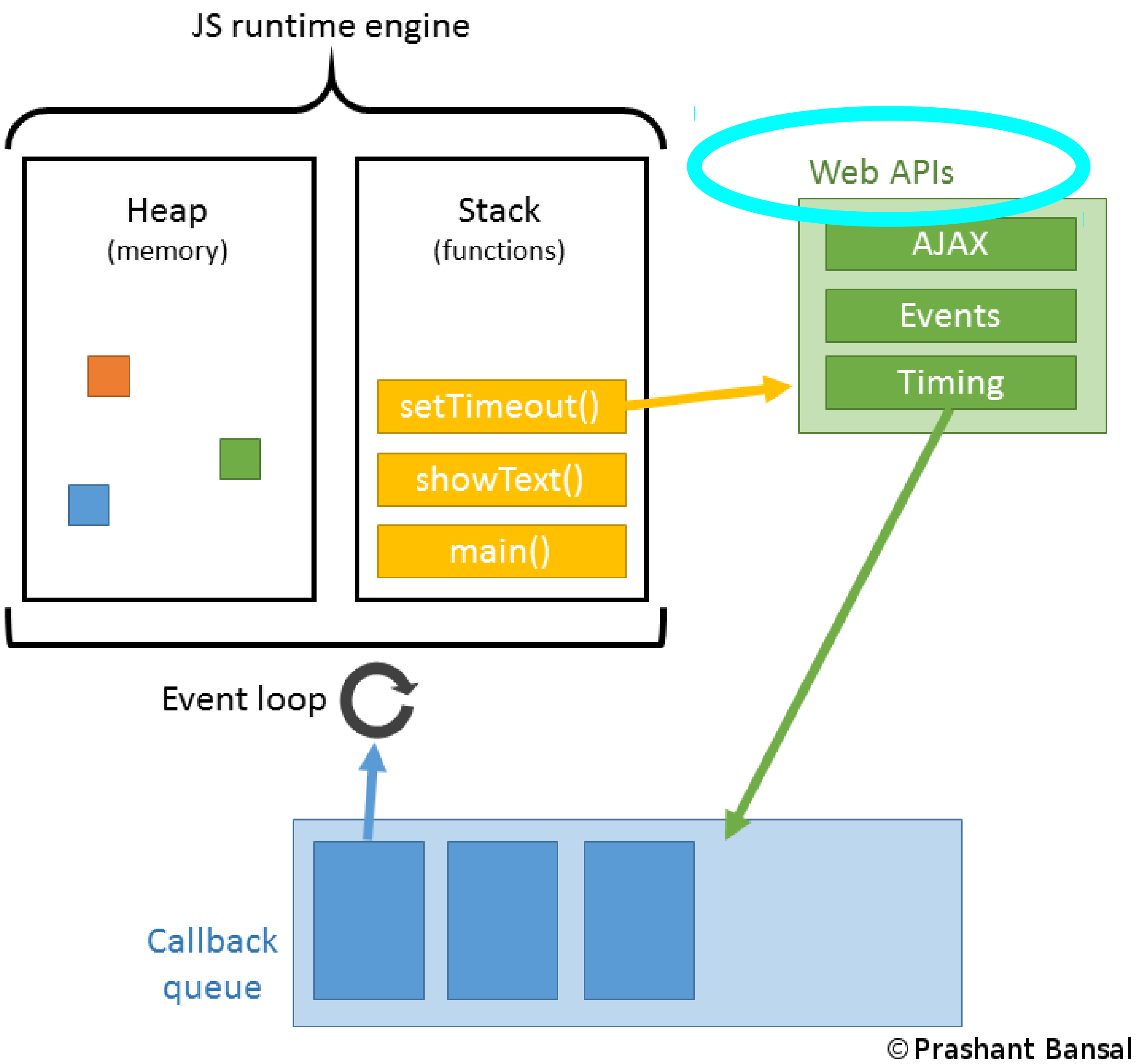
```
1                                   test.js:2
```

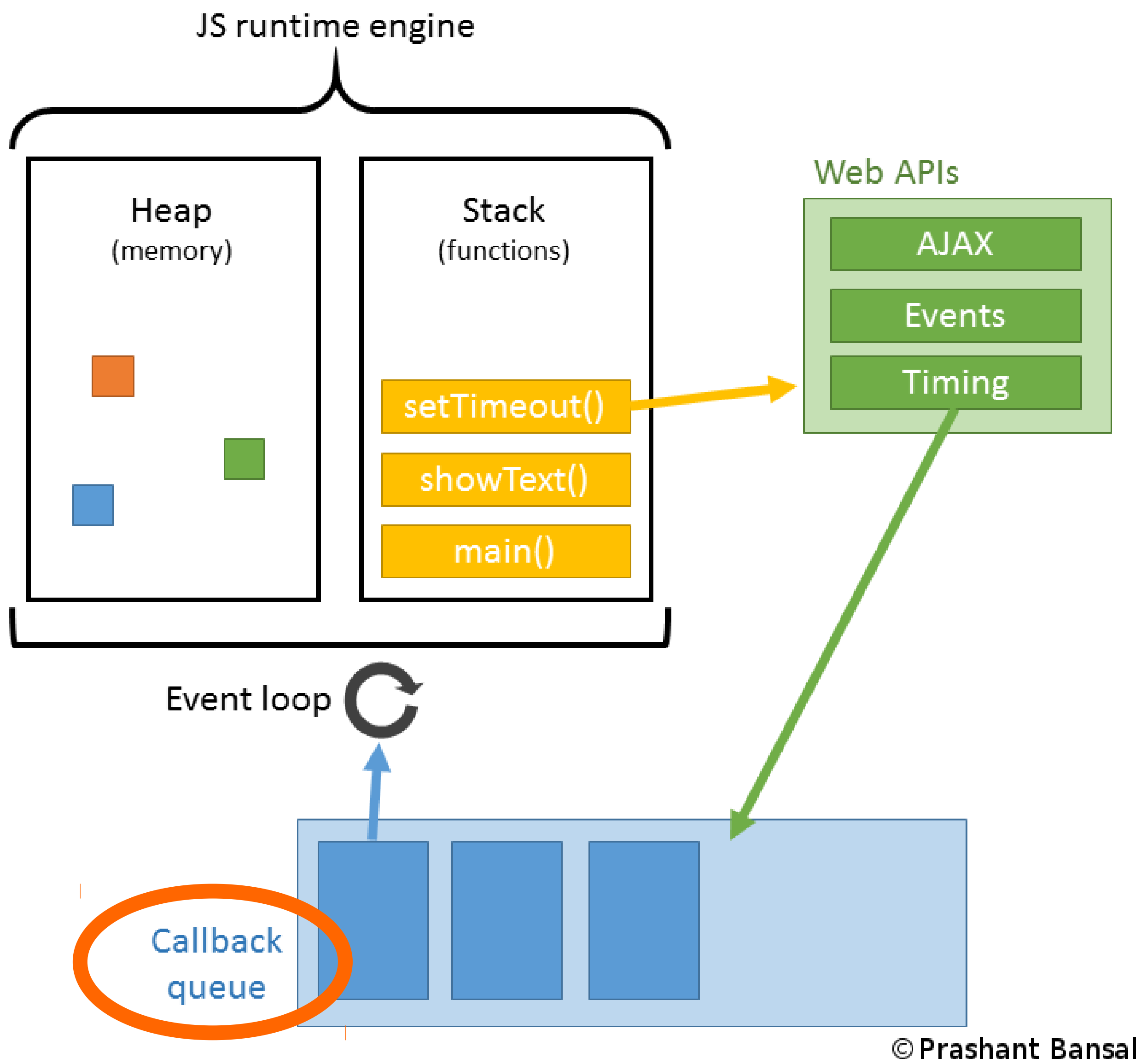


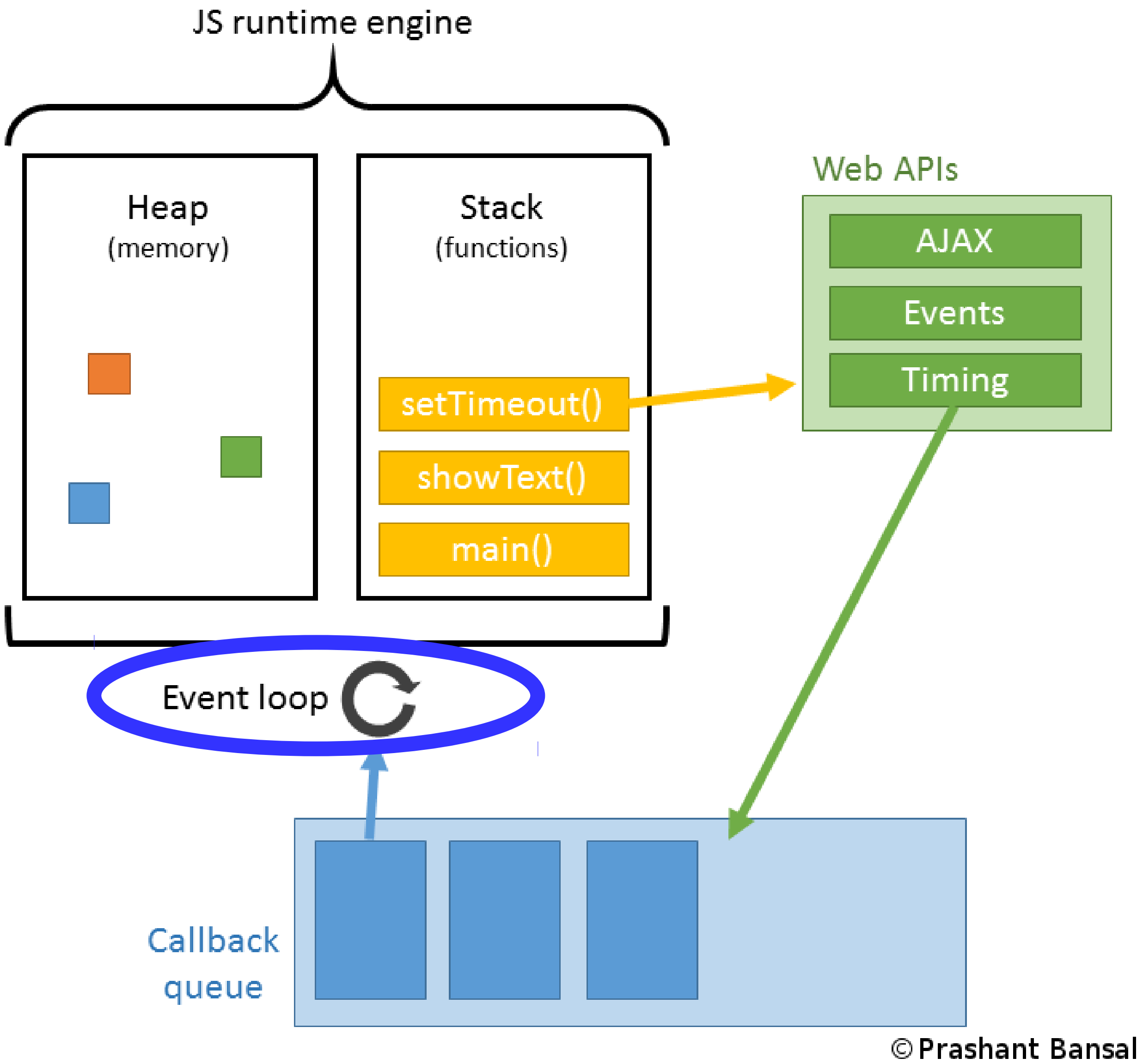
V8




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


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


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

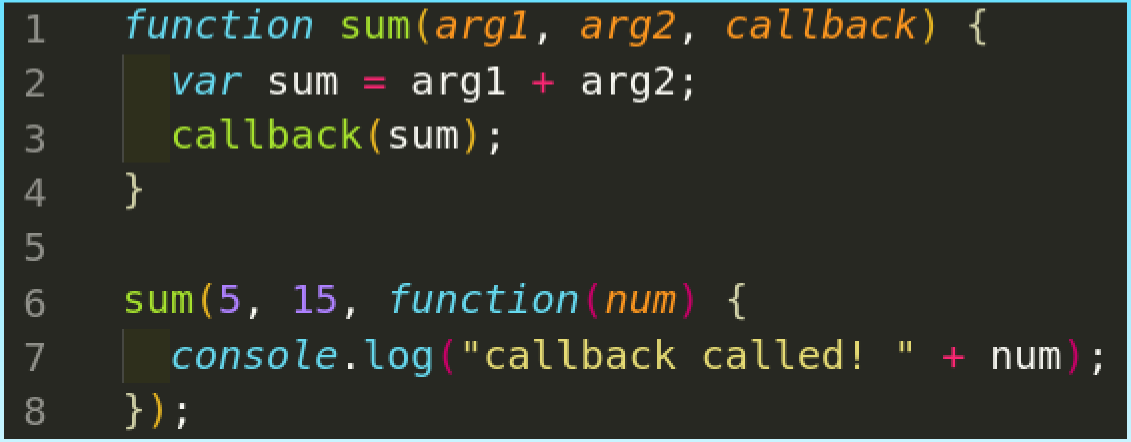

JS runtime engine
Heap
(memory)
Stack
(functions)
setTimeout()
showText()
main()
Web APIs
AJAX
Events
Timing
Event loop
Callback
queue
© Prashant Bansal

CALLBACK

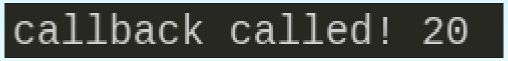
- Функция, которая передается аргументом.
- Вызывается после того, как выполнится другая функция.

CALLBACK

- Функция, которая передается аргументом.
- Вызывается после того, как выполнится другая функция.



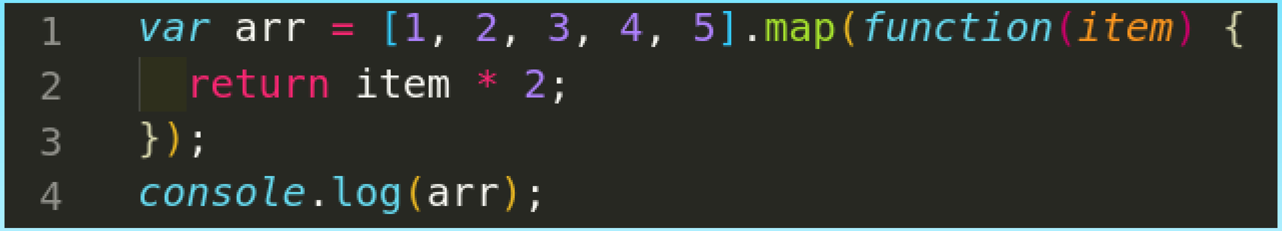
```
function sum(arg1, arg2, callback) {
    var sum = arg1 + arg2;
    callback(sum);
}

sum(5, 15, function(num) {
    console.log("callback called! " + num);
});
```



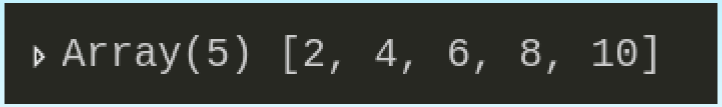
```
callback called! 20
```

CALLBACK

- Функция, которая передается аргументом.
- Вызывается после того, как выполнится другая функция.

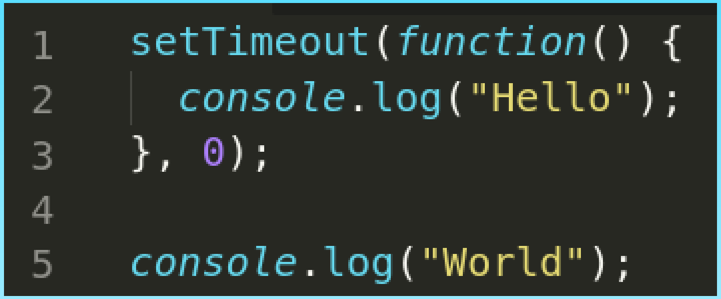


```
var arr = [1, 2, 3, 4, 5].map(function(item) {
    return item * 2;
});
console.log(arr);
```

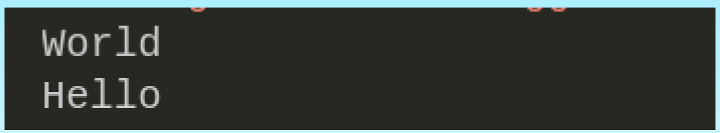


```
▸ Array(5) [2, 4, 6, 8, 10]
```

CALLBACK
Async



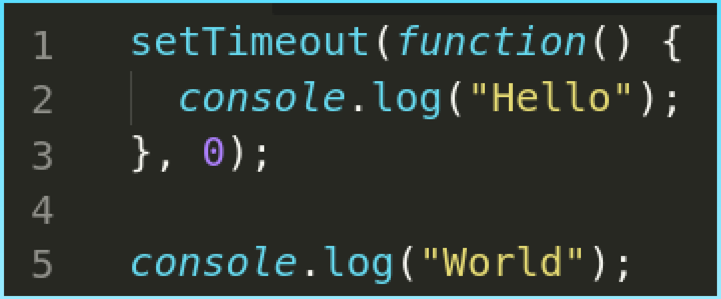
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



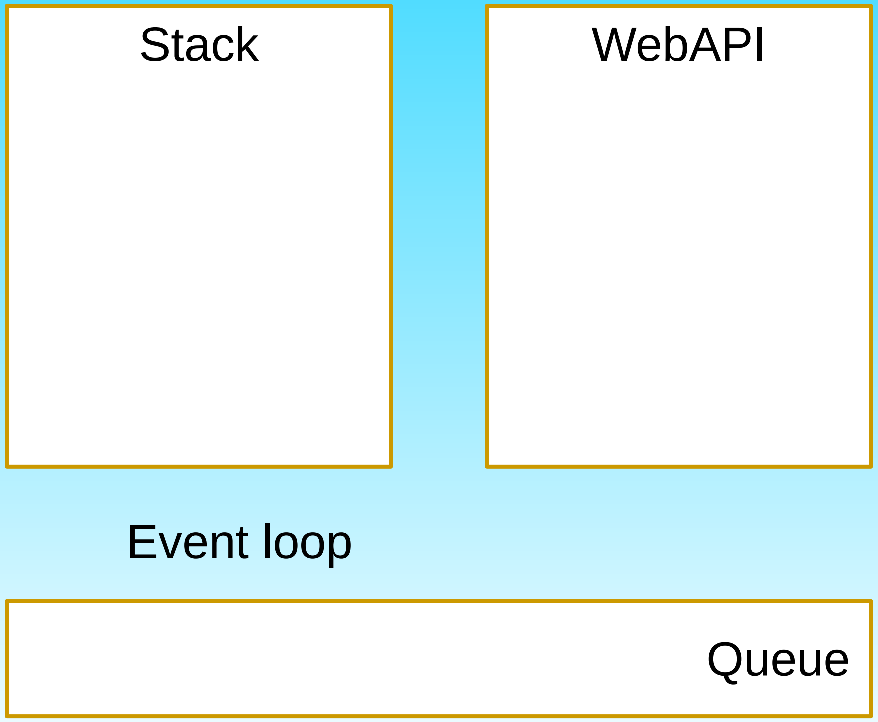
```
World
Hello
```

CALLBACK

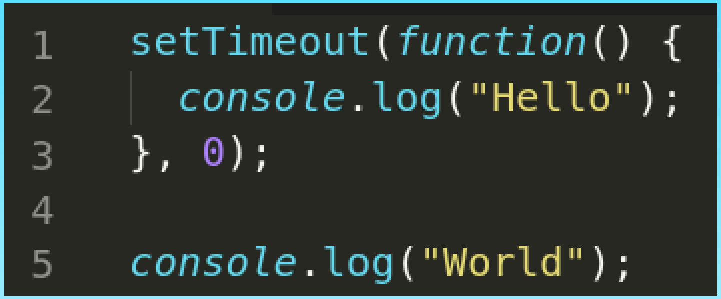
Async



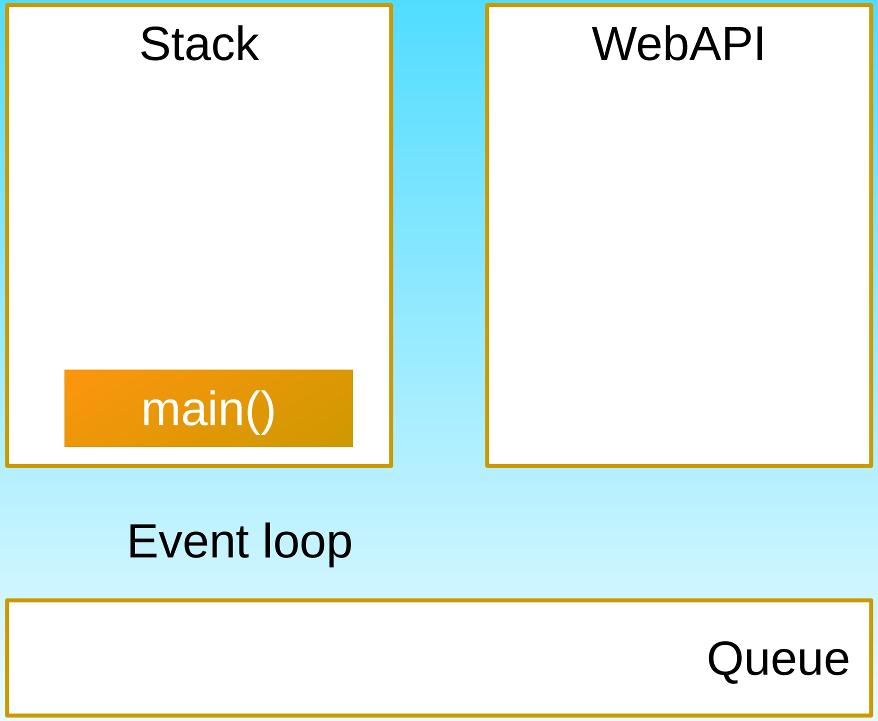
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



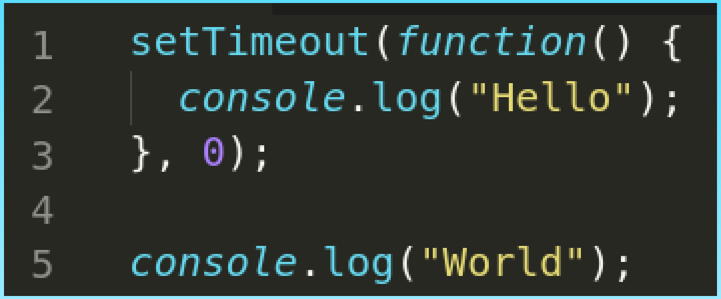
CALLBACK
Async



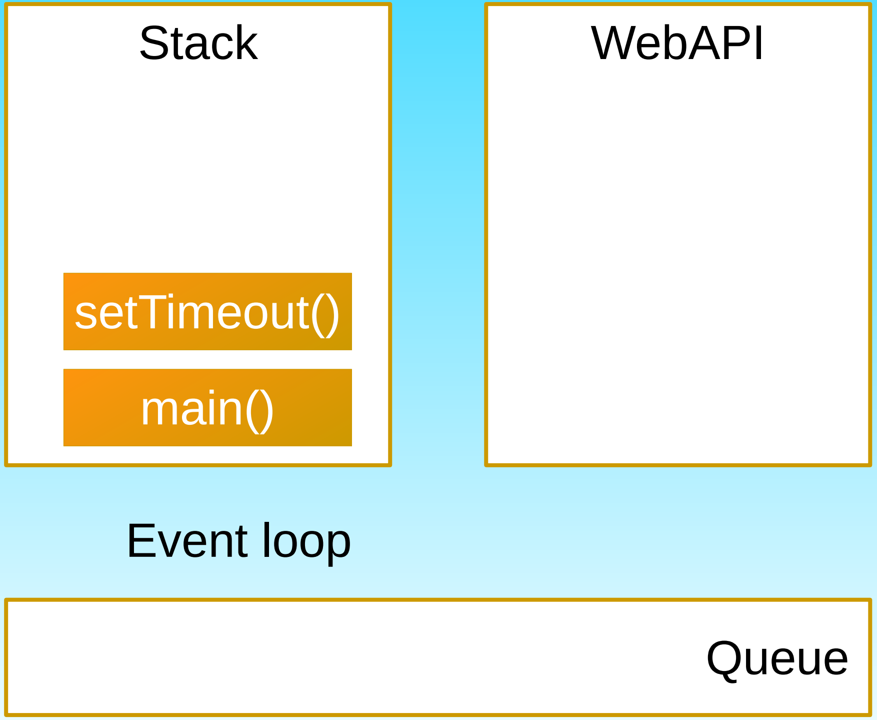
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



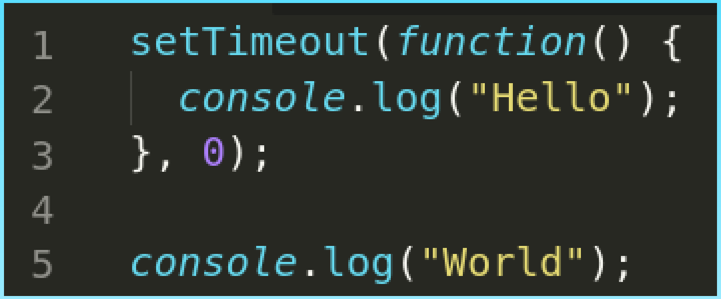
CALLBACK
Async



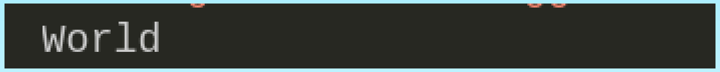
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



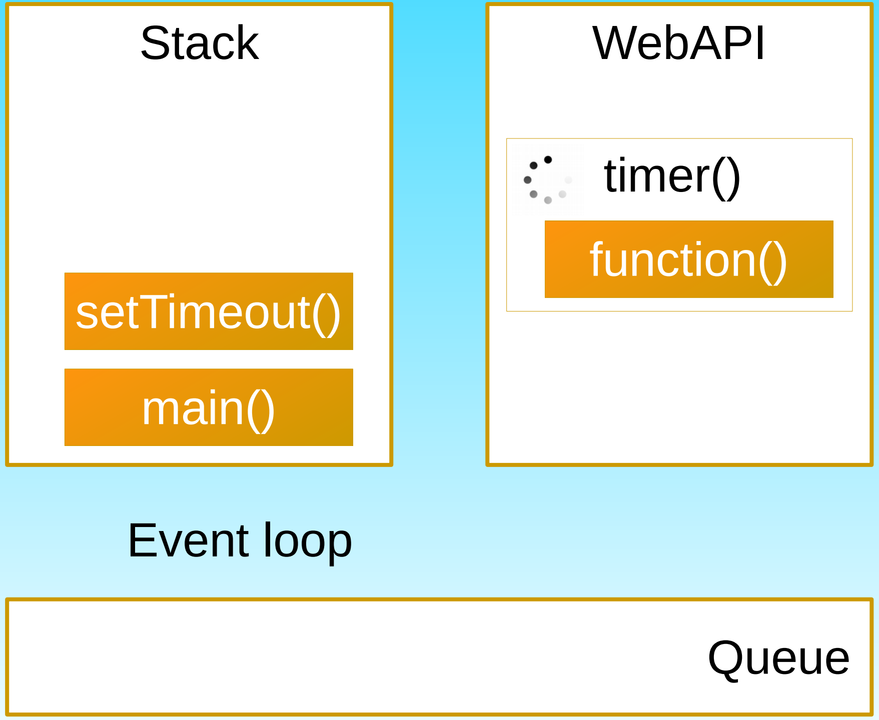
CALLBACK
Async



```
setTimeout(function() {
  console.log("Hello");
}, 0);

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

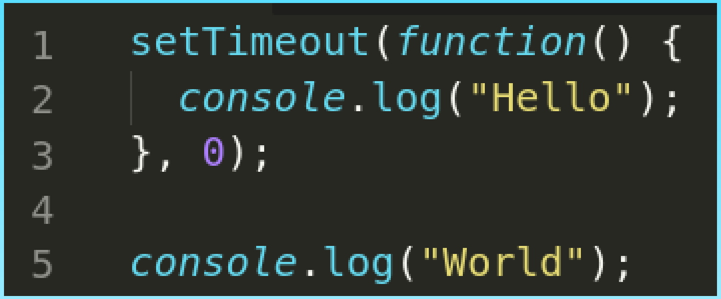


```
World
```

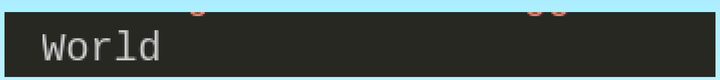


CALLBACK

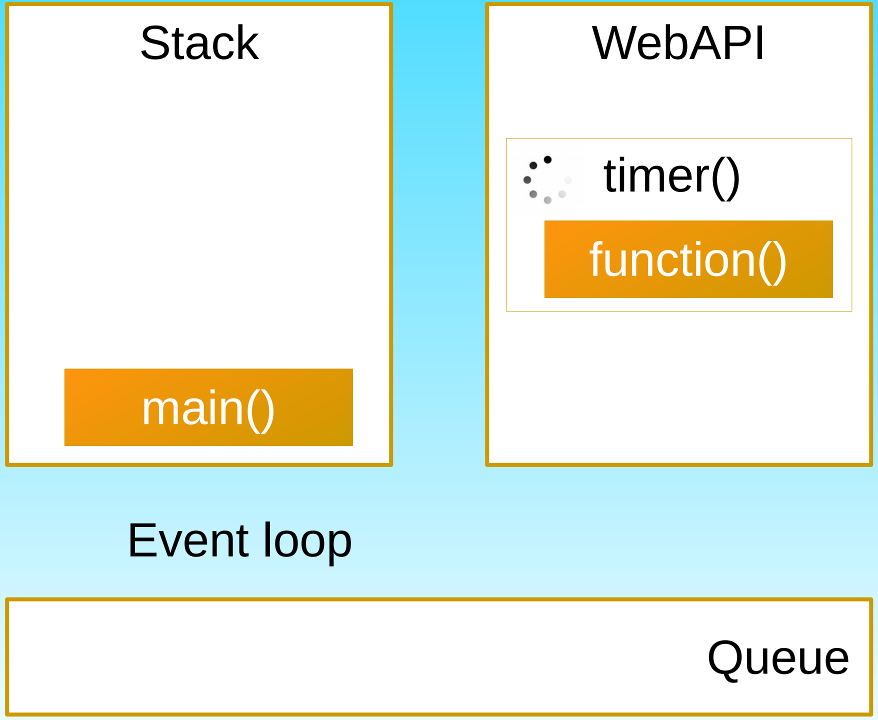
Async



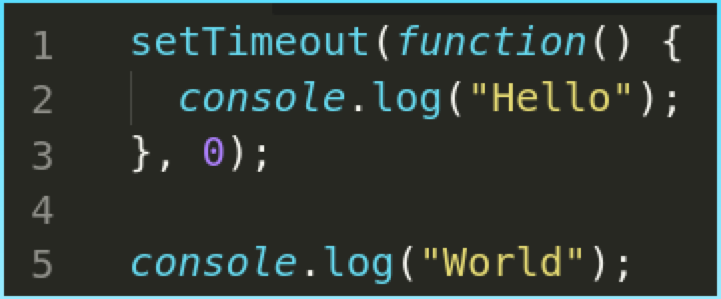
```javascript
setTimeout(function() {
    console.log("Hello");
}, 0);

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



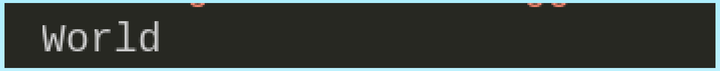
```
World
```



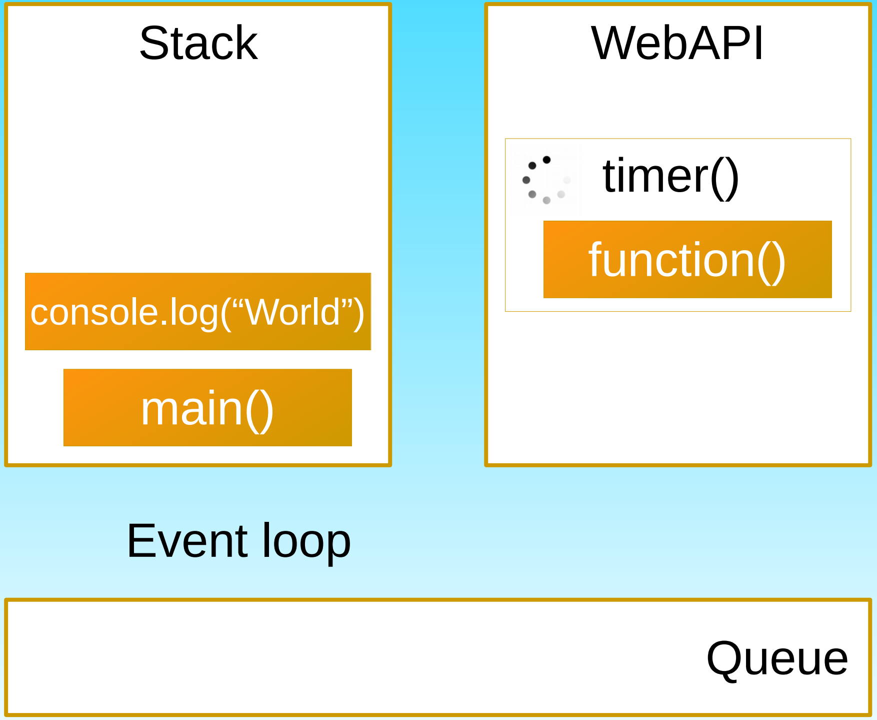
CALLBACK
Async



```
setTimeout(function() {
  console.log("Hello");
}, 0);

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

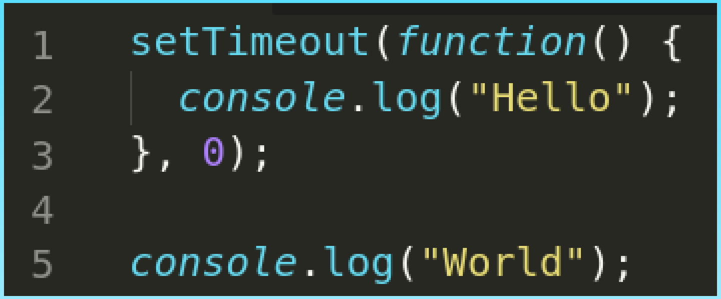


```
World
```

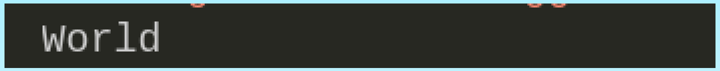


CALLBACK

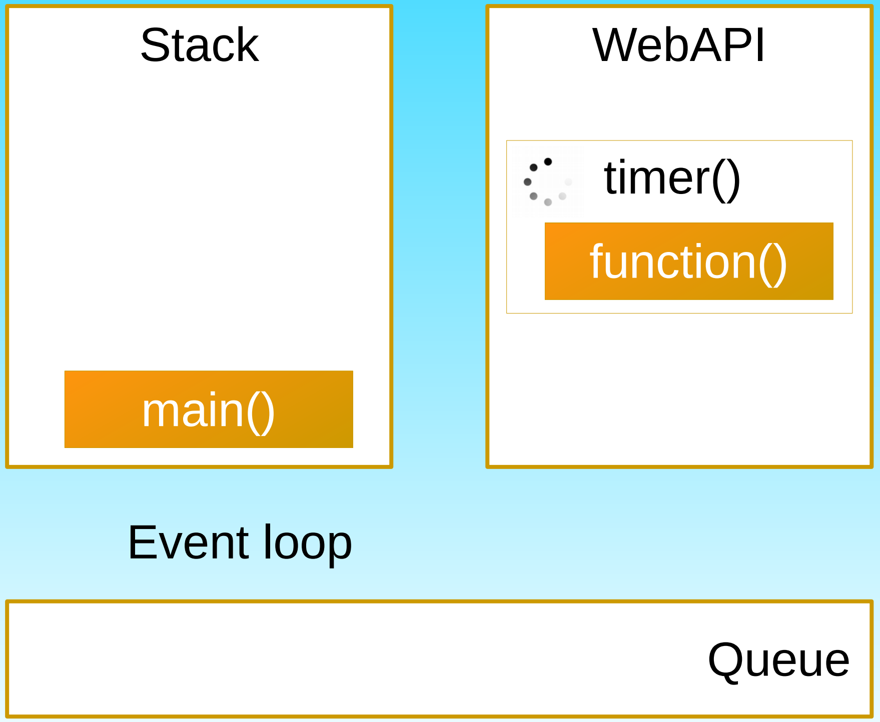
Async



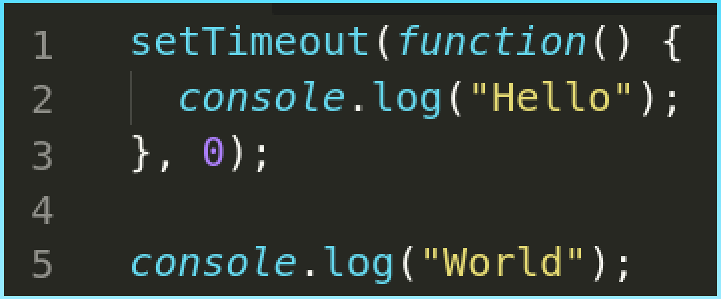
```javascript
setTimeout(function() {
  console.log("Hello");
}, 0);

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



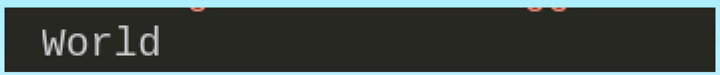
```
World
```



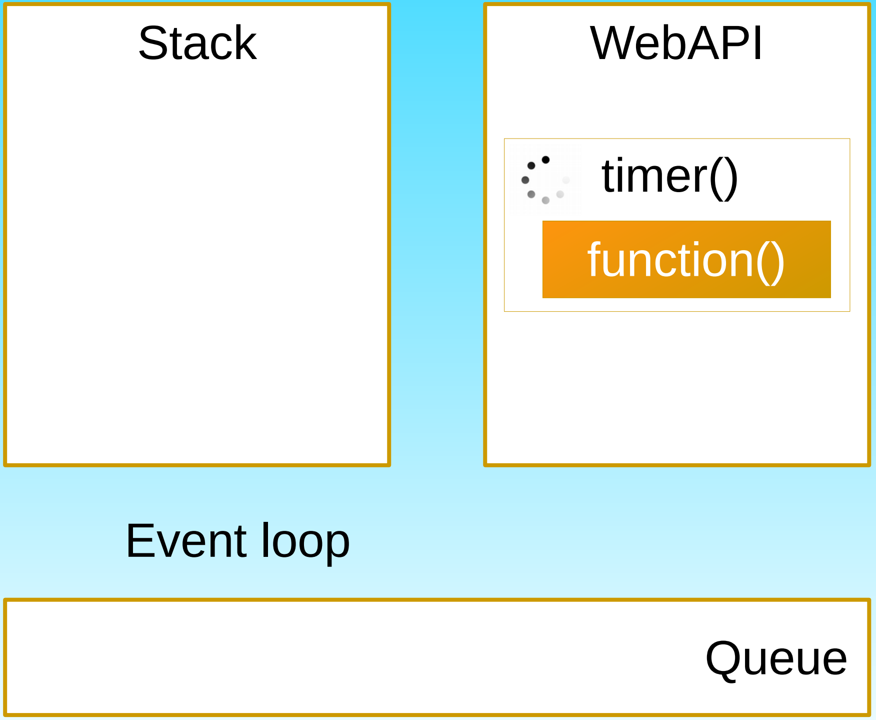
CALLBACK Async



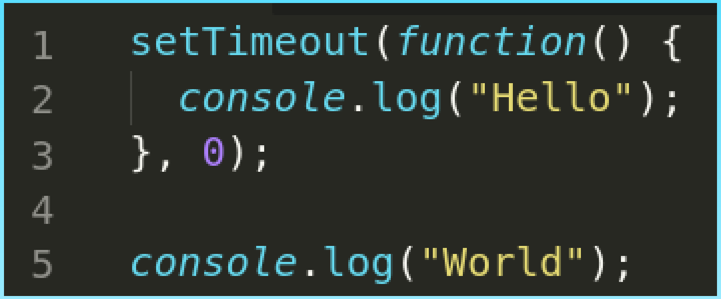
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



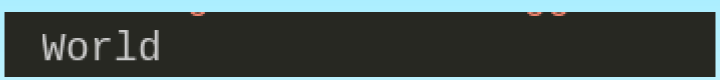
```
World
```



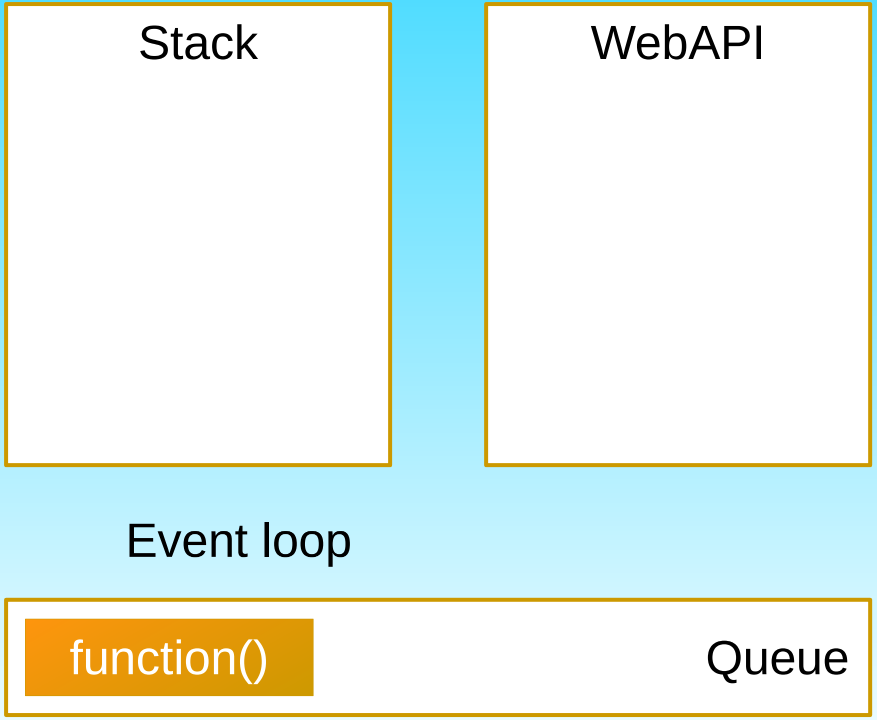
CALLBACK
Async



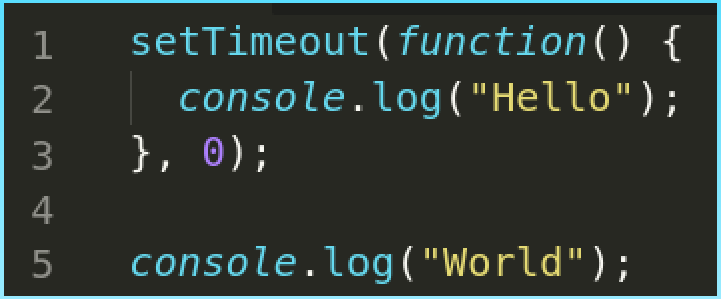
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



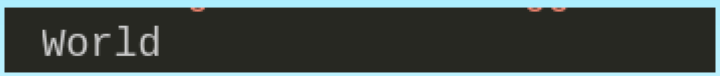
```
World
```



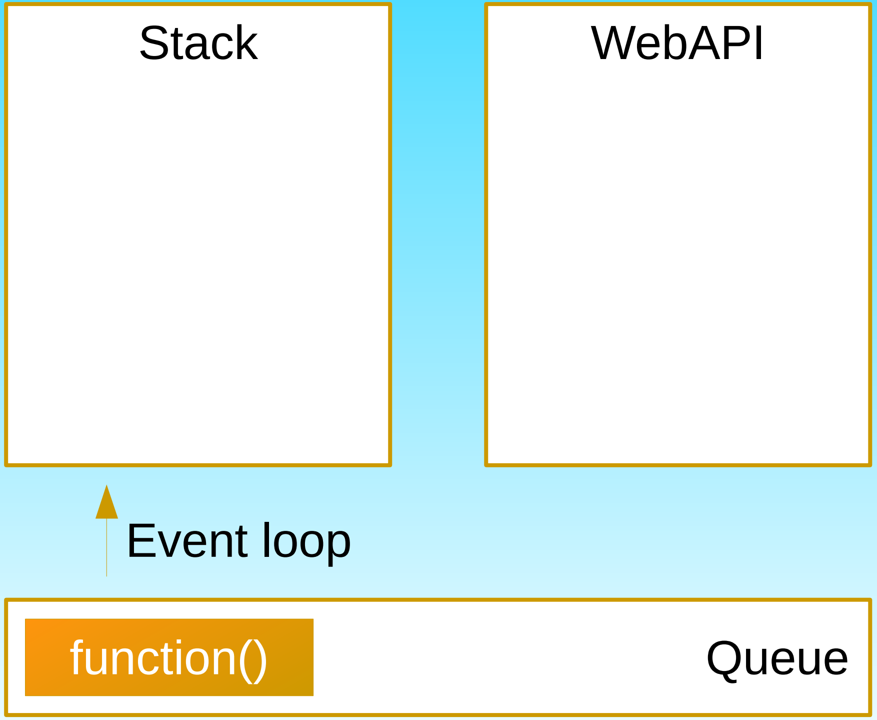
CALLBACK
Async



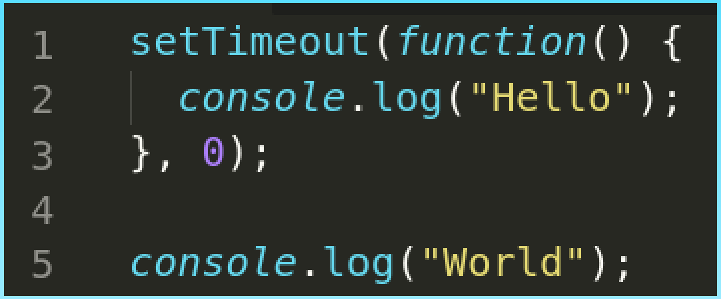
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



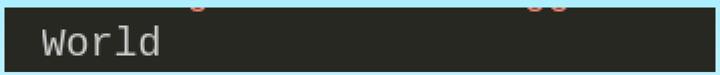
```
World
```



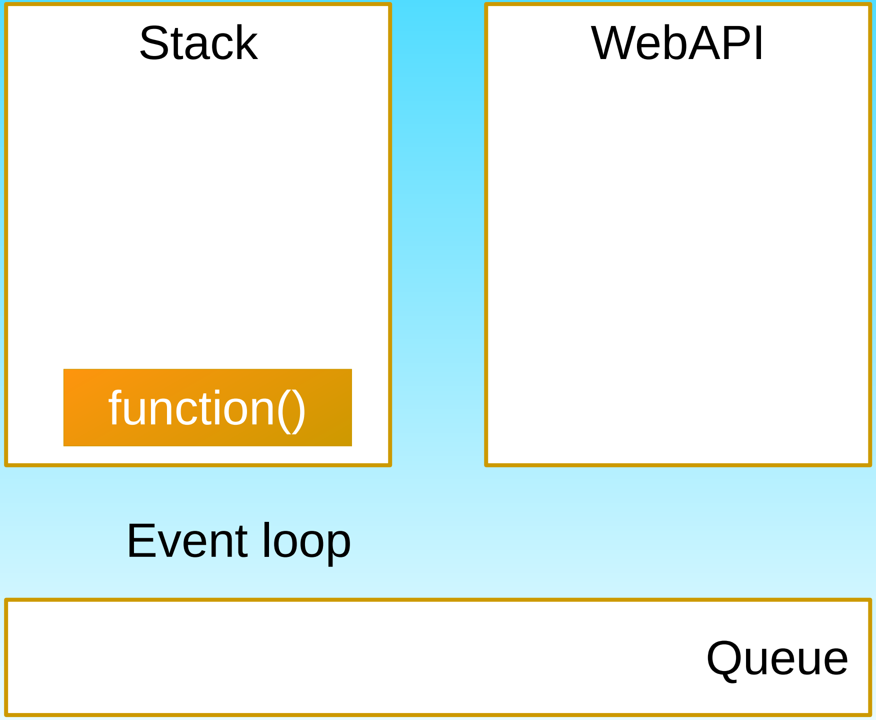
CALLBACK
Async



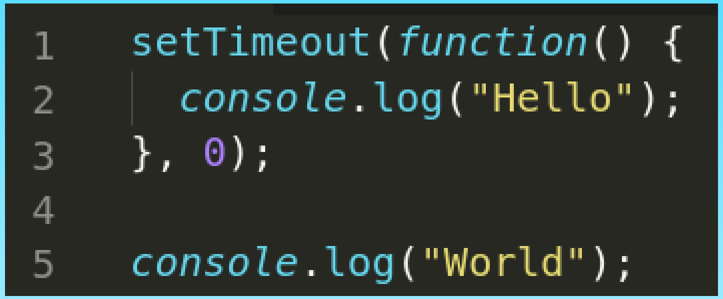
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



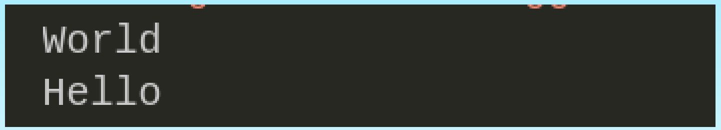
```
World
```



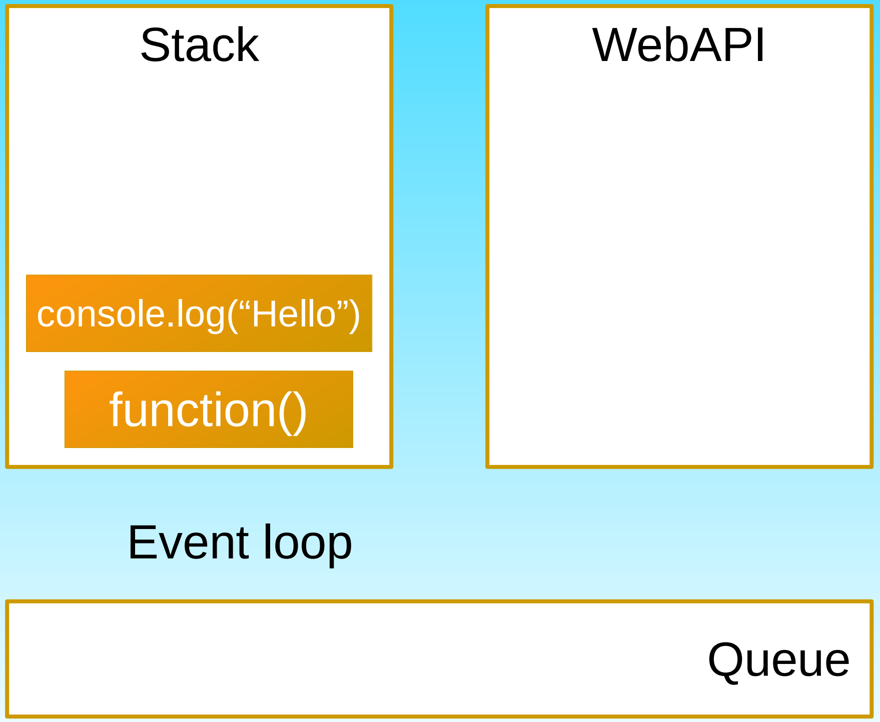
CALLBACK
Async



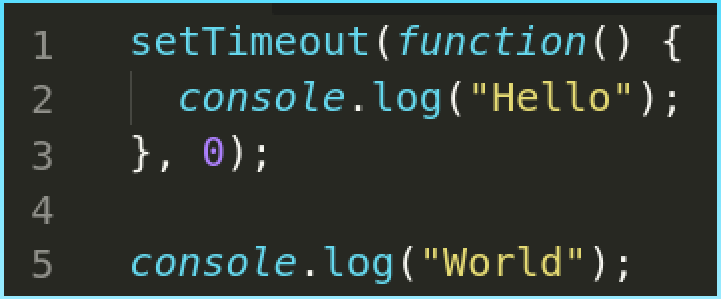
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



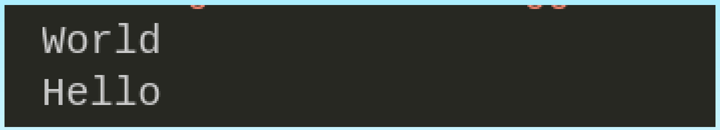
```
World
Hello
```



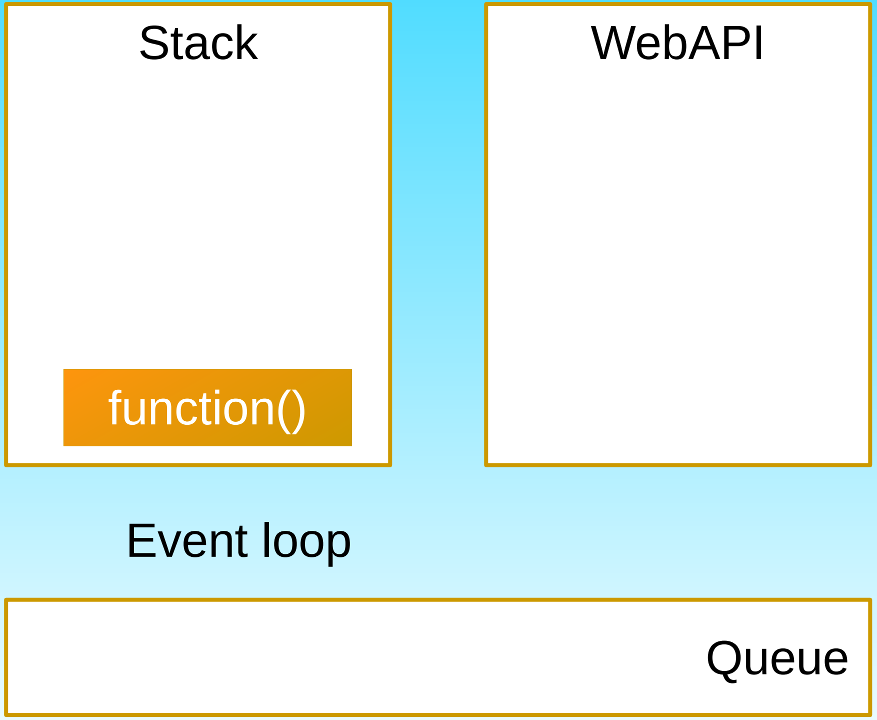
CALLBACK
Async



```
setTimeout(function() {
  console.log("Hello");
}, 0);

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

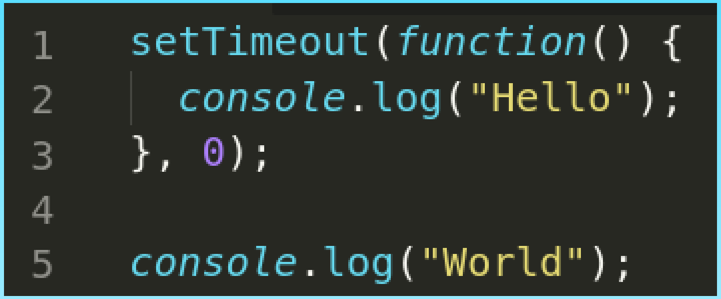


```
World
Hello
```

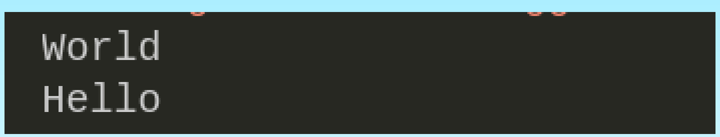


CALLBACK

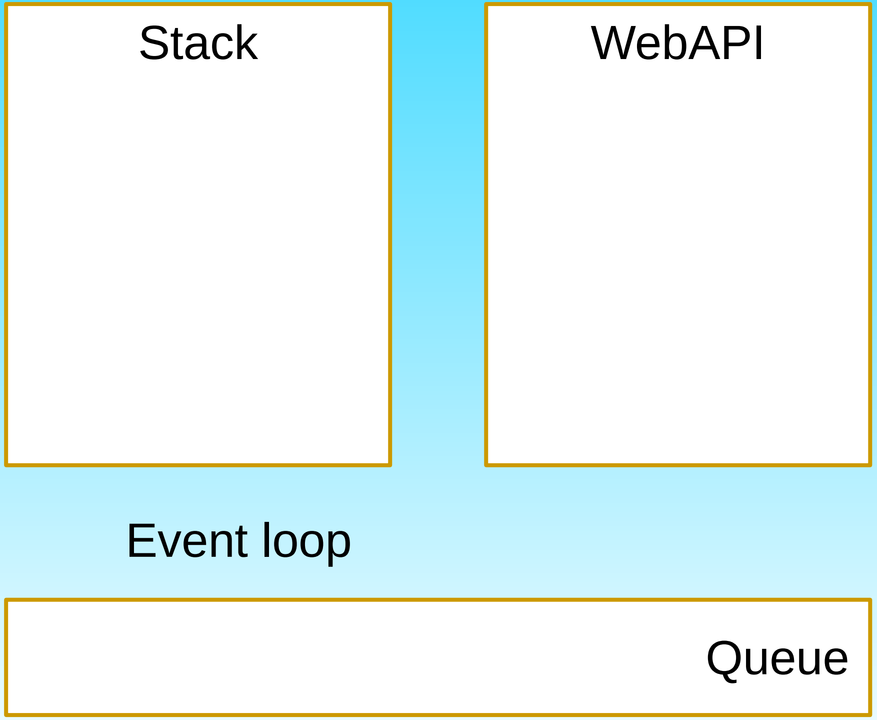
Async

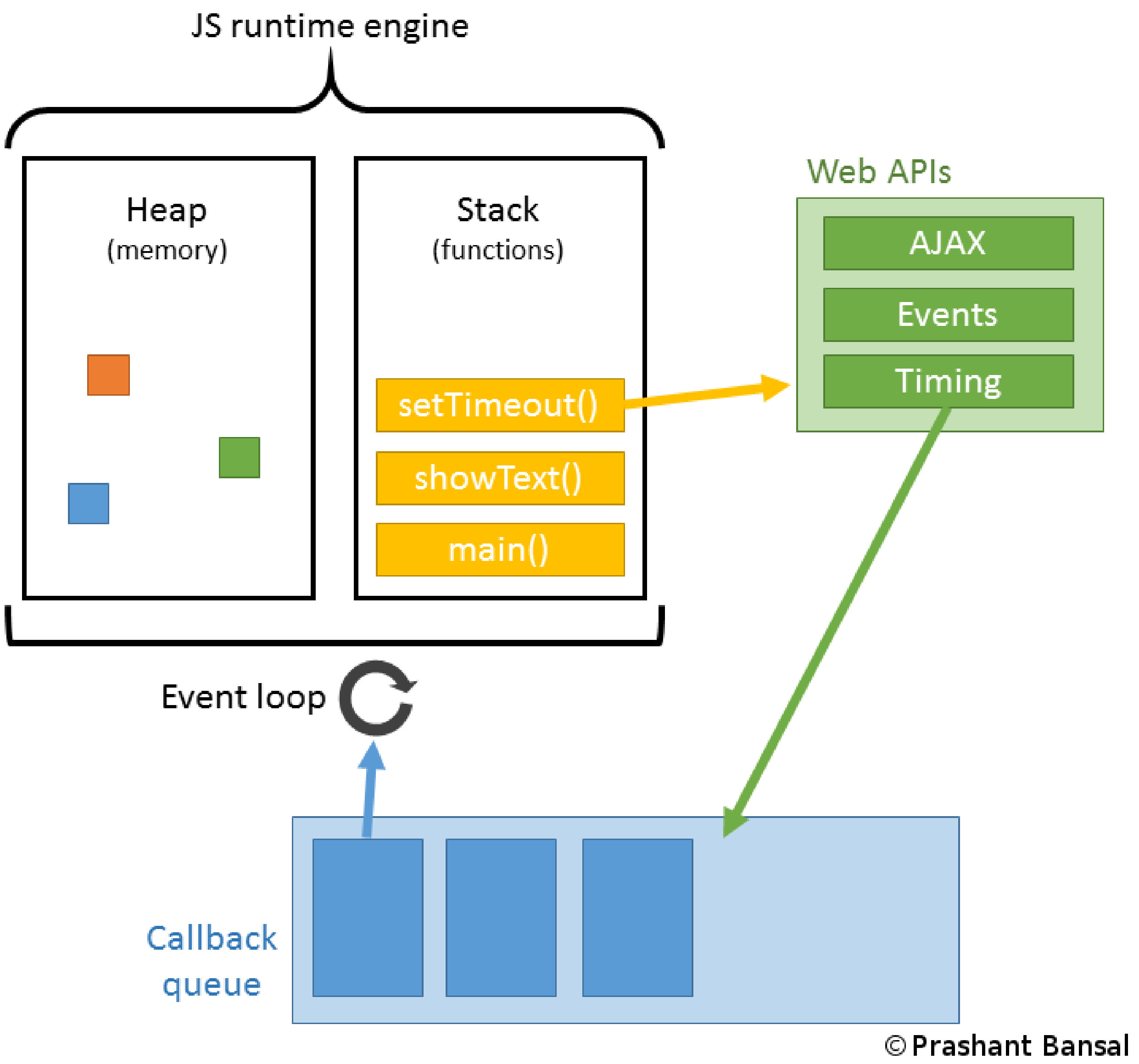


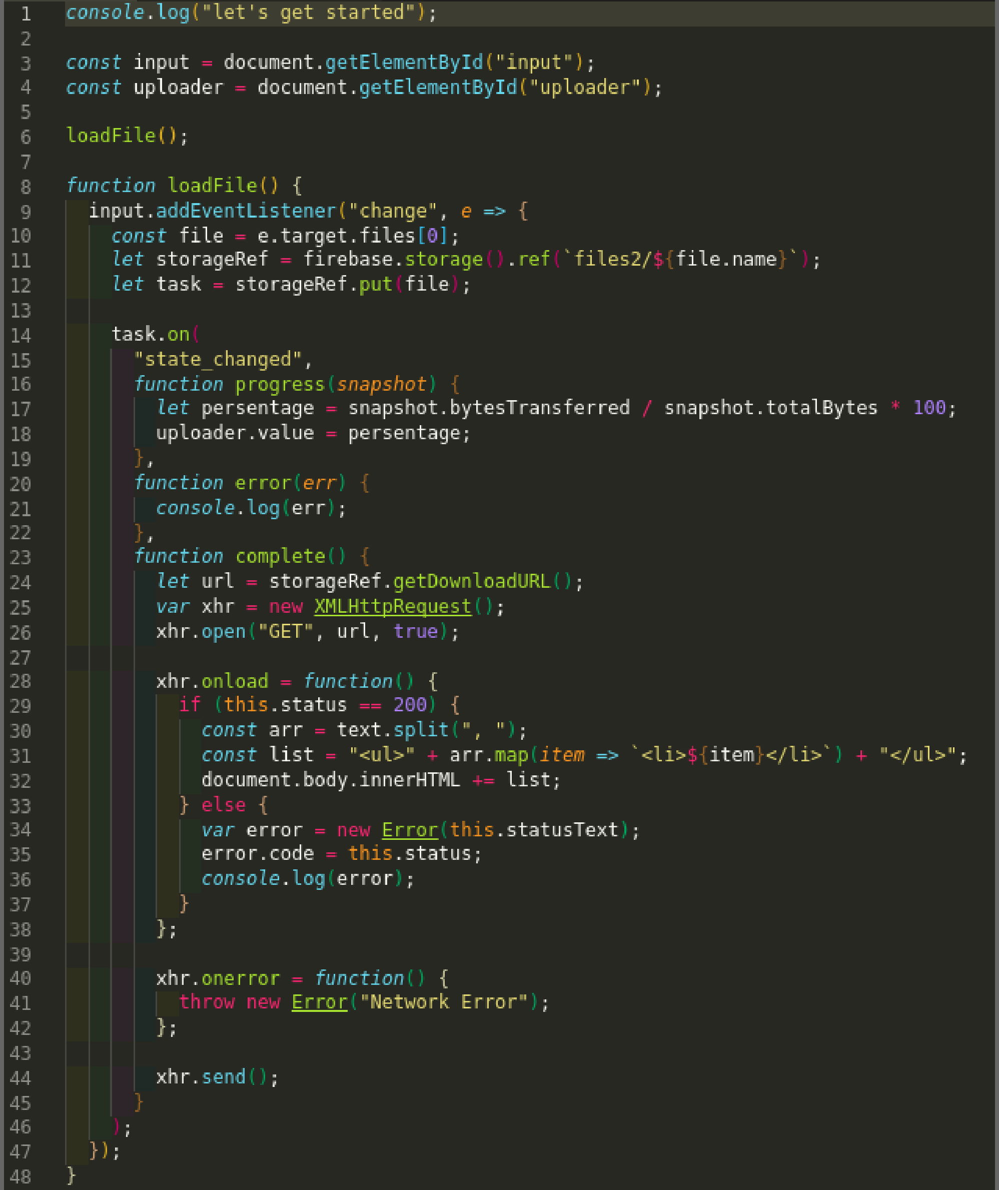
```
setTimeout(function() {
  console.log("Hello");
}, 0);

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



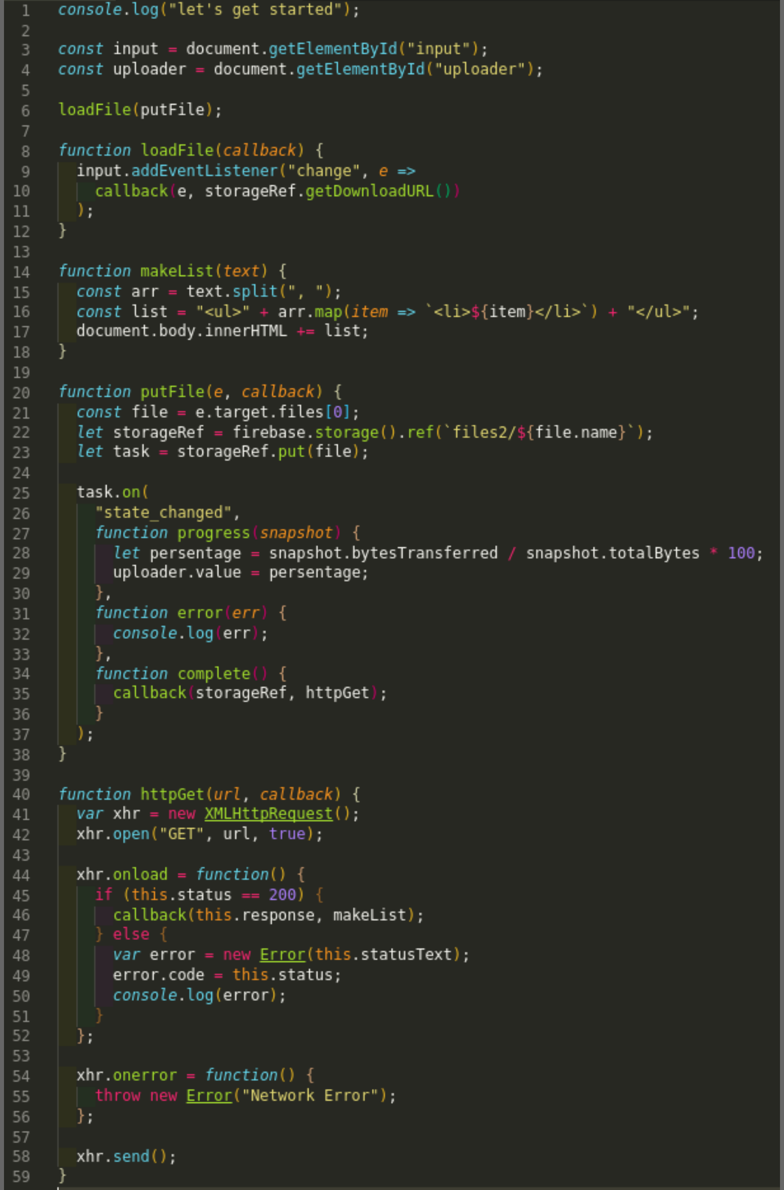
```
World
Hello
```




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

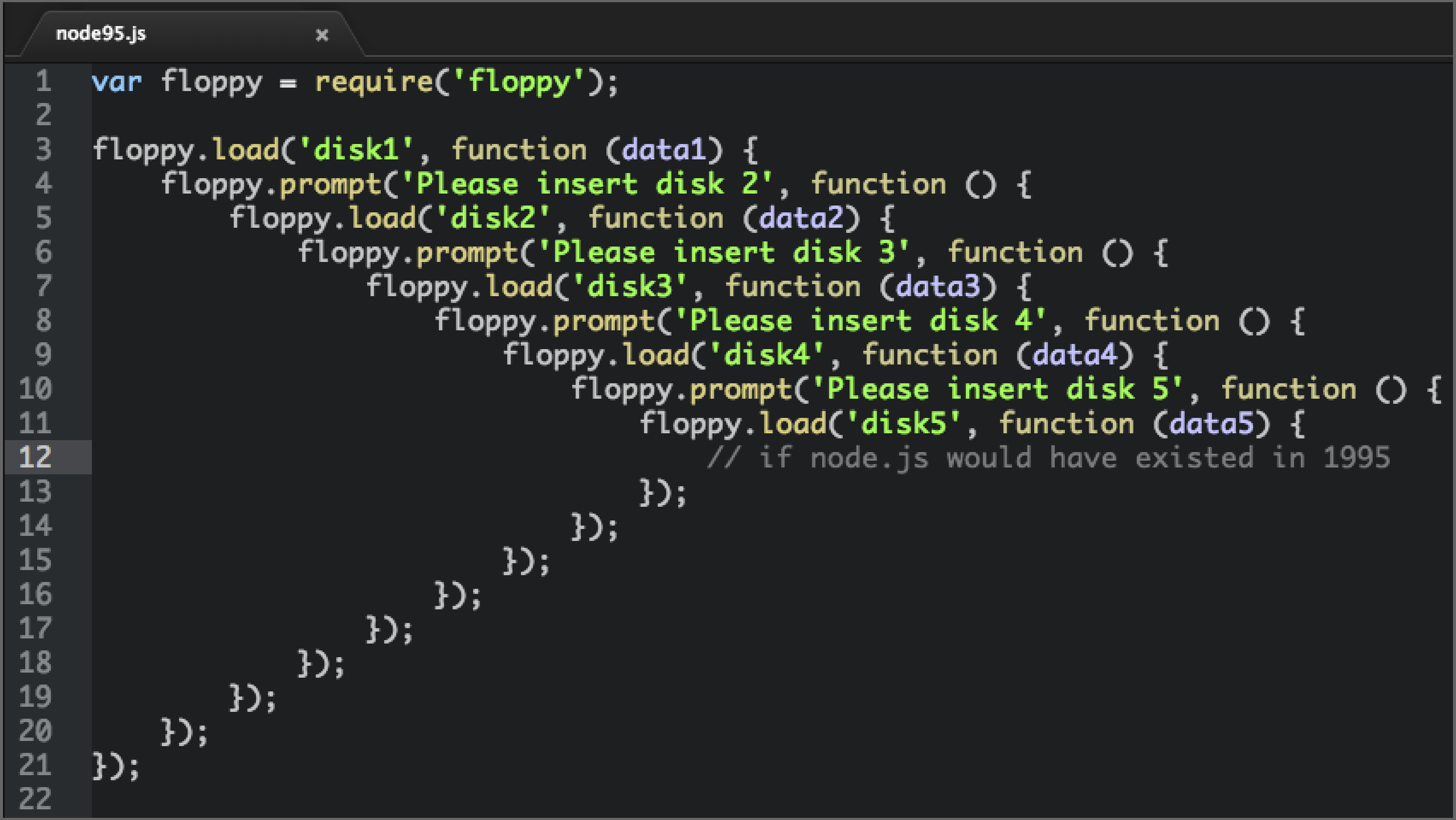


```
console.log("let's get started");

const input = document.getElementById("input");
const uploader = document.getElementById("uploader");

loadFile();

function loadFile() {
  input.addEventListener("change", e => {
    const file = e.target.files[0];
    let storageRef = firebase.storage().ref(`files2/${file.name}`);
    let task = storageRef.put(file);

    task.on(
      "state_changed",
      function progress(snapshot) {
        let persentage = snapshot.bytesTransferred / snapshot.totalBytes * 100;
        uploader.value = persentage;
      },
      function error(err) {
        console.log(err);
      },
      function complete() {
        let url = storageRef.getDownloadURL();
        var xhr = new XMLHttpRequest();
        xhr.open("GET", url, true);

        xhr.onload = function() {
          if (this.status == 200) {
            const arr = text.split(", ");
            const list = "<ul>" + arr.map(item => `<li>${item}</li>`) + "</ul>";
            document.body.innerHTML += list;
          } else {
            var error = new Error(this.statusText);
            error.code = this.status;
            console.log(error);
          }
        };

        xhr.onerror = function() {
          throw new Error("Network Error");
        };

        xhr.send();
      }
    );
  });
}
```



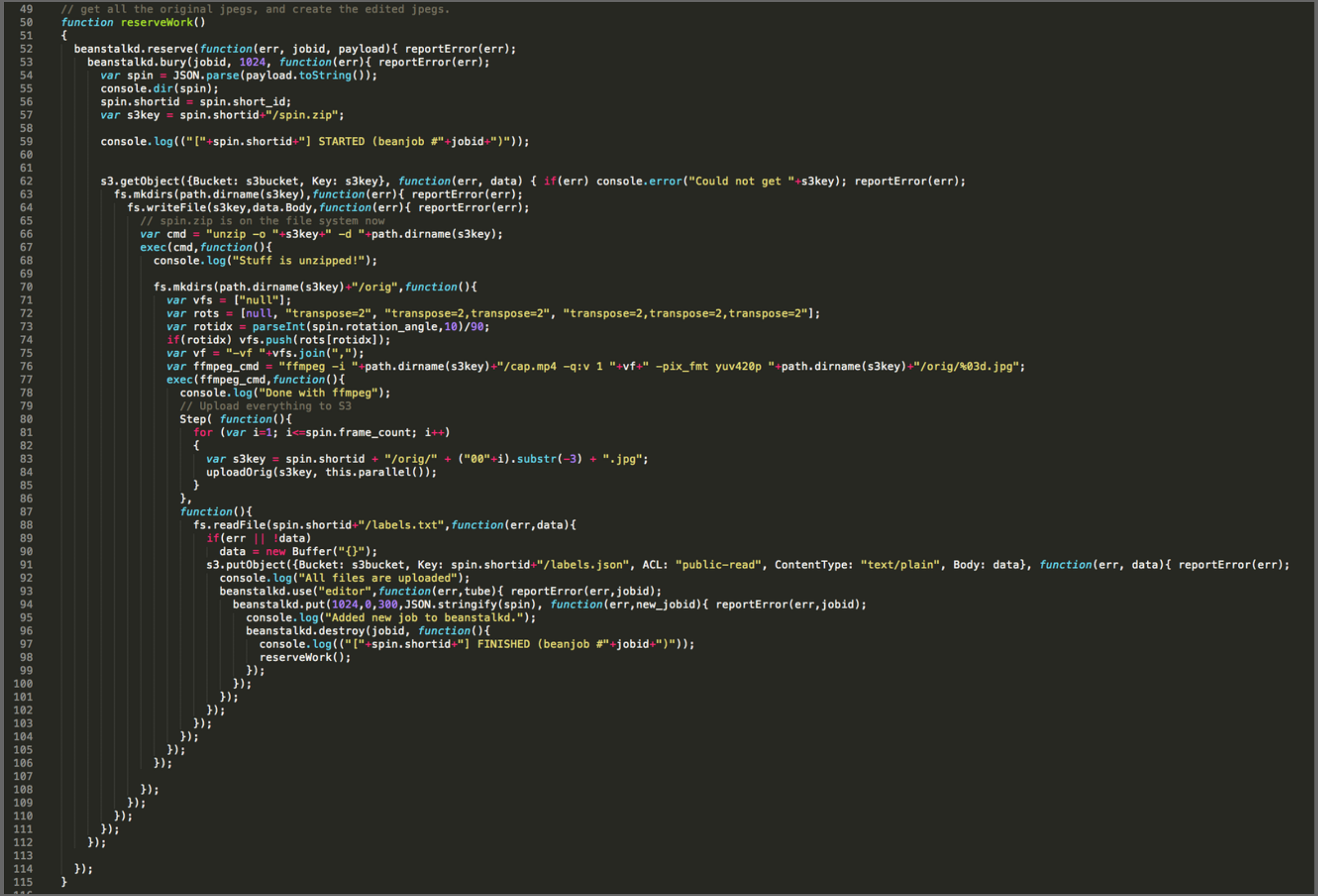
```
console.log("let's get started");

const input = document.getElementById("input");
const uploader = document.getElementById("uploader");

loadFile(putFile);

function loadFile(callback) {
  input.addEventListener("change", e =>
    callback(e, storageRef.getDownloadURL())
  );
}

function makeList(text) {
  const arr = text.split(", ");
  const list = "<ul>" + arr.map(item => `<li>${item}</li>`) + "</ul>";
  document.body.innerHTML += list;
}

function putFile(e, callback) {
  const file = e.target.files[0];
  let storageRef = firebase.storage().ref(`files2/${file.name}`);
  let task = storageRef.put(file);

  task.on(
    "state_changed",
    function progress(snapshot) {
      let persentage = snapshot.bytesTransferred / snapshot.totalBytes * 100;
      uploader.value = persentage;
    },
    function error(err) {
      console.log(err);
    },
    function complete() {
      callback(storageRef, httpGet);
    }
  );
}

function httpGet(url, callback) {
  var xhr = new XMLHttpRequest();
  xhr.open("GET", url, true);

  xhr.onload = function() {
    if (this.status == 200) {
      callback(this.response, makeList);
    } else {
      var error = new Error(this.statusText);
      error.code = this.status;
      console.log(error);
    }
  };

  xhr.onerror = function() {
    throw new Error("Network Error");
  };

  xhr.send();
}
```

CALLBACK hell



```
var floppy = require('floppy');

floppy.load('disk1', function (data1) {
    floppy.prompt('Please insert disk 2', function () {
        floppy.load('disk2', function (data2) {
            floppy.prompt('Please insert disk 3', function () {
                floppy.load('disk3', function (data3) {
                    floppy.prompt('Please insert disk 4', function () {
                        floppy.load('disk4', function (data4) {
                            floppy.prompt('Please insert disk 5', function () {
                                floppy.load('disk5', function (data5) {
                                    // if node.js would have existed in 1995
                                });
                            });
                        });
                    });
                });
            });
        });
    });
});
```

CALLBACK hell



```javascript
// get all the original jpegs, and create the edited jpegs.
function reserveWork()
{
  beanstalkd.reserve(function(err, jobid, payload){ reportError(err);
    beanstalkd.bury(jobid, 1024, function(err){ reportError(err);
      var spin = JSON.parse(payload.toString());
      console.dir(spin);
      spin.shortid = spin.short_id;
      var s3key = spin.shortid+"/spin.zip";

      console.log(("["+spin.shortid+"] STARTED (beanjob #"+jobid+")"));


      s3.getObject({Bucket: s3bucket, Key: s3key}, function(err, data) { if(err) console.error("Could not get "+s3key); reportError(err);
        fs.mkdirs(path.dirname(s3key),function(err){ reportError(err);
          fs.writeFile(s3key,data.Body,function(err){ reportError(err);
            // spin.zip is on the file system now
            var cmd = "unzip -o "+s3key+" -d "+path.dirname(s3key);
            exec(cmd,function(){
              console.log("Stuff is unzipped!");

              fs.mkdirs(path.dirname(s3key)+"/orig",function(){
                var vfs = ["null"];
                var rots = [null, "transpose=2", "transpose=2,transpose=2", "transpose=2,transpose=2,transpose=2"];
                var rotidx = parseInt(spin.rotation_angle,10)/90;
                if(rotidx) vfs.push(rots[rotidx]);
                var vf = "-vf "+vfs.join(",");
                var ffmpeg_cmd = "ffmpeg -i "+path.dirname(s3key)+"/cap.mp4 -q:v 1 "+vf+" -pix_fmt yuv420p "+path.dirname(s3key)+"/orig/%03d.jpg";
                exec(ffmpeg_cmd,function(){
                  console.log("Done with ffmpeg");
                  // Upload everything to S3
                  Step( function(){
                    for (var i=1; i<=spin.frame_count; i++)
                    {
                      var s3key = spin.shortid + "/orig/" + ("00"+i).substr(-3) + ".jpg";
                      uploadOrig(s3key, this.parallel());
                    }
                  },
                  function(){
                    fs.readFile(spin.shortid+"/labels.txt",function(err,data){
                      if(err || !data)
                        data = new Buffer("{}");
                      s3.putObject({Bucket: s3bucket, Key: spin.shortid+"/labels.json", ACL: "public-read", ContentType: "text/plain", Body: data}, function(err, data){ reportError(err);
                        console.log("All files are uploaded");
                        beanstalkd.use("editor",function(err,tube){ reportError(err,jobid);
                          beanstalkd.put(1024,0,300,JSON.stringify(spin), function(err,new_jobid){ reportError(err,jobid);
                            console.log("Added new job to beanstalkd.");
                            beanstalkd.destroy(jobid, function(){
                              console.log(("["+spin.shortid+"] FINISHED (beanjob #"+jobid+")"));
                              reserveWork();
                            });
                          });
                        });
                      });
                    });
                  });
                });
              });

            });
          });
        });
      });
    });

  });
}
```

JS Promise

#Javascript

25.4 Promise Objects

A Promise is an object that is used as a placeholder for the eventual results of a deferred (and possibly asynchronous) computation.

Any Promise object is in one of three mutually exclusive states: fulfilled, rejected, and pending:

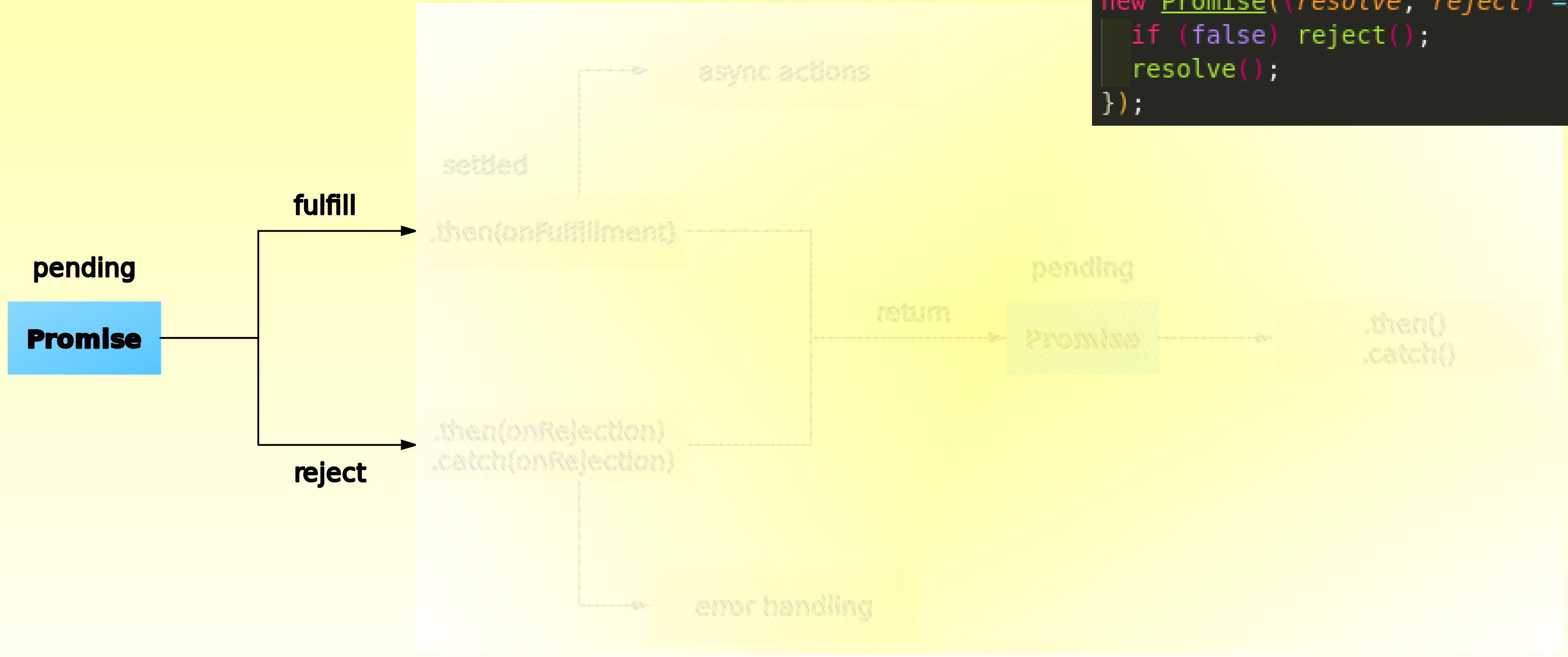
A promise p is fulfilled if p.then(f, r) will immediately enqueue a Job to call the function f.

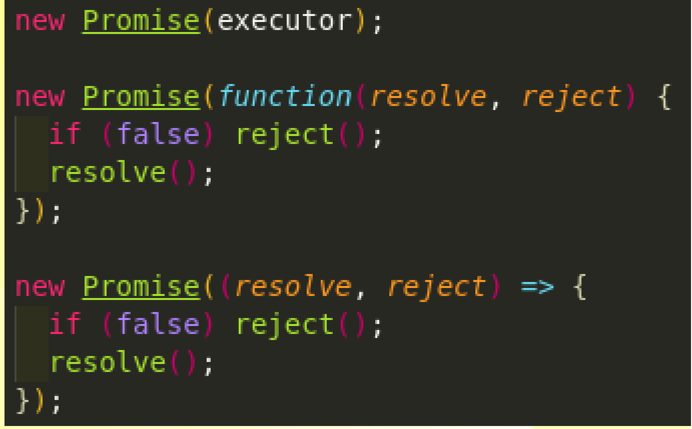
A promise p is rejected if p.then(f, r) will immediately enqueue a Job to call the function r.

A promise is pending if it is neither fulfilled nor rejected.

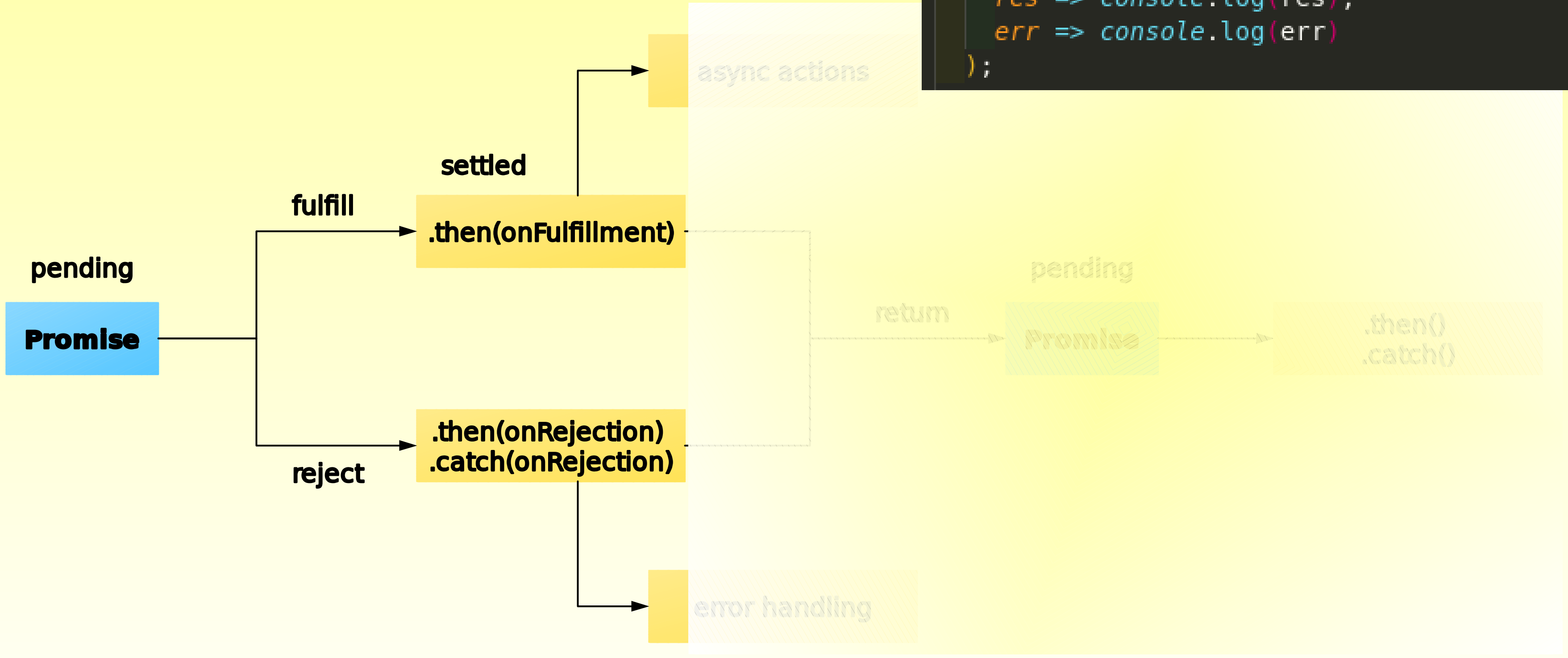
A promise is said to be settled if it is not pending, i.e. if it is either fulfilled or rejected.

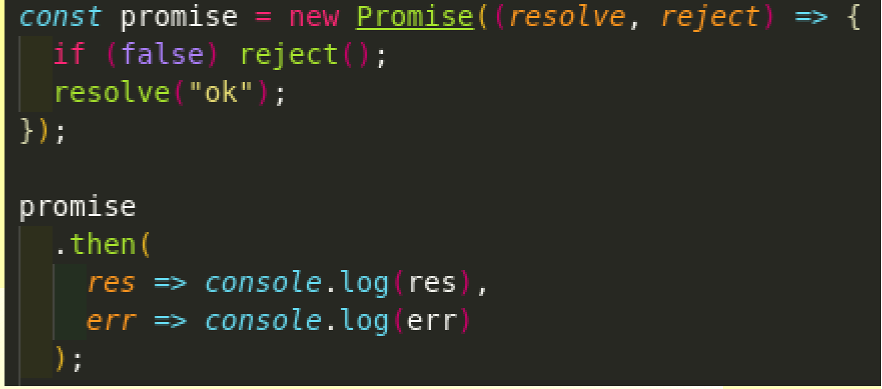
A promise is resolved if it is settled or if it has been “locked in” to match the state of another promise. Attempting to resolve or reject a resolved promise has no effect. A promise is unresolved if it is not resolved. An unresolved promise is always in the pending state. A resolved promise may be pending, fulfilled or rejected.





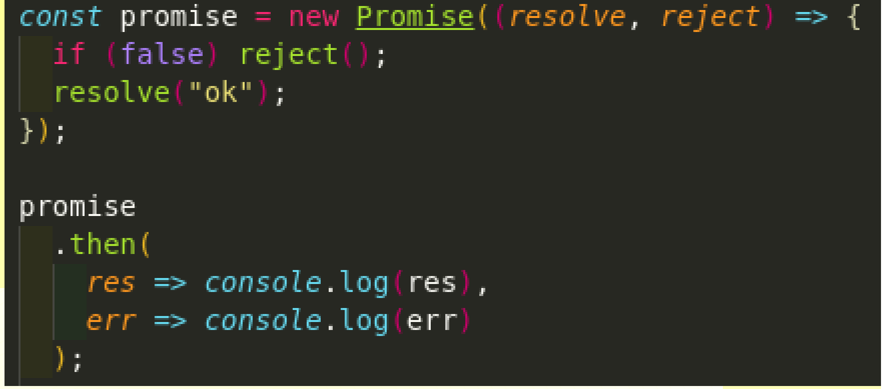
```
new Promise(executor);

new Promise(function(resolve, reject) {
  if (false) reject();
  resolve();
});

new Promise((resolve, reject) => {
  if (false) reject();
  resolve();
});
```



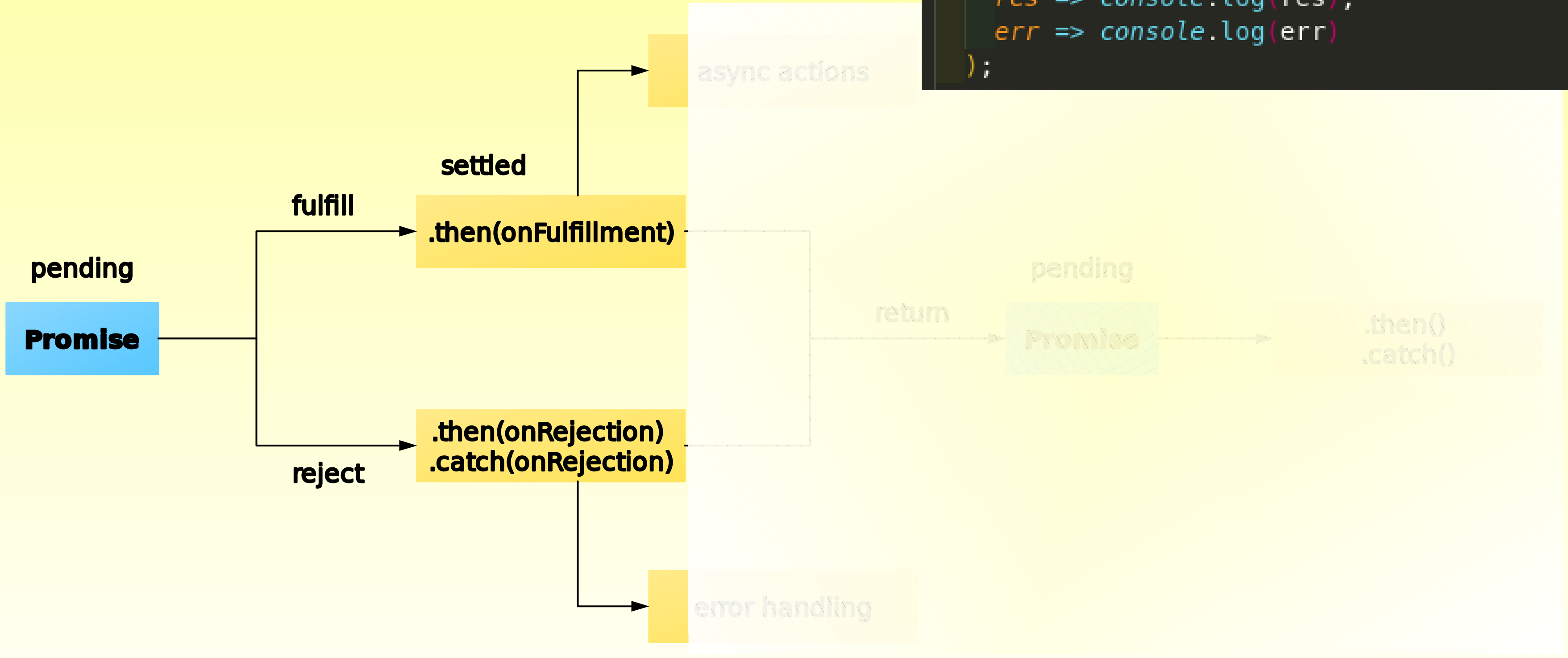


```
const promise = new Promise((resolve, reject) => {
  if (false) reject();
  resolve("ok");
});

promise
  .then(
    res => console.log(res),
    err => console.log(err)
  );
```

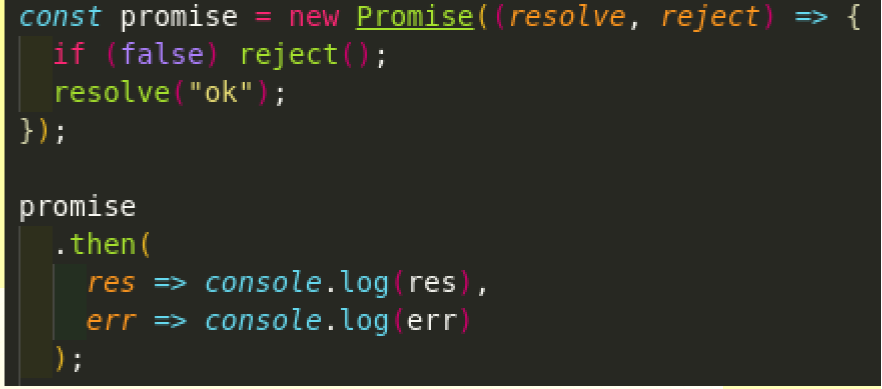
.then - это new Promise



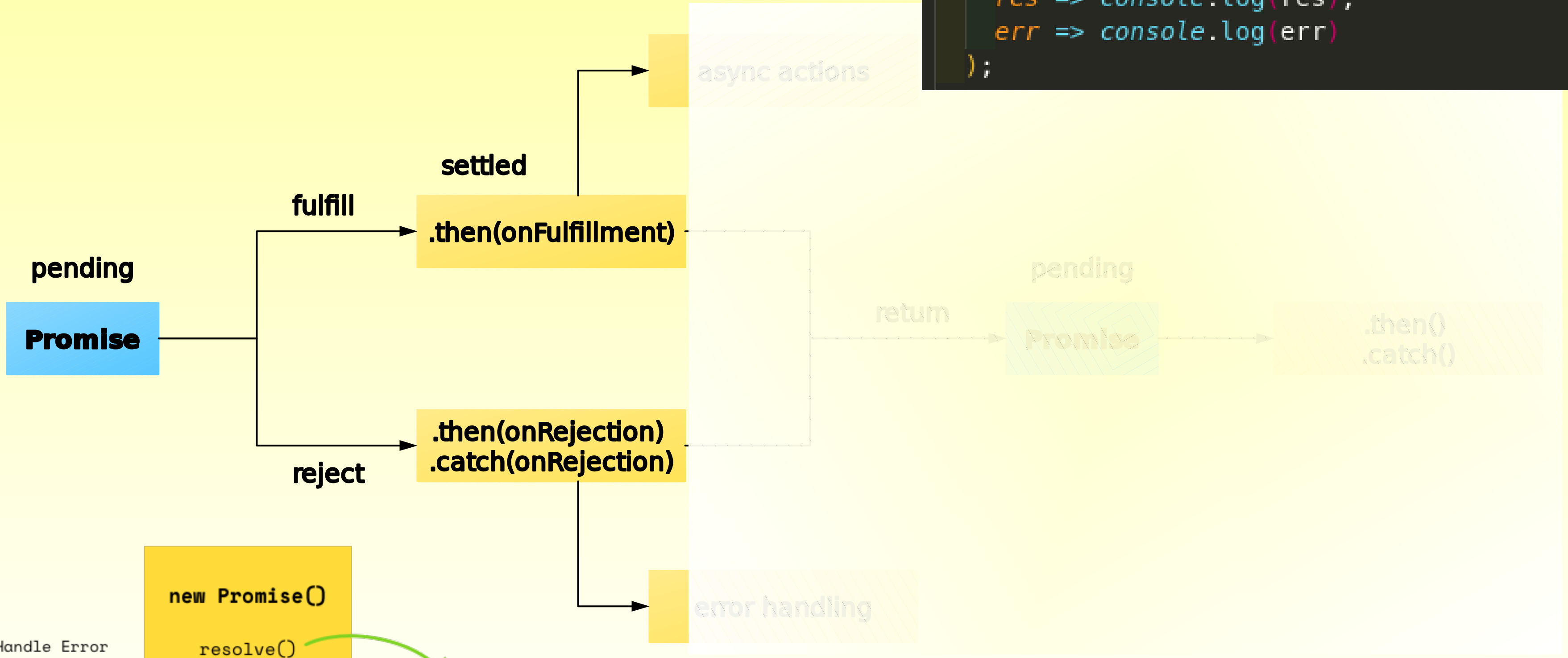
```
const promise = new Promise((resolve, reject) => {
  if (false) reject();
  resolve("ok");
});

promise
  .then(
    res => console.log(res),
    err => console.log(err)
  );
```

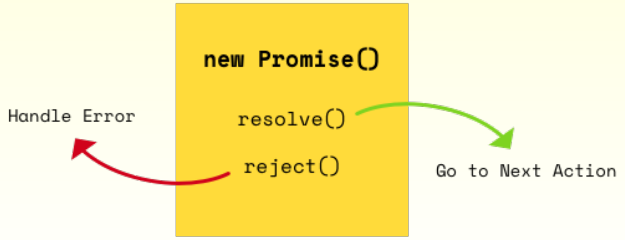


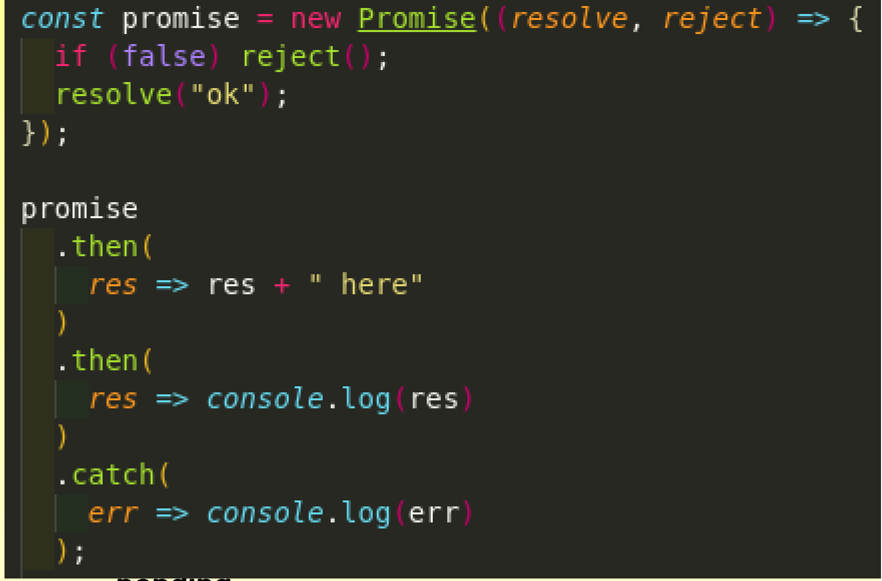
.then - это new Promise



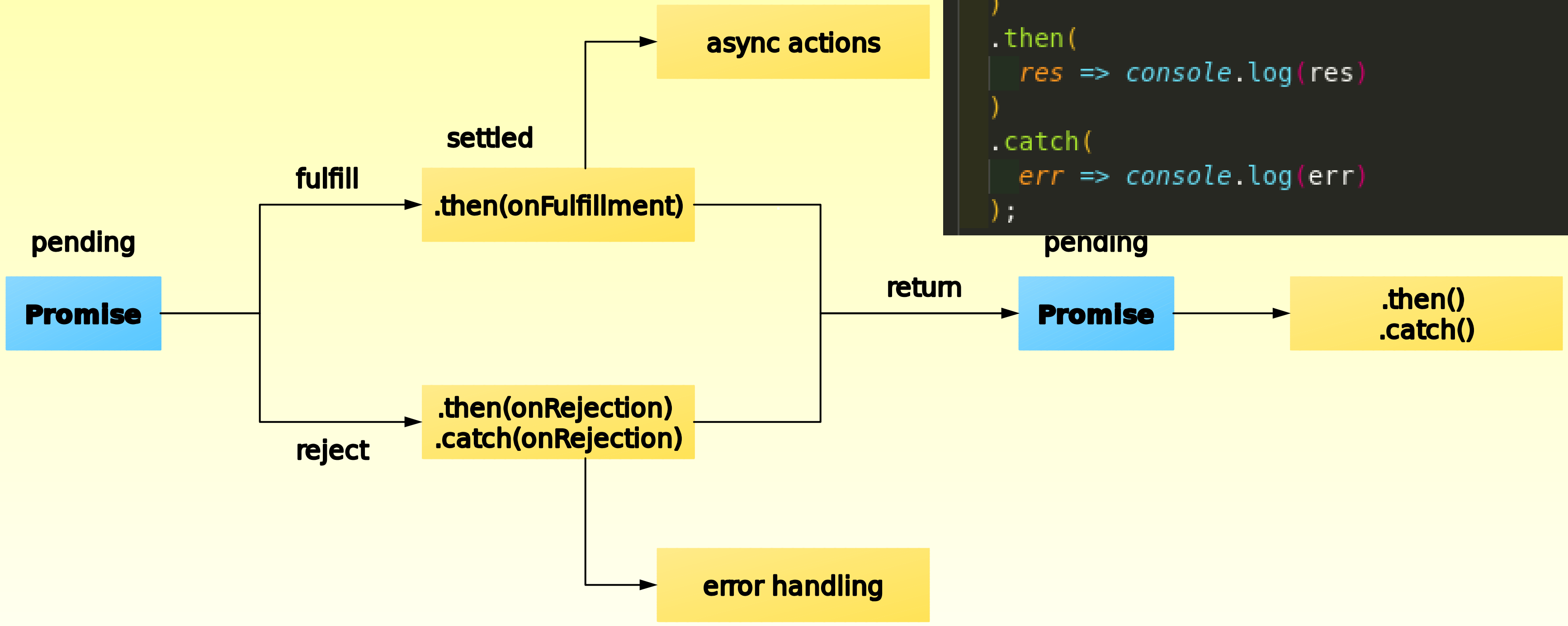
```
const promise = new Promise((resolve, reject) => {
  if (false) reject();
  resolve("ok");
});

promise
  .then(
    res => console.log(res),
    err => console.log(err)
  );
```

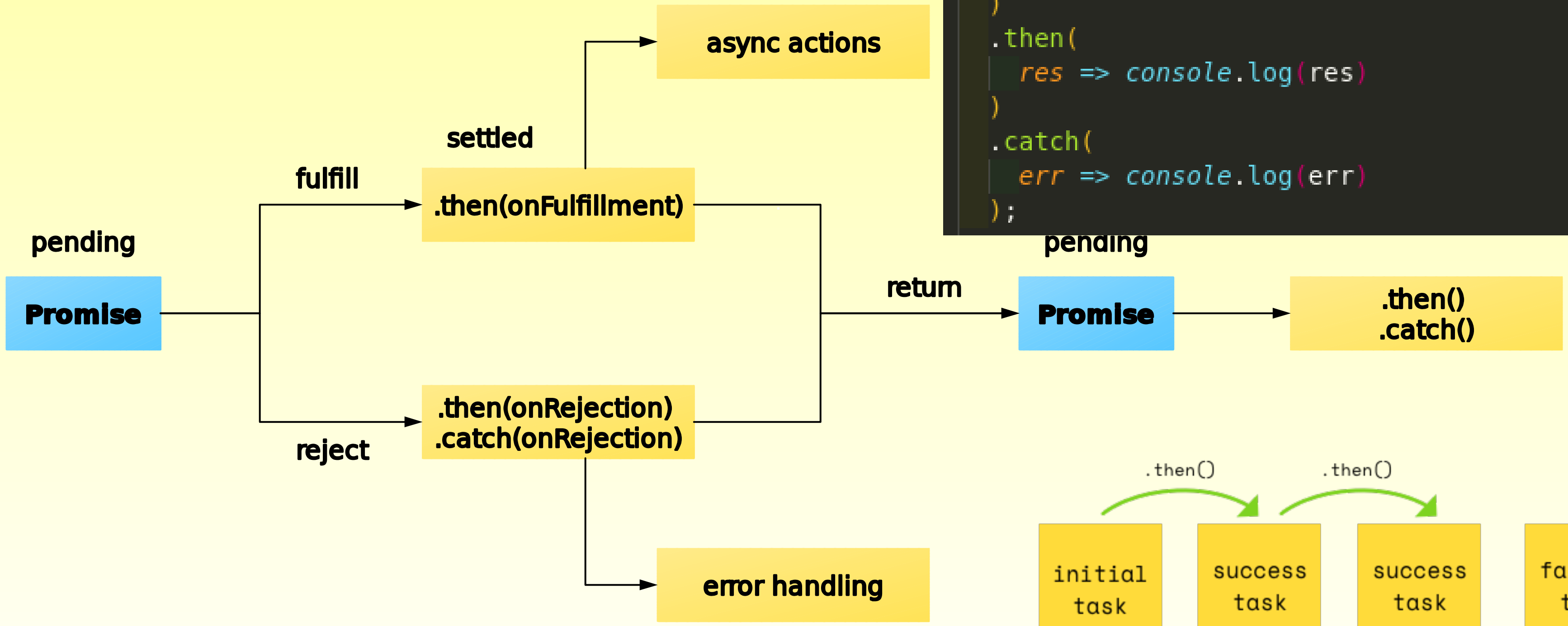


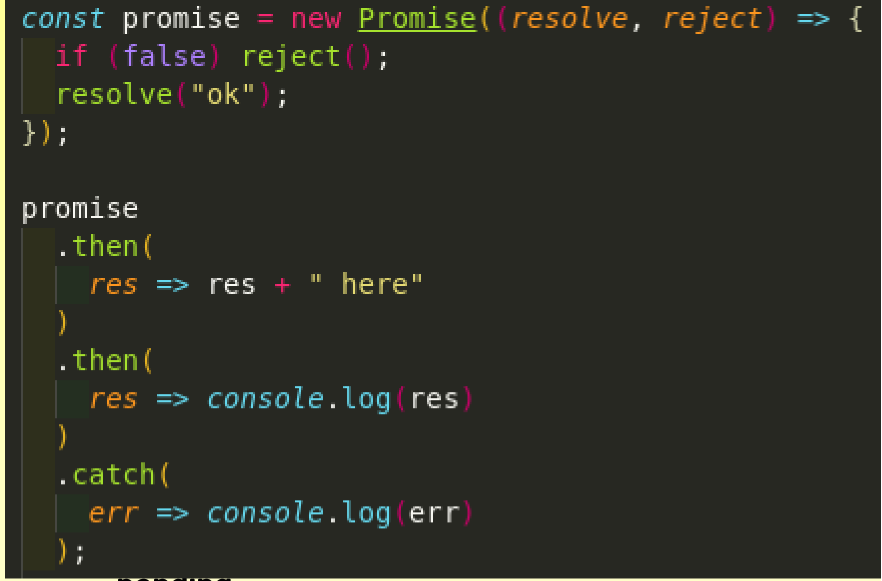




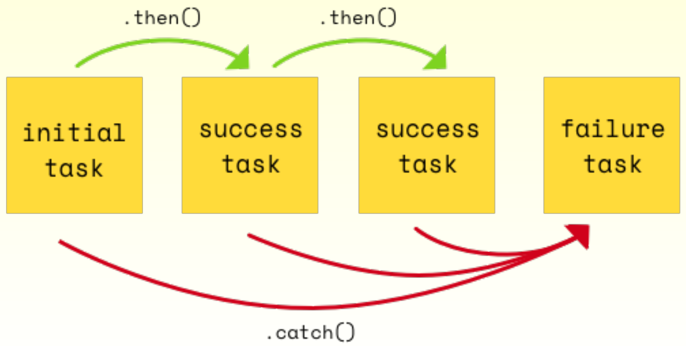
```
const promise = new Promise((resolve, reject) => {
  if (false) reject();
  resolve("ok");
});

promise
  .then(
    res => res + " here"
  )
  .then(
    res => console.log(res)
  )
  .catch(
    err => console.log(err)
  );
```



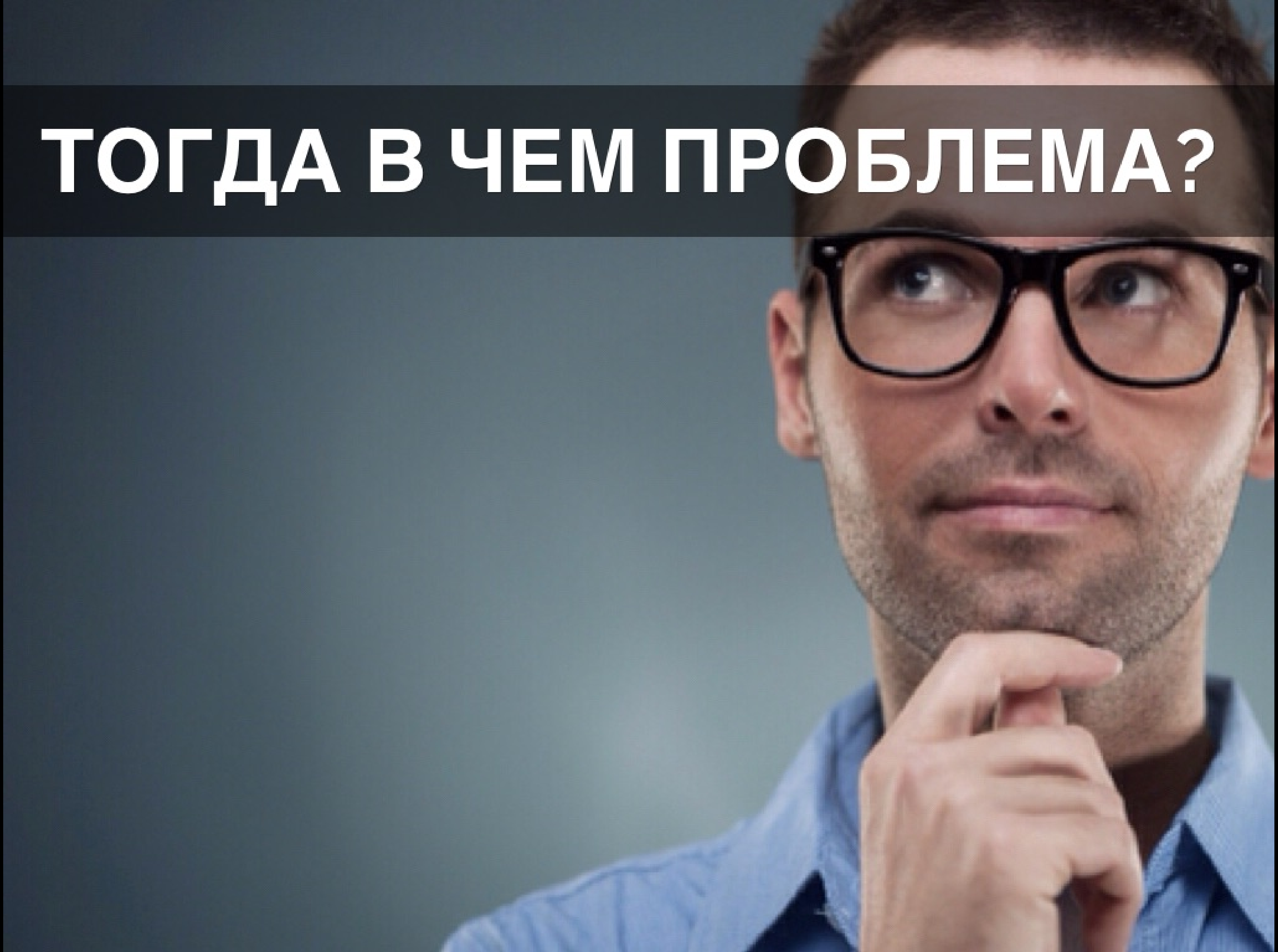




```
const promise = new Promise((resolve, reject) => {
  if (false) reject();
  resolve("ok");
});

promise
  .then(
    res => res + " here"
  )
  .then(
    res => console.log(res)
  )
  .catch(
    err => console.log(err)
  );
```

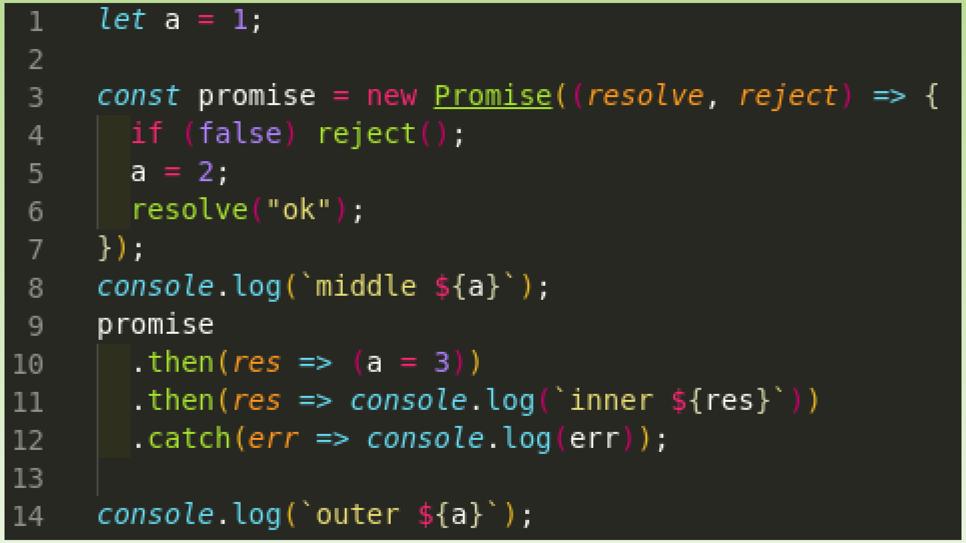


ТОГДА В ЧЕМ ПРОБЛЕМА?



Ассинхронность

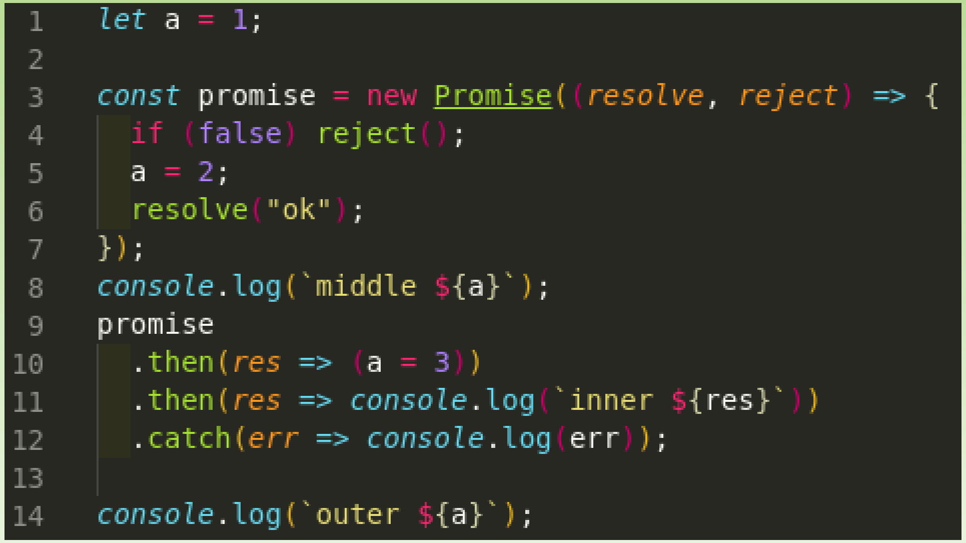
Если есть промис – то забудь про синхронный код!



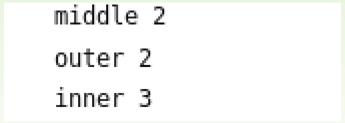
```
let a = 1;

const promise = new Promise((resolve, reject) => {
  if (false) reject();
  a = 2;
  resolve("ok");
});
console.log(`middle ${a}`);
promise
  .then(res => (a = 3))
  .then(res => console.log(`inner ${res}`))
  .catch(err => console.log(err));

console.log(`outer ${a}`);
```

Ассинхронность

Если есть промис – то забудь про синхронный код!



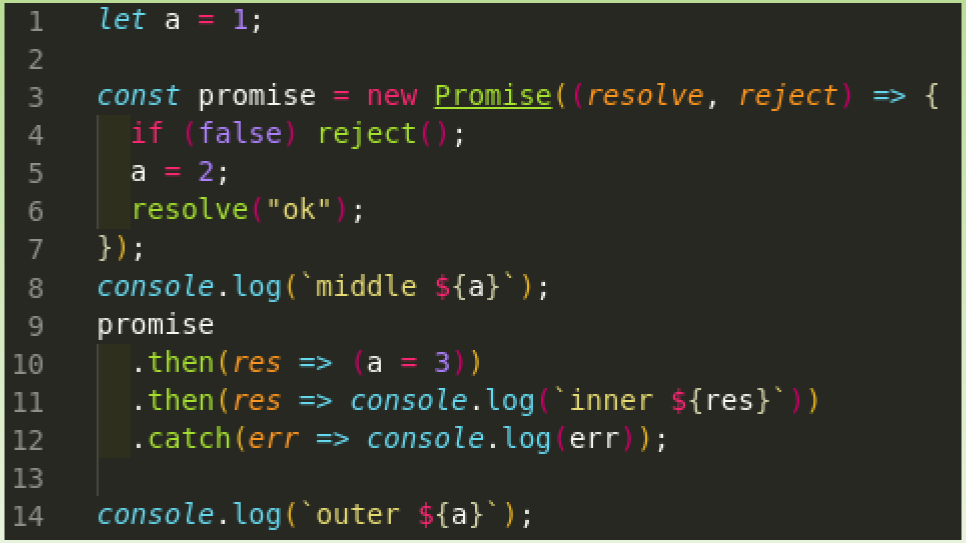
```
let a = 1;

const promise = new Promise((resolve, reject) => {
  if (false) reject();
  a = 2;
  resolve("ok");
});
console.log(`middle ${a}`);
promise
  .then(res => (a = 3))
  .then(res => console.log(`inner ${res}`))
  .catch(err => console.log(err));

console.log(`outer ${a}`);
```



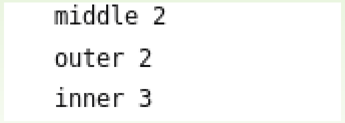
```
middle 2
outer 2
inner 3
```

Ассинхронность

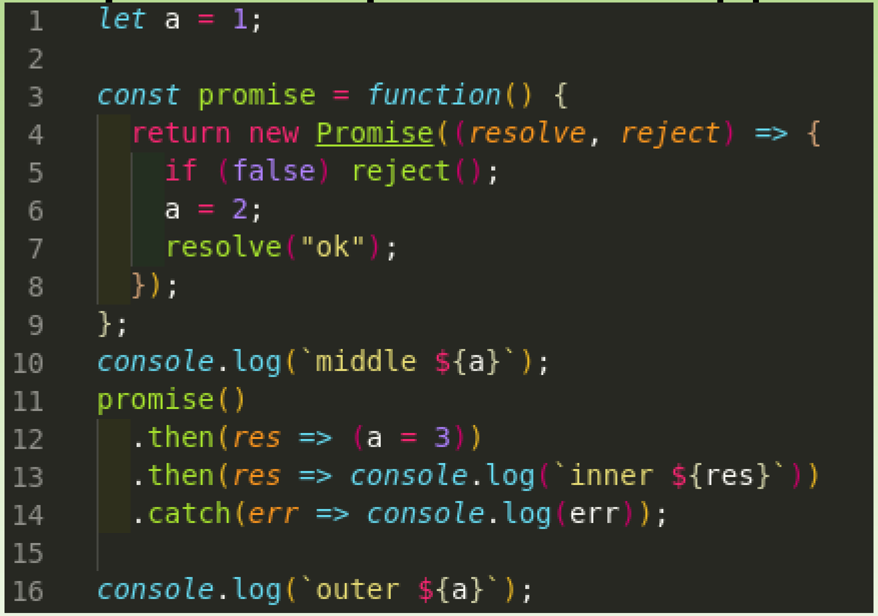
Если есть промис – то забудь про синхронный код!



```javascript
let a = 1;

const promise = new Promise((resolve, reject) => {
  if (false) reject();
  a = 2;
  resolve("ok");
});
console.log(`middle ${a}`);
promise
  .then(res => (a = 3))
  .then(res => console.log(`inner ${res}`))
  .catch(err => console.log(err));

console.log(`outer ${a}`);
```



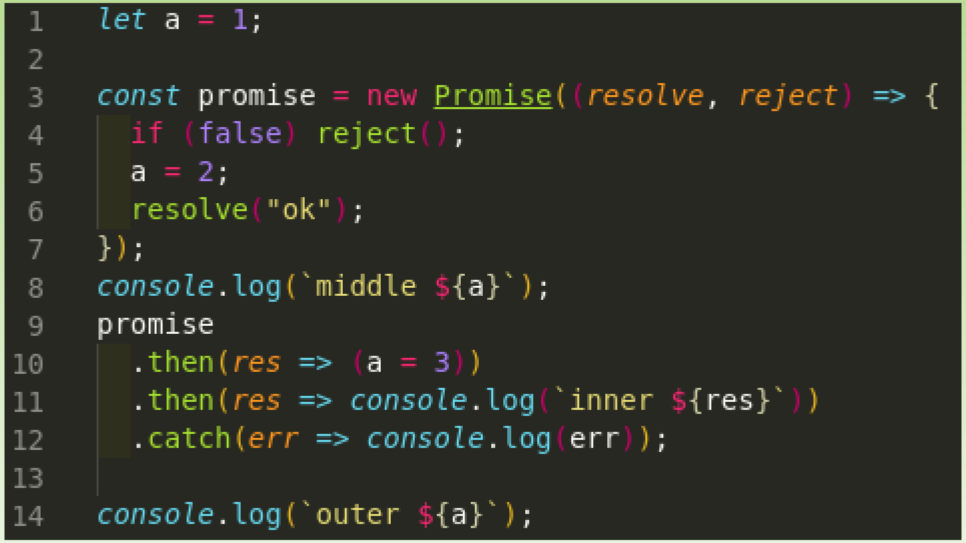
```
middle 2
outer 2
inner 3
```



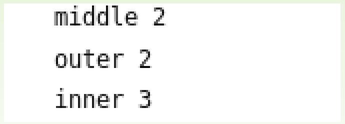
```javascript
let a = 1;

const promise = function() {
  return new Promise((resolve, reject) => {
    if (false) reject();
    a = 2;
    resolve("ok");
  });
};
console.log(`middle ${a}`);
promise()
  .then(res => (a = 3))
  .then(res => console.log(`inner ${res}`))
  .catch(err => console.log(err));

console.log(`outer ${a}`);
```

Ассинхронность

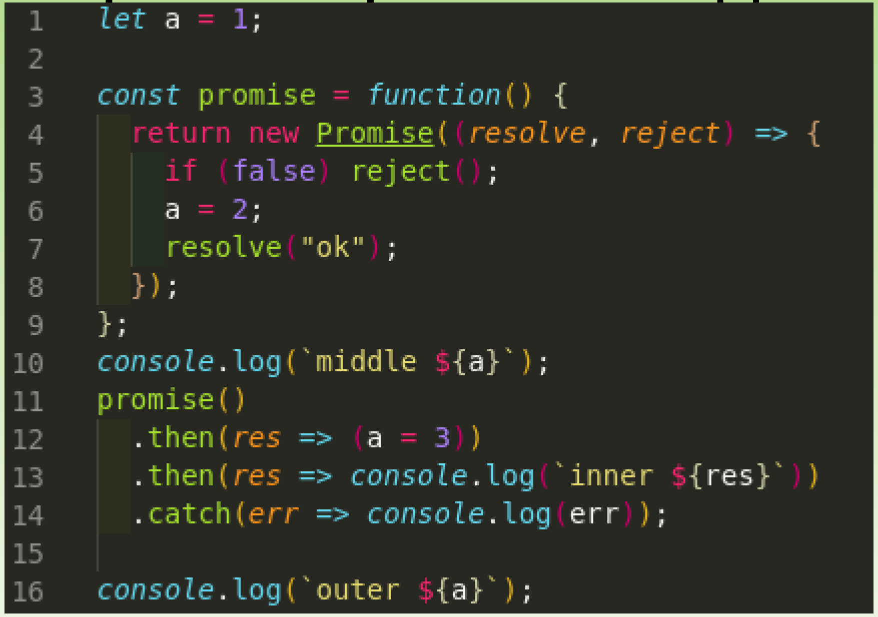
Если есть промис – то забудь про синхронный код!



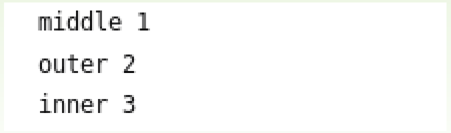
```javascript
let a = 1;

const promise = new Promise((resolve, reject) => {
  if (false) reject();
  a = 2;
  resolve("ok");
});
console.log(`middle ${a}`);
promise
  .then(res => (a = 3))
  .then(res => console.log(`inner ${res}`))
  .catch(err => console.log(err));

console.log(`outer ${a}`);
```



```
middle 2
outer 2
inner 3
```

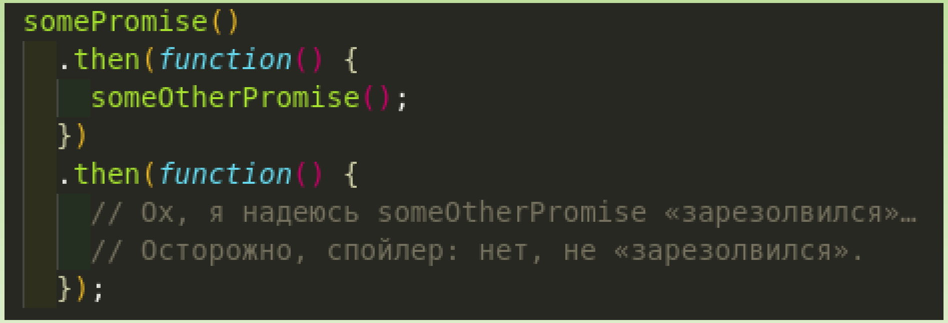


```javascript
let a = 1;

const promise = function() {
  return new Promise((resolve, reject) => {
    if (false) reject();
    a = 2;
    resolve("ok");
  });
};
console.log(`middle ${a}`);
promise()
  .then(res => (a = 3))
  .then(res => console.log(`inner ${res}`))
  .catch(err => console.log(err));

console.log(`outer ${a}`);
```



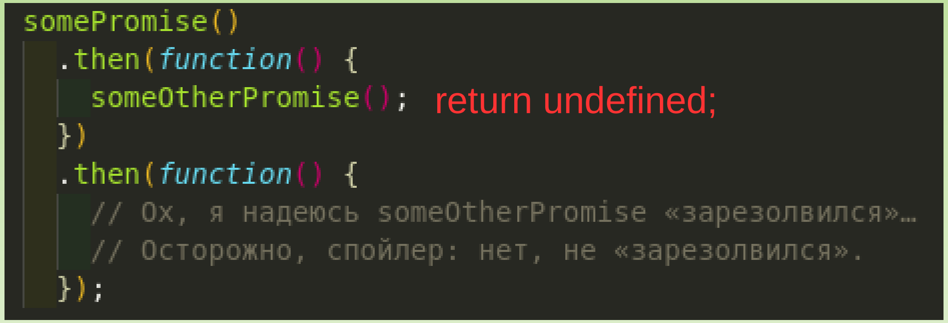
```
middle 1
outer 2
inner 3
```

Ассинхронный код должен возвращать промис



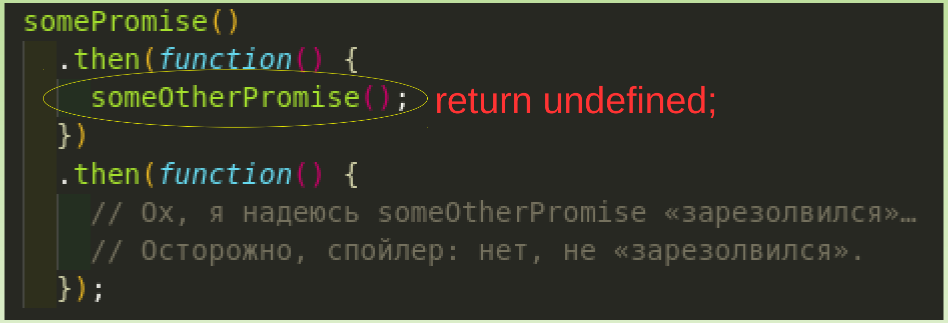
```
somePromise()
  .then(function() {
    someOtherPromise();
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Осторожно, спойлер: нет, не «зарезолвился».
  });
```

Ассинхронный код должен возвращать промис



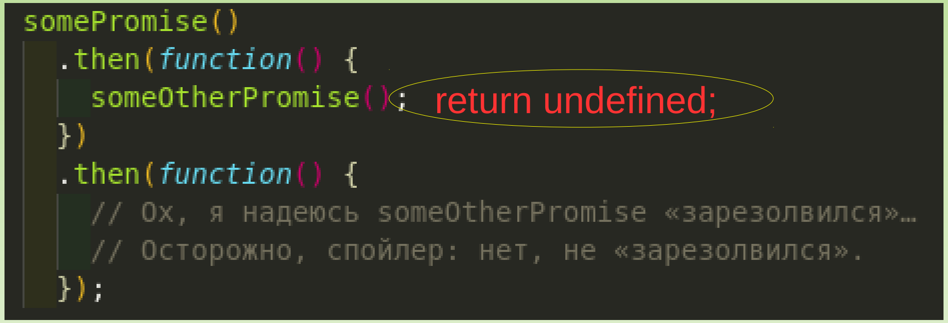
```
somePromise()
  .then(function() {
    someOtherPromise();  return undefined;
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Осторожно, спойлер: нет, не «зарезолвился».
  });
```

Ассинхронный код должен возвращать промис



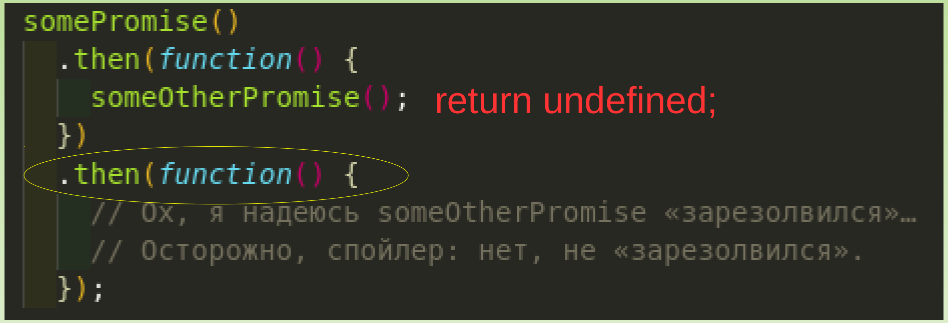
```
somePromise()
  .then(function() {
    someOtherPromise(); return undefined;
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Осторожно, спойлер: нет, не «зарезолвился».
  });
```

Ассинхронный код должен возвращать промис



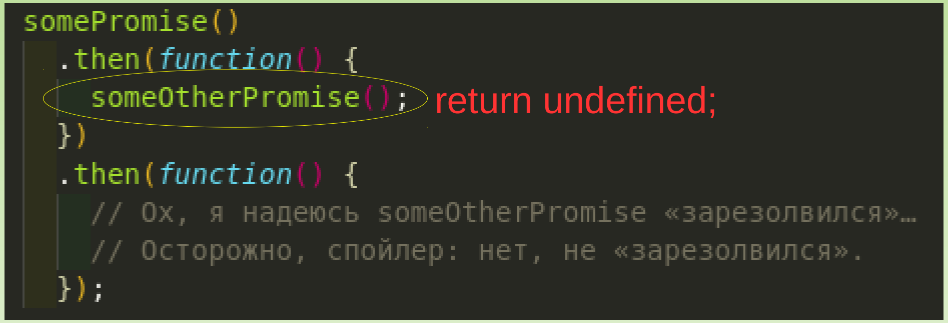
```
somePromise()
  .then(function() {
    someOtherPromise(); return undefined;
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Осторожно, спойлер: нет, не «зарезолвился».
  });
```

Ассинхронный код должен возвращать промис



```
somePromise()
  .then(function() {
    someOtherPromise(); return undefined;
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Осторожно, спойлер: нет, не «зарезолвился».
  });
```

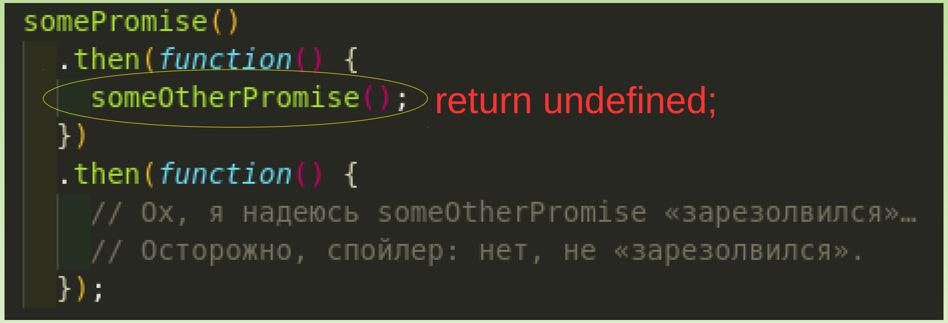
Ассинхронный код должен возвращать промис



```
somePromise()
  .then(function() {
    someOtherPromise(); return undefined;
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Осторожно, спойлер: нет, не «зарезолвился».
  });
```

Ассинхронный код должен возвращать промис

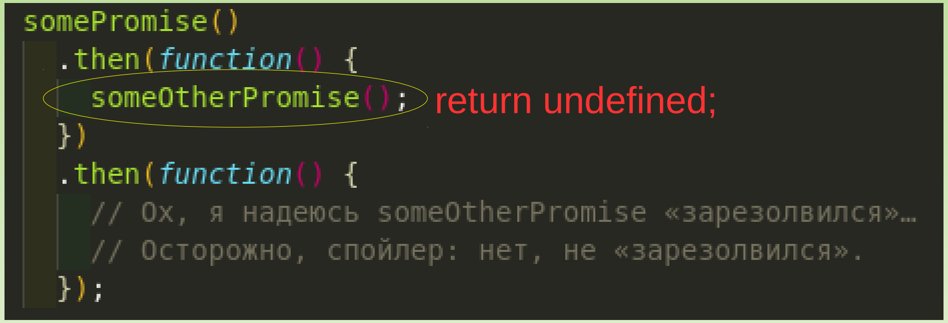
Что мы можем сделать в .then “fulfilled”?



```
somePromise()
  .then(function() {
    someOtherPromise();
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Осторожно, спойлер: нет, не «зарезолвился».
  });
```

return undefined;

Ассинхронный код должен возвращать промис



```
somePromise()
  .then(function() {
    someOtherPromise();
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Осторожно, спойлер: нет, не «зарезолвился».
  });
```

return undefined;

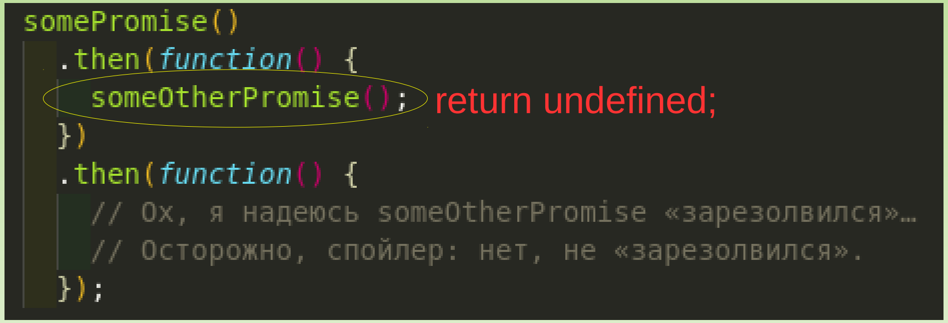
Что мы можем сделать в .then “fulfilled”?

Вернуть (return) другой промис

Вернуть (return) синхронное значение (или undefined)

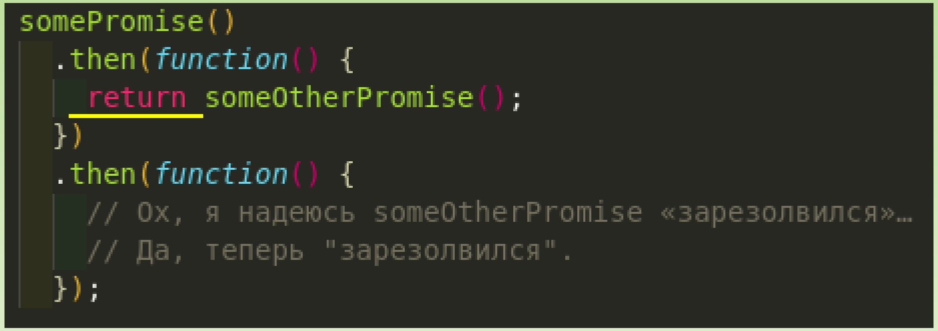
Выдать (throw) синхронную ошибку

Ассинхронный код должен возвращать промис



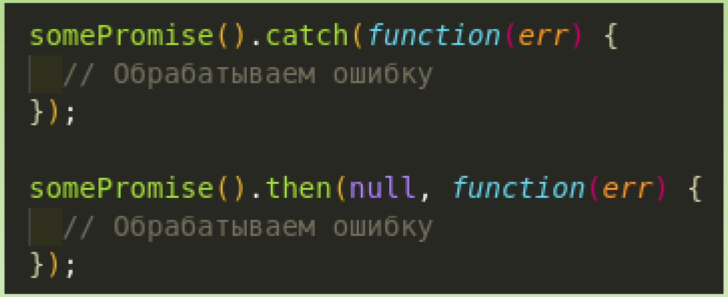
```
somePromise()
  .then(function() {
    someOtherPromise();
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Осторожно, спойлер: нет, не «зарезолвился».
  });
```

return undefined;



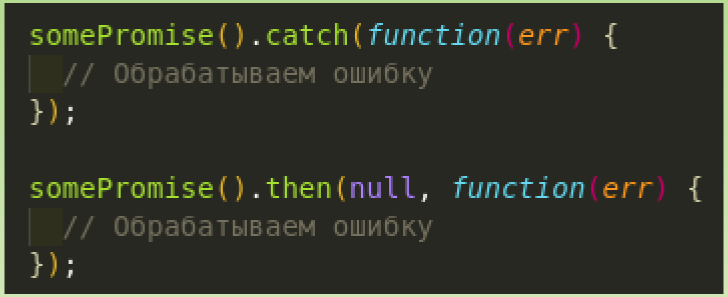
```
somePromise()
  .then(function() {
    return someOtherPromise();
  })
  .then(function() {
    // Ох, я надеюсь someOtherPromise «зарезолвился»…
    // Да, теперь "зарезолвился".
  });
```

Обработка ошибок происходит в следующем .then

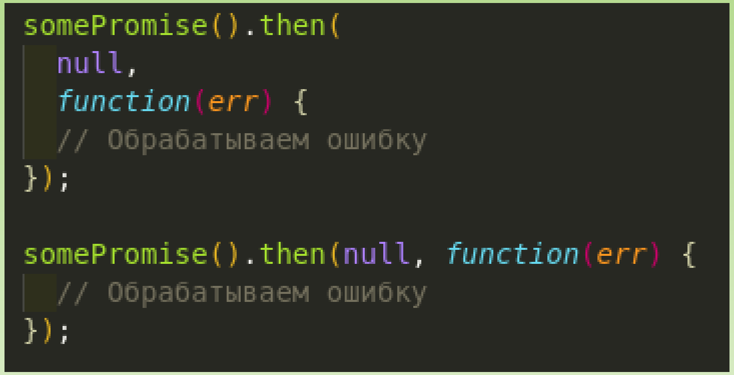


```
somePromise().catch(function(err) {
  // Обрабатываем ошибку
});

somePromise().then(null, function(err) {
  // Обрабатываем ошибку
});
```

Обработка ошибок происходит в следующем .then

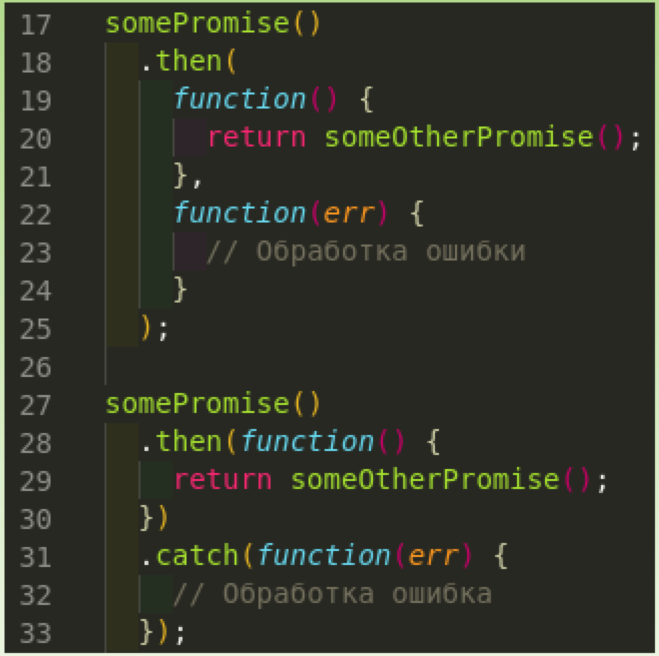


```
somePromise().catch(function(err) {
  // Обрабатываем ошибку
});

somePromise().then(null, function(err) {
  // Обрабатываем ошибку
});
```



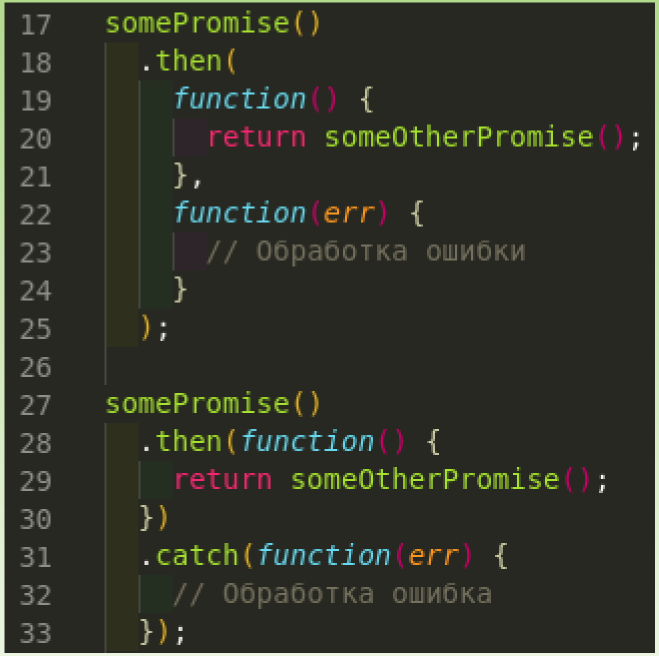
```
somePromise().then(
  null,
  function(err) {
  // Обрабатываем ошибку
});

somePromise().then(null, function(err) {
  // Обрабатываем ошибку
});
```

Обработка ошибок происходит в следующем .then

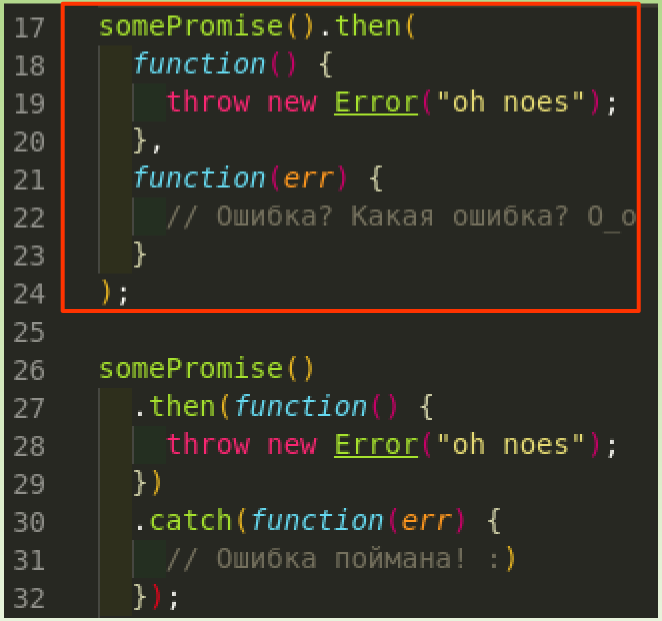


```
somePromise()
  .then(
    function() {
      return someOtherPromise();
    },
    function(err) {
      // Обработка ошибки
    }
  );

somePromise()
  .then(function() {
    return someOtherPromise();
  })
  .catch(function(err) {
    // Обработка ошибка
  });
```

Обработка ошибок происходит в следующем .then

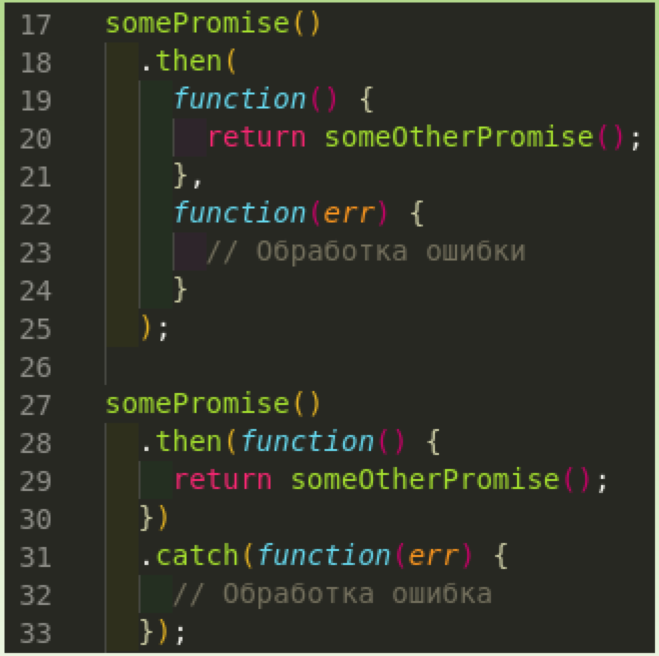


```
somePromise()
  .then(
    function() {
      return someOtherPromise();
    },
    function(err) {
      // Обработка ошибки
    }
  );

somePromise()
  .then(function() {
    return someOtherPromise();
  })
  .catch(function(err) {
    // Обработка ошибка
  });
```

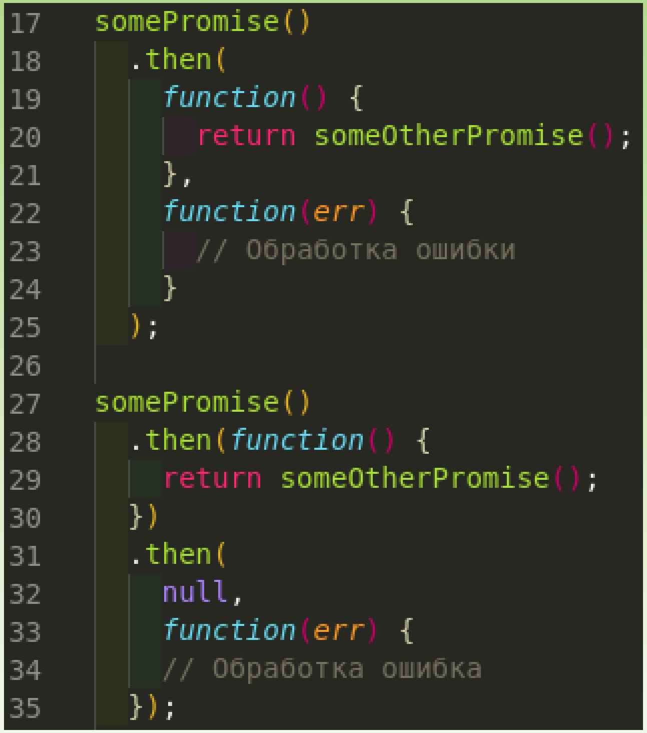


```
somePromise().then(
  function() {
    throw new Error("oh noes");
  },
  function(err) {
    // Ошибка? Какая ошибка? O_o
  }
);

somePromise()
  .then(function() {
    throw new Error("oh noes");
  })
  .catch(function(err) {
    // Ошибка поймана! :)
  });
```

Обработка ошибок происходит в следующем .then

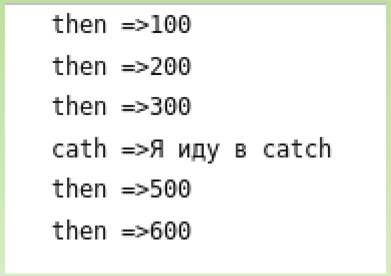


```
someProimse()
  .then(
    function() {
      return someOtherPromise();
    },
    function(err) {
      // Обработка ошибки
    }
  );

someProimse()
  .then(function() {
    return someOtherPromise();
  })
  .catch(function(err) {
    // Обработка ошибка
  });
```



```
someProimse()
  .then(
    function() {
      return someOtherPromise();
    },
    function(err) {
      // Обработка ошибки
    }
  );

someProimse()
  .then(function() {
    return someOtherPromise();
  })
  .then(
    null,
    function(err) {
    // Обработка ошибка
  });
```

.catch тоже передает значения

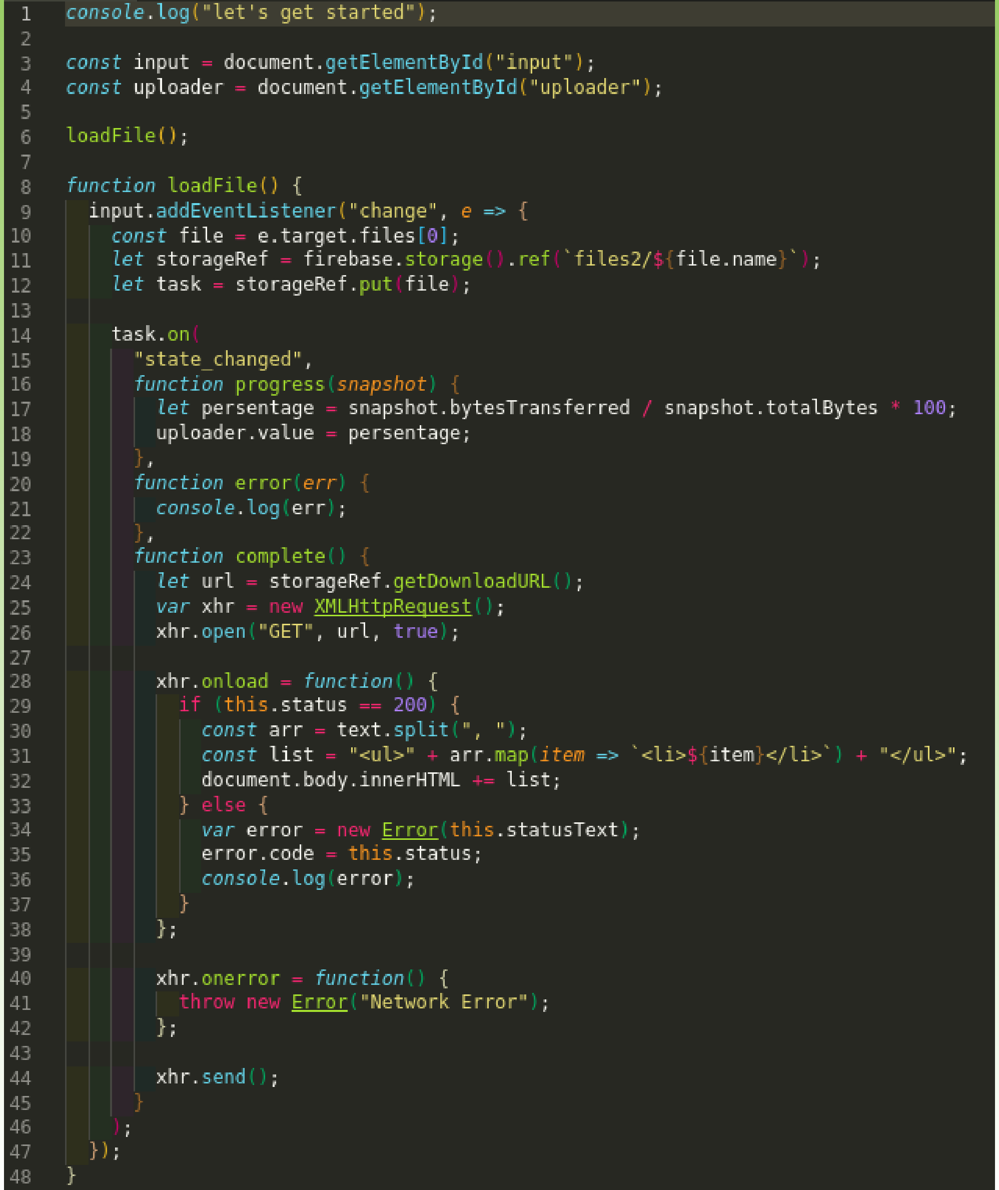


```
then =>100
then =>200
then =>300
cath =>Я иду в catch
then =>500
then =>600
```

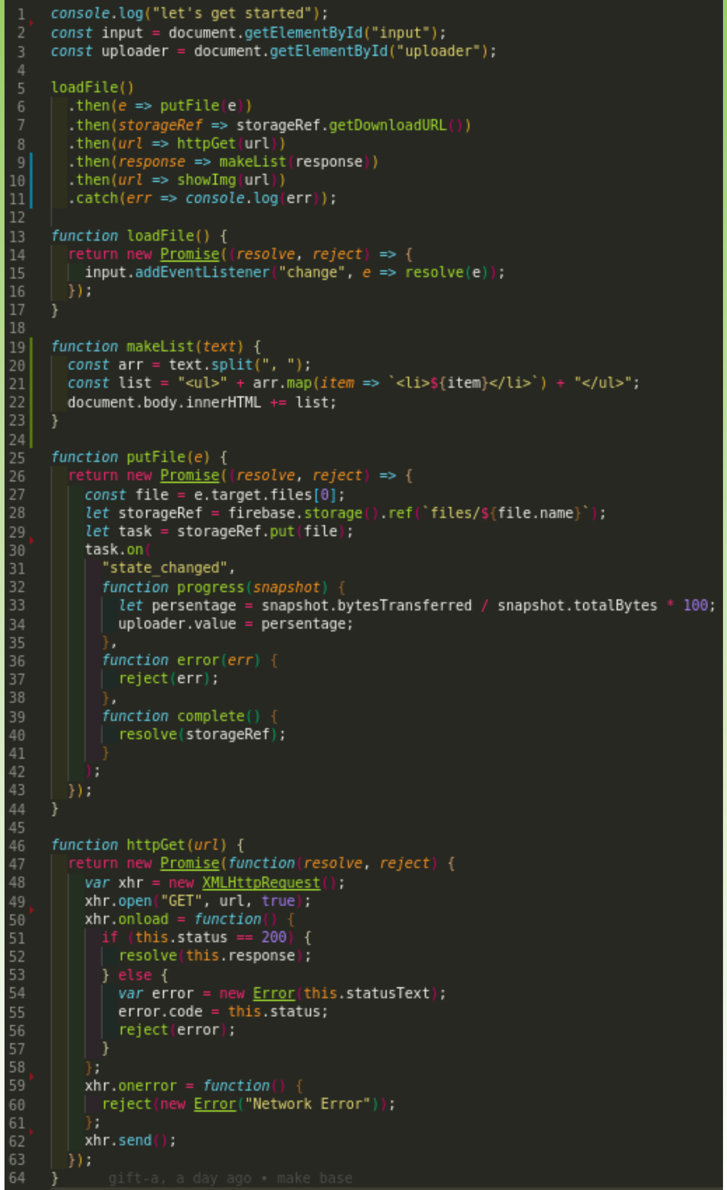
https://gist.github.com/I-0-I/971134a22a425c0717b1ebd21a1979bb

.catch тоже передает значения

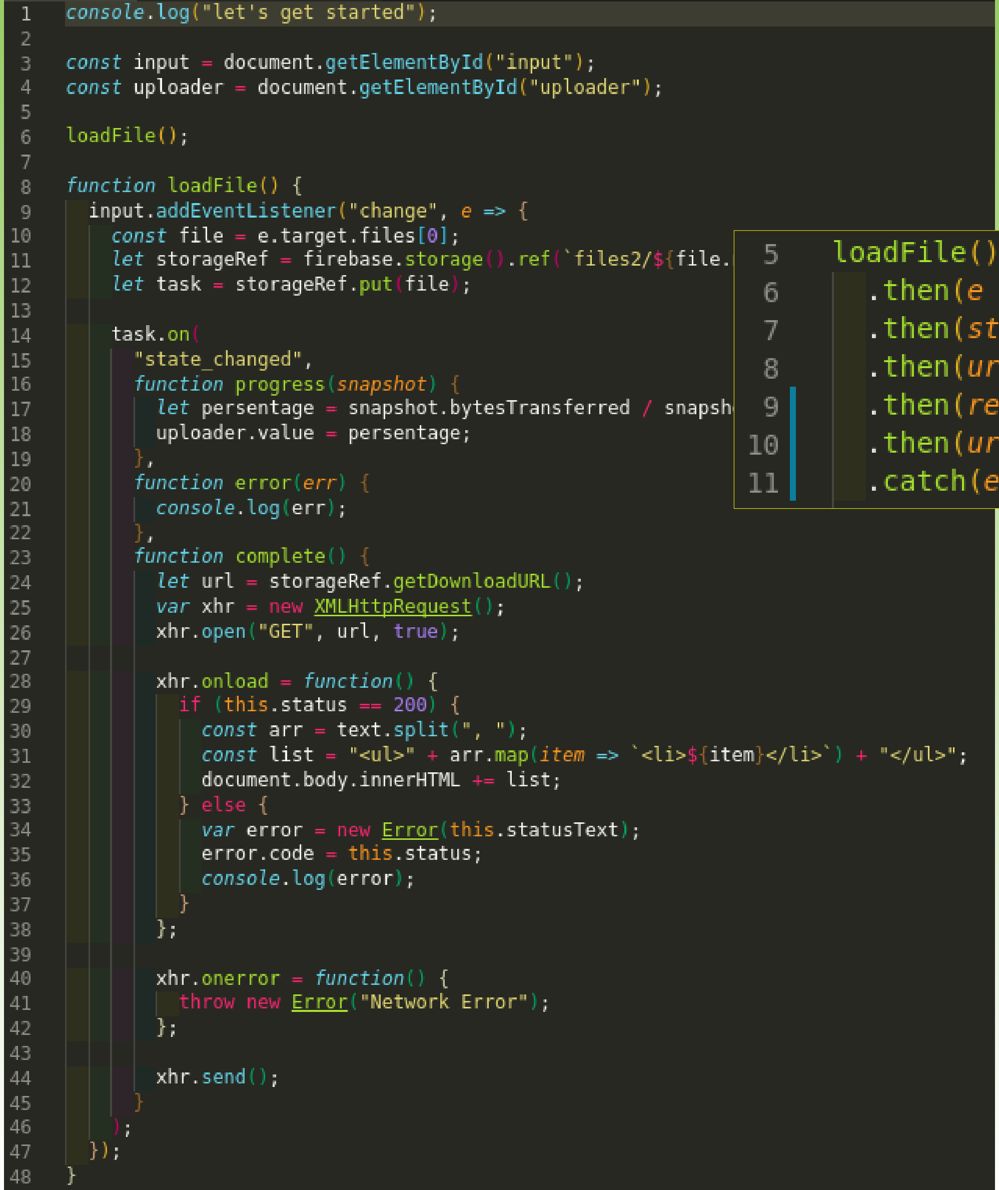
https://gist.github.com/I-0-I/971134a22a425c0717b1ebd21a1979bb



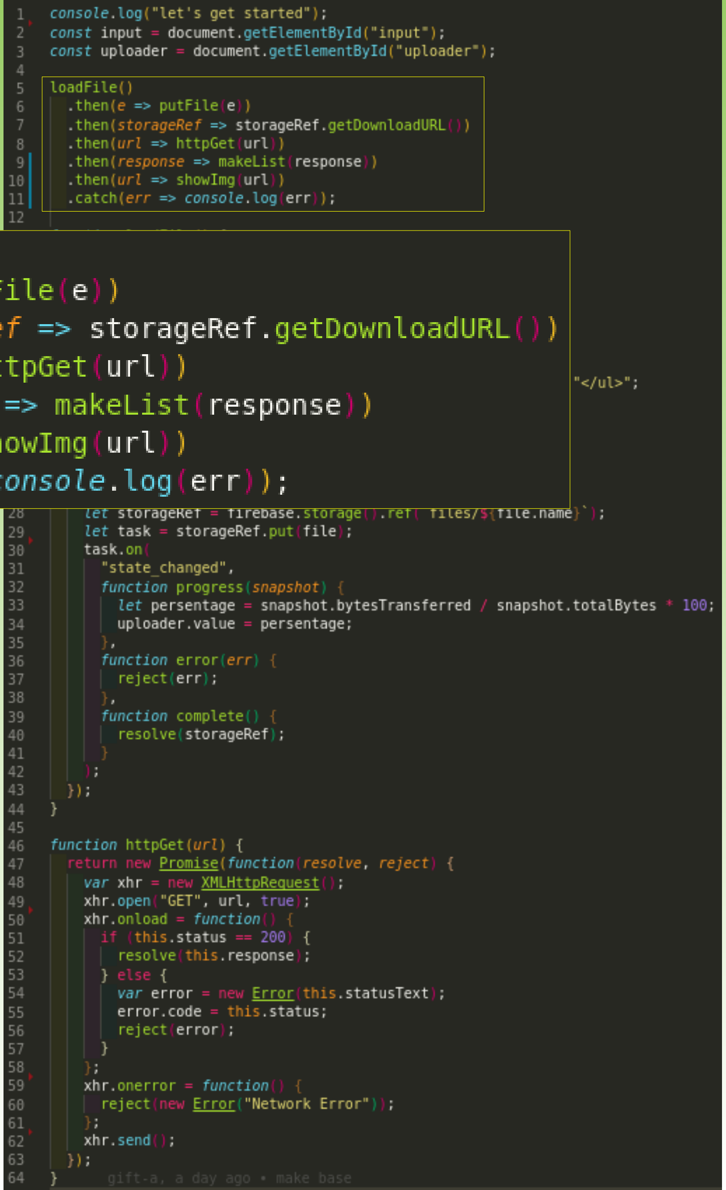
```javascript
console.log("let's get started");

const input = document.getElementById("input");
const uploader = document.getElementById("uploader");

loadFile();

function loadFile() {
  input.addEventListener("change", e => {
    const file = e.target.files[0];
    let storageRef = firebase.storage().ref(`files2/${file.name}`);
    let task = storageRef.put(file);

    task.on(
      "state_changed",
      function progress(snapshot) {
        let persentage = snapshot.bytesTransferred / snapshot.totalBytes * 100;
        uploader.value = persentage;
      },
      function error(err) {
        console.log(err);
      },
      function complete() {
        let url = storageRef.getDownloadURL();
        var xhr = new XMLHttpRequest();
        xhr.open("GET", url, true);

        xhr.onload = function() {
          if (this.status == 200) {
            const arr = text.split(", ");
            const list = "<ul>" + arr.map(item => `<li>${item}</li>`) + "</ul>";
            document.body.innerHTML += list;
          } else {
            var error = new Error(this.statusText);
            error.code = this.status;
            console.log(error);
          }
        };

        xhr.onerror = function() {
          throw new Error("Network Error");
        };

        xhr.send();
      }
    );
  });
}
```



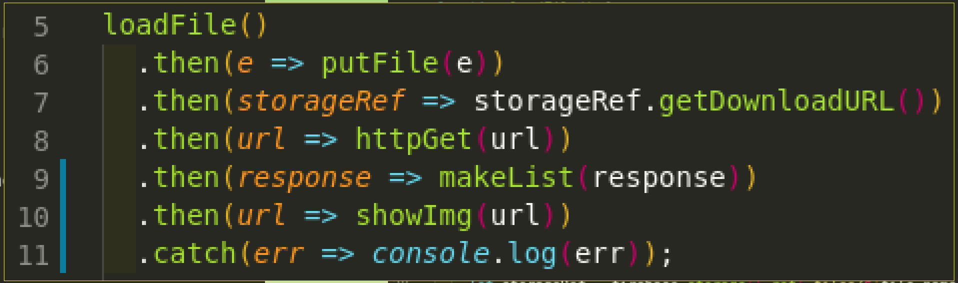
```javascript
console.log("let's get started");
const input = document.getElementById("input");
const uploader = document.getElementById("uploader");

loadFile()
  .then(e => putFile(e))
  .then(storageRef => storageRef.getDownloadURL())
  .then(url => httpGet(url))
  .then(response => makeList(response))
  .then(url => showImg(url))
  .catch(err => console.log(err));

function loadFile() {
  return new Promise((resolve, reject) => {
    input.addEventListener("change", e => resolve(e));
  });
}

function makeList(text) {
  const arr = text.split(", ");
  const list = "<ul>" + arr.map(item => `<li>${item}</li>`) + "</ul>";
  document.body.innerHTML += list;
}

function putFile(e) {
  return new Promise((resolve, reject) => {
    const file = e.target.files[0];
    let storageRef = firebase.storage().ref(`files/${file.name}`);
    let task = storageRef.put(file);
    task.on(
      "state_changed",
      function progress(snapshot) {
        let persentage = snapshot.bytesTransferred / snapshot.totalBytes * 100;
        uploader.value = persentage;
      },
      function error(err) {
        reject(err);
      },
      function complete() {
        resolve(storageRef);
      }
    );
  });
}

function httpGet(url) {
  return new Promise(function(resolve, reject) {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", url, true);
    xhr.onload = function() {
      if (this.status == 200) {
        resolve(this.response);
      } else {
        var error = new Error(this.statusText);
        error.code = this.status;
        reject(error);
      }
    };
    xhr.onerror = function() {
      reject(new Error("Network Error"));
    };
    xhr.send();
  });
}      gift-a, a day ago • make base
```



```javascript
console.log("let's get started");

const input = document.getElementById("input");
const uploader = document.getElementById("uploader");

loadFile();

function loadFile() {
  input.addEventListener("change", e => {
    const file = e.target.files[0];
    let storageRef = firebase.storage().ref(`files2/${file.
    let task = storageRef.put(file);

    task.on(
      "state_changed",
      function progress(snapshot) {
        let persentage = snapshot.bytesTransferred / snapsh
        uploader.value = persentage;
      },
      function error(err) {
        console.log(err);
      },
      function complete() {
        let url = storageRef.getDownloadURL();
        var xhr = new XMLHttpRequest();
        xhr.open("GET", url, true);

        xhr.onload = function() {
          if (this.status == 200) {
            const arr = text.split(", ");
            const list = "<ul>" + arr.map(item => `<li>${item}</li>`) + "</ul>";
            document.body.innerHTML += list;
          } else {
            var error = new Error(this.statusText);
            error.code = this.status;
            console.log(error);
          }
        };

        xhr.onerror = function() {
          throw new Error("Network Error");
        };

        xhr.send();
      }
    );
  });
}
```



```javascript
console.log("let's get started");
const input = document.getElementById("input");
const uploader = document.getElementById("uploader");

loadFile()
  .then(e => putFile(e))
  .then(storageRef => storageRef.getDownloadURL())
  .then(url => httpGet(url))
  .then(response => makeList(response))
  .then(url => showImg(url))
  .catch(err => console.log(err));

"</ul>";
      let storageRef = firebase.storage().ref(`files/${file.name}`);
      let task = storageRef.put(file);
      task.on(
        "state_changed",
        function progress(snapshot) {
          let persentage = snapshot.bytesTransferred / snapshot.totalBytes * 100;
          uploader.value = persentage;
        },
        function error(err) {
          reject(err);
        },
        function complete() {
          resolve(storageRef);
        }
      );
    });
}

function httpGet(url) {
  return new Promise(function(resolve, reject) {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", url, true);
    xhr.onload = function() {
      if (this.status == 200) {
        resolve(this.response);
      } else {
        var error = new Error(this.statusText);
        error.code = this.status;
        reject(error);
      }
    };
    xhr.onerror = function() {
      reject(new Error("Network Error"));
    };
    xhr.send();
  });
}         gift-a, a day ago • make base
```



```javascript
loadFile()
  .then(e => putFile(e))
  .then(storageRef => storageRef.getDownloadURL())
  .then(url => httpGet(url))
  .then(response => makeList(response))
  .then(url => showImg(url))
  .catch(err => console.log(err));
```

Links

http://latentflip.com/loupe/?code=!!!PGJ1dHRvbj5DbGljayBtZSE8L2J1dHRvbj4%3D

https://www.ecma-international.org/ecma-262/6.0/#sec-promise-objects
https://vimeo.com/74925301
https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Promise

https://habrahabr.ru/company/mailru/blog/269465/
https://habrahabr.ru/post/312670/