Asynchronous Programming and Promises

Promises. Async / Await.

SoftUni Team
Technical Trainers

Software University
http://softuni.bg

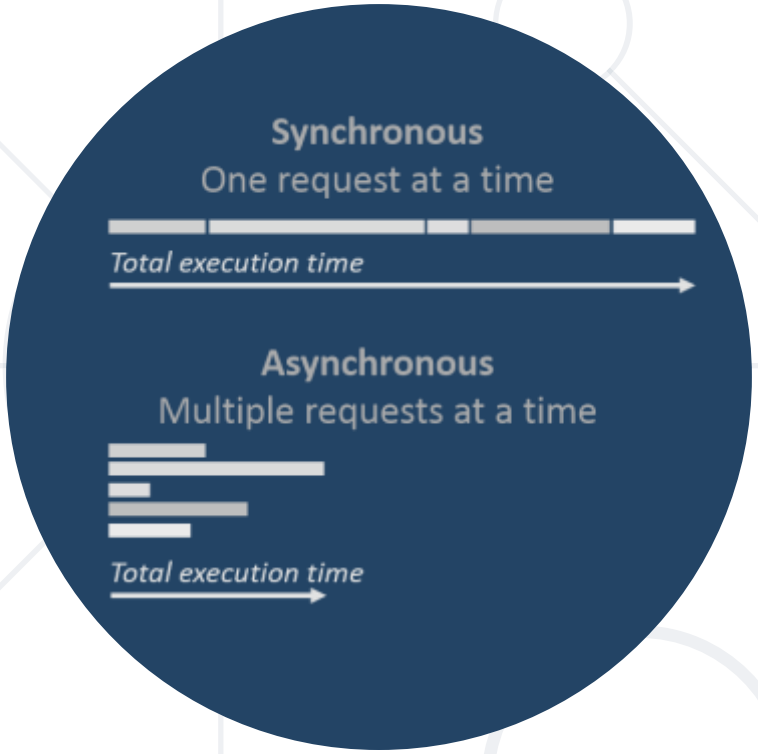
Table of Contents

Have a Question?

SoftUni Foundation

sli.do

#JS-CORE



Asynchronous Programming

Synchronous vs Asynchronous

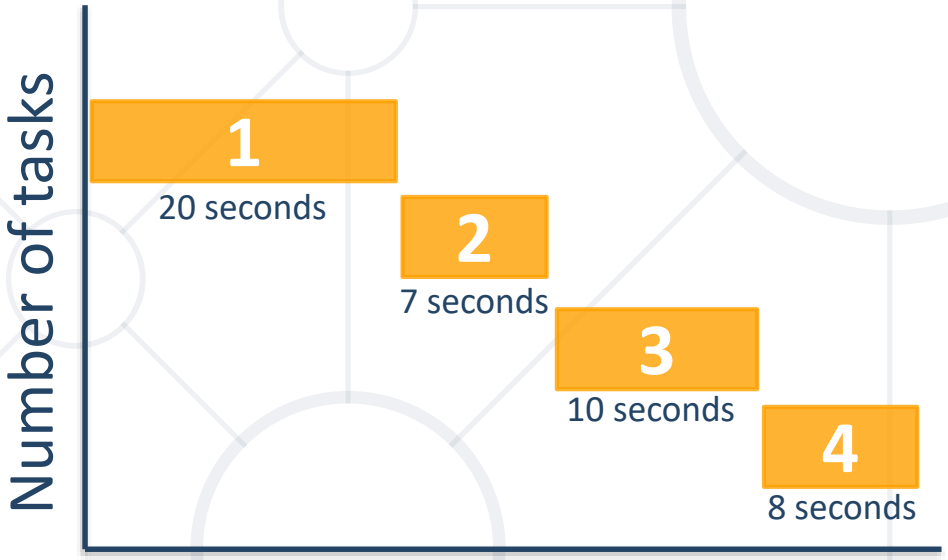
Asynchronous Programming

- Deals with the needs to run several tasks in **parallel**
- Asynchronous Programming is based on callbacks
- There can be **asynchronous code**, but it is **generally single-threaded**
- Handled with:
  - **Promises**
  - **Async** / **Await** pattern

Asynchronous Programming

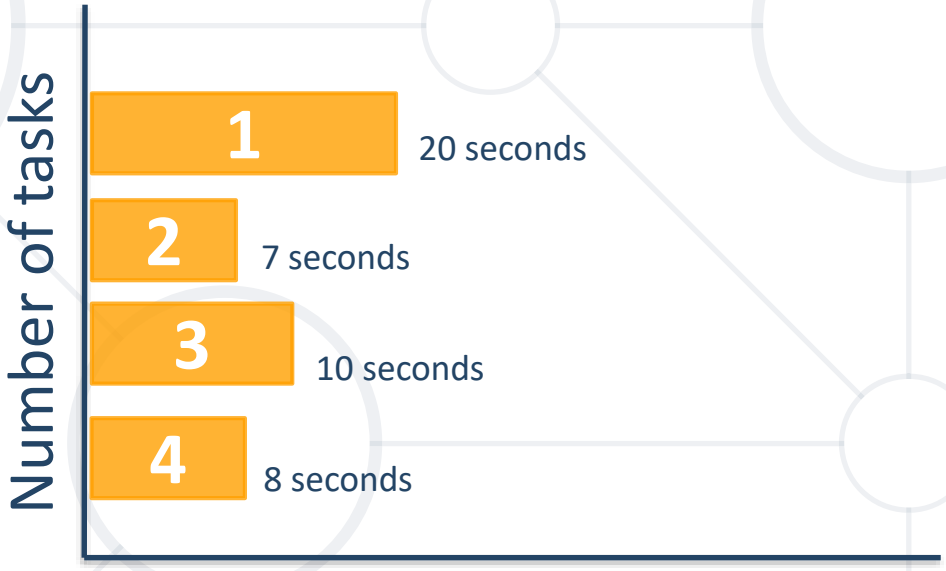
SoftUni Foundation

Synchronous



Total time taken by the tasks:
45 seconds

Asynchronous



Total time taken by the tasks:
20 seconds

Asynchronous Programming - Example

The following commands will be executed as follows:

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

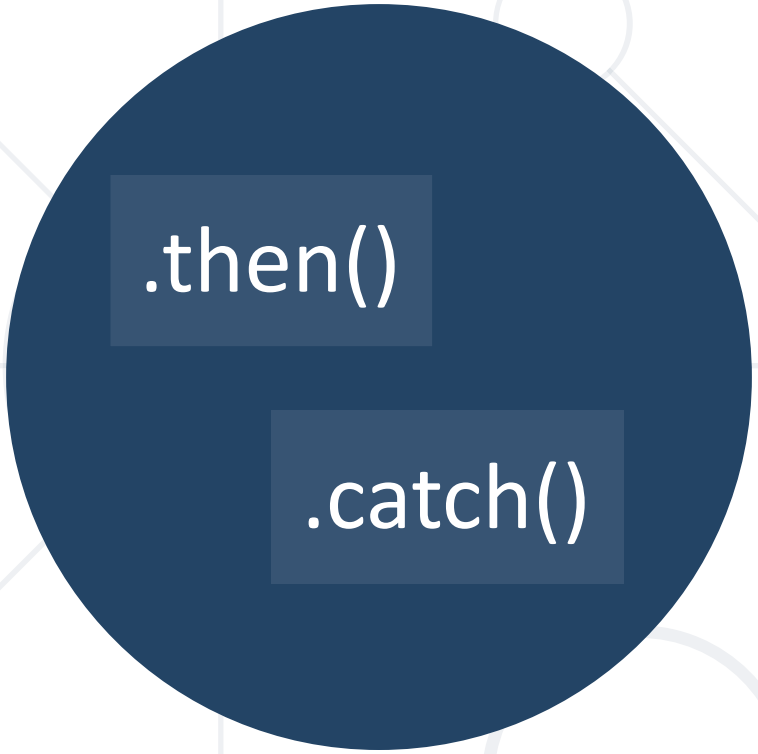
```
setTimeout(function() {
  console.log("Goodbye!");
}, 2000);
```

```
console.log("Hello again!");
```

```
// Hello.
```

```
// Hello again!
```

```
// Goodbye!
```



Promises

Objects Holding Asynchronous Operations

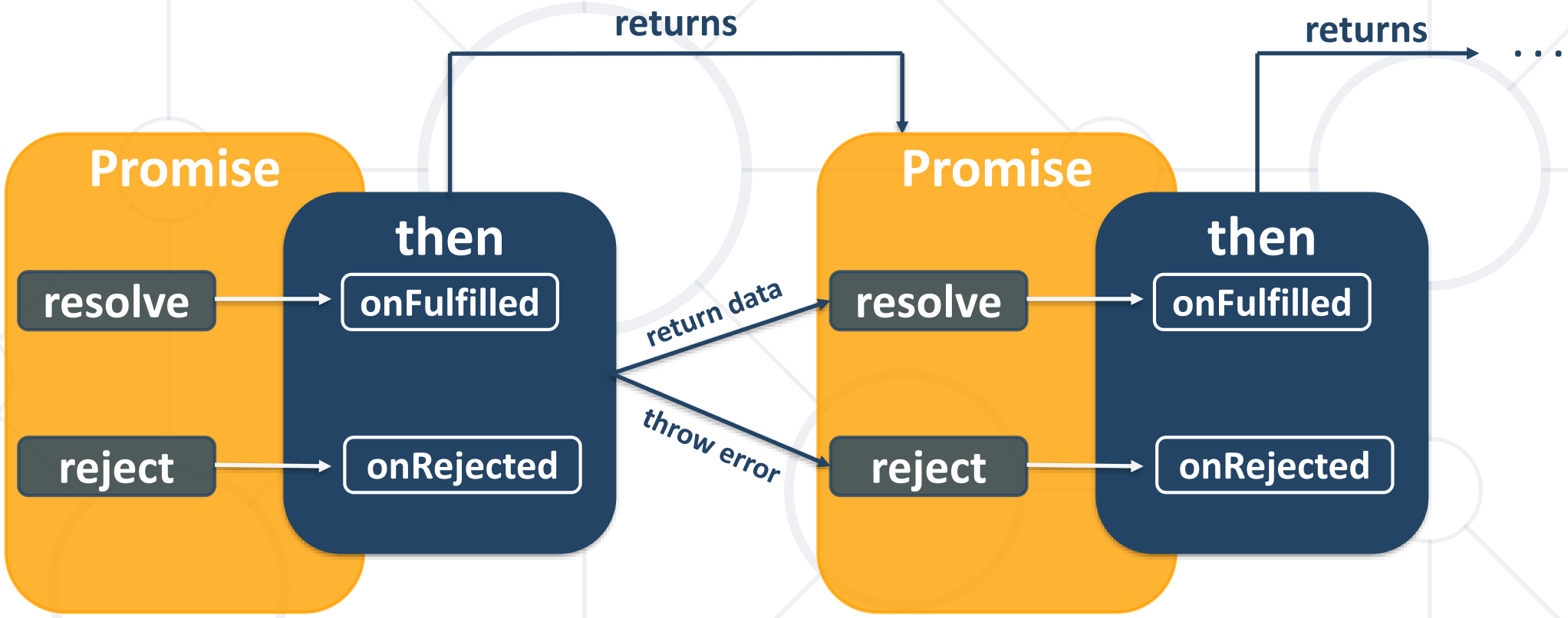
What is a Promise?

An object holding an asynchronous operation

- States:
  - **Pending** - operation still running (unfinished)
  - **Fulfilled** - operation finished (and the result is available)
  - **Failed** - operation is failed (and an error is available)
- Promises use the **Promise** object

```
new Promise(executor);
```

What is a Promise?



Promise Methods

- `Promise.reject(reason)`
  - Returns an object that is rejected with the given reason
- `Promise.resolve(value)`
  - Returns an object that is resolved with the given value
- `Promise.all(iterable)`
  - Returns a promise that either fulfills when **all** of the promises **have fulfilled** or rejects as soon as **one** of them **rejects**

Promise.then() - Example

```
console.log('Before promise');
```

```
new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('done');
  }, 500);
})
.then(function(result) {
  console.log('Then returned: ' + result);
});
```

Resolved after 500 ms

```
console.log('After promise');
```

```
// Before promise
```

```
// After promise
```

```
// Then returned: done
```

Promise.catch() - Example

```
console.log('Before promise');
```

```
new Promise(function(resolve, reject) {
  setTimeout(function() {
    reject('fail');
  }, 500);
})
.then(function(result) { console.log(result); });
.catch(function(error) { console.log(error); });
```

Rejected after 500 ms

```
console.log('After promise');
```

```
// Before promise
```

```
// After promise
```

```
// fail
```

Promises with jQuery AJAX

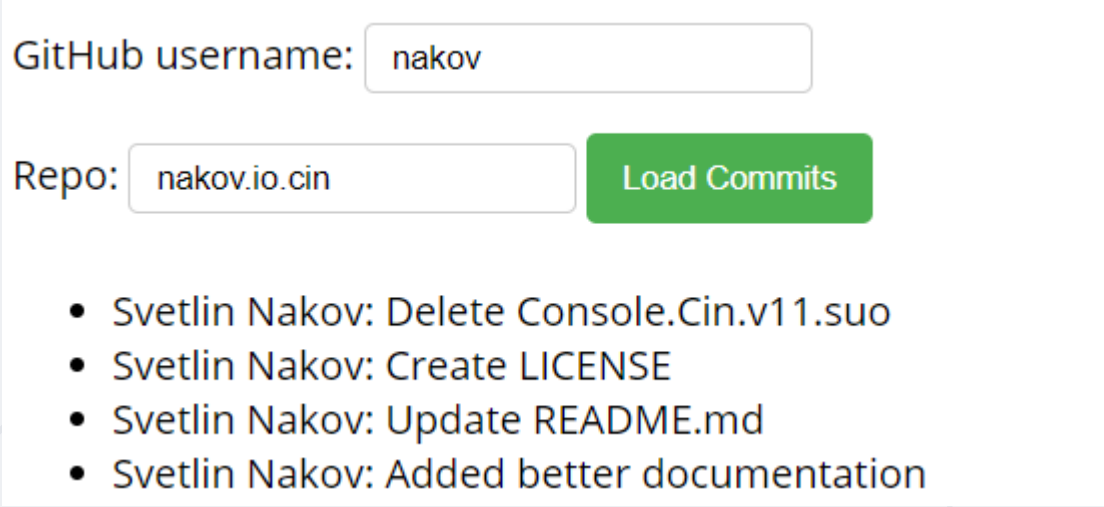
jQuery Promise

The **Deferred object** is a **chainable** utility object.

- Can register **multiple callbacks** into callback queues
- Invoke **callback queues** and relay the success or failure state of a function
- Is **thenable** - can be casted to **native Promise**
- Some of the arguments passed to **then()** method will be **discarded**

Problem: Load GitHub Commits with AJAX

```
GitHub username:
<input type="text" id="username" value="nakov" /> <br>
Repo: <input type="text" id="repo" value="nakov.io.cin" />
<button onclick="loadCommits()">Load Commits</button>
<ul id="commits"></ul>
<script>
  function loadCommits() {
     // AJAX call …
  }
</script>
```



Solution: Load GitHub Commits with AJAX

```
function loadCommits() {
  $("#commits").empty();
  let url = "https://api.github.com/repos/" +
    $("#username").val() + "/" +
    $("#repo").val() + "/commits";
  $.get(url)
    .then(displayCommits)
    .catch(displayError);
  function displayCommits(commits) { … }
  function displayError(err) { … }
}
```

jQuery AJAX methods return promises

Solution: Load GitHub Commits with AJAX (2)

```
function displayCommits(commits) {
  for (let commit of commits)
    $("#commits").append($("<li>").text(
      commit.commit.author.name + ": " +
      commit.commit.message
    ));
}
function displayError(err) {
  $("#commits").append($("<li>").text("Error: " +
    err.status + ' (' + err.statusText + ')'));
}
```

Check your solution here: https://judge.softuni.bg/Contests/1570

Problem: Blog

Create a **Kinvey app** and then add **user** "peter" with **password** "p"

- Create **comments** "Com1a" and "Com1b" for "Post1
- Create **comments** "Com2a", "Com2b" and "Com2c" for "Post2“
- Display **all posts** and view the **selected** post along with its **comments**

Solution: Blog - Create First Post

Insert your Kinvey App ID here

```
POST /appdata/kid_S1htVfcmm/posts/ HTTP/1.1
Host: baas.kinvey.com
Authorization: Basic cGV0ZXI6cA==
Content-Type: application/json

{ "title":"Post1", "body":"Post #1 body" }
```

Base64(user:pass)

```
{ "title": "Post3", "body": "Post #3 body", …,
  "_id": "5c9a3e3b13ebac4e57c0451e" }
```

Remember the post _id

Solution: Blog - Create Comments

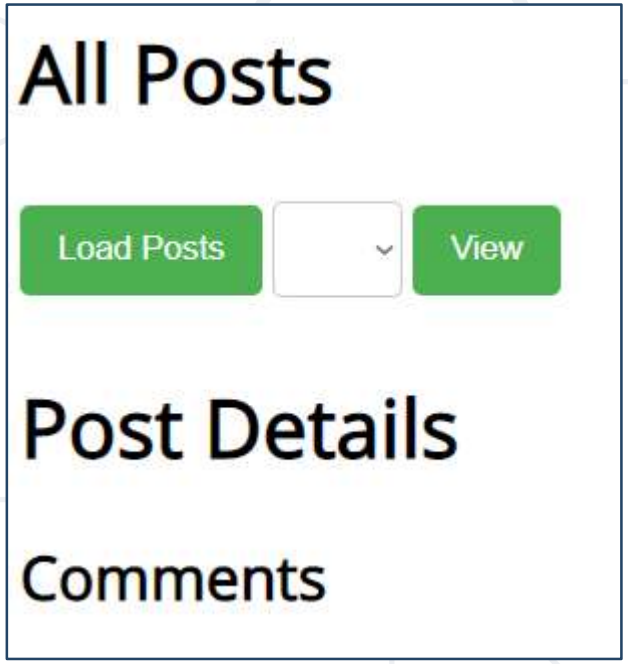
```
POST /appdata/kid_S1htVfcmm/comments/ HTTP/1.1
Host: baas.kinvey.com
Authorization: Basic cGV0ZXI6cA==
Content-Type: application/json

{ "text":"Com1a", "post_id":"5c9a3e3b13ebac4e57c0451e" }
```

Use post _id from the previous request

```
POST /appdata/kid_S1htVfcmm/comments/ HTTP/1.1
Host: baas.kinvey.com
Authorization: Basic cGV0ZXI6cA==
Content-Type: application/json

{ "text":"Com2a", "post_id":"5c9a3e3b13ebac4e57c0451e" }
```

Solution: Blog - HTML Code

```html
<script src="jquery-3.1.1.min.js"></script>
<script src="blog.js"></script>

<h1>All Posts</h1>
<button id="btnLoadPosts">Load</button>
<select id="posts"></select>
<button id="btnViewPost">View</button>

<h1 id="post-title">Post Details</h1>
<ul id="post-body"></ul>
<h2>Comments</h2>
<ul id="post-comments"></ul>
```



Solution: Blog - JS Code

```
$(document).ready(function() {
  const kinveyAppId = "kid_S1htVfcmm";
  const serviceUrl = "https://baas.kinvey.com/appdata/" +
    kinveyAppId;
  const kinveyUsername = "peter";
  const kinveyPassword = "p";
  const base64auth = btoa(kinveyUsername + ":" +
    kinveyPassword);
  const authHeaders = { "Authorization": "Basic " + base64auth
};
  $("#btnLoadPosts").click(loadPostsClick);
  $("#btnViewPost").click(viewPostClick);
  function loadPostsClick() { … }
  function viewPostClick() { … }
  …
});
```

Solution: Blog - Load Posts

```
function loadPostsClick() {
  let loadPostsRequest = {
    url: serviceUrl + "/posts",
    headers: authHeaders,
  };
  $.ajax(loadPostsRequest)
    .then(displayPosts)
    .catch(displayError);
}
```

Solution: Blog - Display Posts as Options

```
function displayPosts(posts) {
  $("#posts").empty();
  for (let post of posts) {
    let option = $("<option>")
      .text(post.title)
      .val(post._id);
    $("#posts").append(option);
  }
}
```

```
▼<select id="posts">
    <option value="5c9a3e3b13ebac4e57c0451e">Post 1</option>
    <option value="5c9a3e4aa698481aa90621e6">Post 2</option>
  </select>
```

Solution: Blog - Handle AJAX Errors

```
function displayError(err) {
  let errorDiv = $("<div>").text("Error: " +
    err.status + ' (' + err.statusText + ')');
  $(document.body).prepend(errorDiv);
  setTimeout(function() {
    $(errorDiv).fadeOut(function() {
      $(errorDiv).remove();
    });
  }, 3000);
}
```

Solution: Blog - Load Post Comments Query

Kinvey allows querying collections:

```
https://baas.kinvey.com/appdata/kid_S1htVfcmm/comments
?query={"post_id":"5c9a3e3b13ebac4e57c0451e"}
```

GET https://baas.kinvey.com/appdata/kid_S1htVfcmm/comments?query={"post_id":"5c9a3e3b13ebac4e57c0451e"}...

☑	Authorization	Basic cGV0ZXI6cA==
☑	Content-Type	application/json
	Key	Value

Body Cookies Headers (9) Test Results

Pretty Raw Preview JSON


```
1 [
2     {
3         "_id": "5c9a3f97fa20820788c0b299",
4         "text": "Com1a",
5         "post_id": "5c9a3e3b13ebac4e57c0451e",
```

Solution: Blog - [View Post] Button Click

```
function viewPostClick() {
  let selectedPostId = $("#posts").val();
  if (!selectedPostId) return;
  let requestPosts = $.ajax({
    url: serviceUrl + "/posts/" + selectedPostId,
    headers: authHeaders });
  let requestComments = $.ajax({ url: serviceUrl +
`/comments/?query={"post_id":"${selectedPostId}"}`,
    headers: authHeaders });
  Promise.all([requestPosts, requestComments])
    .then(displayPostWithComments)
    .catch(displayError);
}
```

Solution: Blog - Display Post with its Comments

```
function displayPostWithComments([post, comments]) {
  $("#post-title").text(post.title);
  $("#post-body").text(post.body);
  $("#post-comments").empty();
  for (let comment of comments) {
    let commentItem = $("<li>")
      .text(comment.text);
    $("#post-comments")
      .append(commentItem);
  }
}
```

Check your solution here: https://judge.softuni.bg/Contests/1570

Async / Await

Simplified Promises

Async Functions

Operate asynchronously via the **event loop**

Contain an **await** expression that:

- Is only valid inside **async functions**
- **Pauses** the execution
- **Waits** for the **Promise's resolution**

Async / **Await** is similar to combining **generators and promises**

Async Functions (2)

```
function resolveAfter2Seconds() {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve('resolved');
    }, 2000);
  });
}
```

```
Expected output:
// calling
// resolved
```

```
async function asyncCall() {
  console.log('calling');
  var result = await resolveAfter2Seconds();
  console.log(result);
}
```

Async Functions (3)

SoftUni Foundation

Do not confuse **await** with **Promise.then()**

- To **await two or more** promises in **parallel**, use **Promise.then()**

If a promise resolves normally, then **await** promise **returns the result**

- In case of a rejection, it **throws an error**

```
async function f() {
  try {
    let response = await fetch();
    let user = await response.json();
  } catch (err) {
    // catches errors both in fetch and
    response.json
    alert(err);
  }
}
```

```
async function f() {
  let response = await fetch();
}

// f() becomes a rejected promise
f().catch(alert);
```

Sequential Execution

To execute different promise methods **one by one**, use Async /Await

```
function doJob(x,sec) {
  return new Promise(resolve => {
  console.log('Start: ' + x);
    setTimeout(() => {
        console.log('End: ' + x);
      resolve(x);
    }, sec *1000);
  });
}
```

```
async function SerialFlow() {
 let result1 = await doJob(1,1);
 let result2 = await doJob(2,2);
 let result3 = await doJob(3,3);
 let finalResult = result1 + result2 + result3;
 console.log(finalResult);
}
```

```
// Start: 1
// End: 1
// Start: 2
// End: 2
// Start: 3
// End: 3
// 6
```

Concurrent Execution

```
async function ParallelFlow() {
 let result1 = doJob(1,1);
 let result2 = doJob(2,2);
 let result3 = doJob(3,3);
 let finalResult = await result1 + await result2 + await result3;
 console.log(finalResult);
}
```

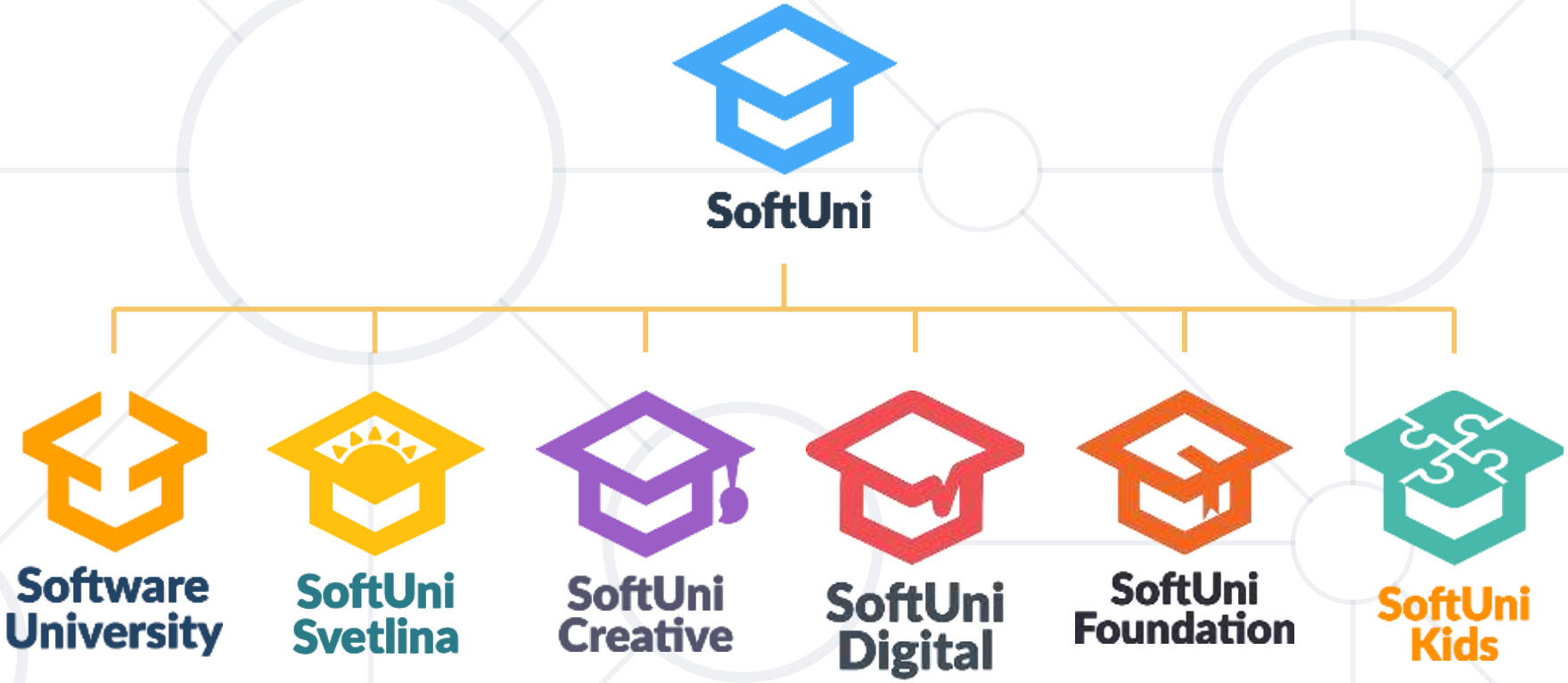
```
// Expected output:
Start: 1
Start: 2
Start: 3
End: 1
End: 2
End: 3
6
```

Live Exercises

Summary

- Promises hold **operations**
  - Can be **resolved** or **rejected**
- jQuery **AJAX** works with **promises**
- **Async** functions contain an **await** expression
  - It **pauses** the **execution**
  - **Waits** for the **Promise's resolution**

Questions?



https://softuni.bg/courses/js-apps

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

LIEBHERR

æternity

SoftUni Organizational Partners

SoftUni Foundation

ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

Lukanet.com

WORLD OF MYTHS

Trainings @ Software University (SoftUni)

- Software University - High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license