Creating Asynchronous Services



Brice Wilson

@brice_wilson www.BriceWilson.net



Overview



What are asynchronous services?

Why should you create them?

Observables

Promises

async/await



What Are Asynchronous Services?

Services with methods that execute asynchronously

Provided to injectors like any other service

Injected into components like any other service

Return different types

- Observables
- Promises

Callers must write additional code to process different return types





Synchronous Execution

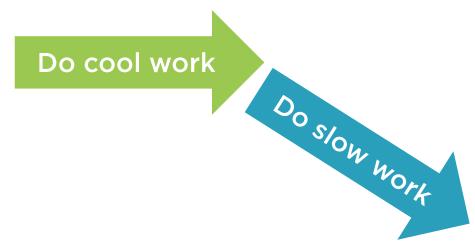


Synchronous Execution

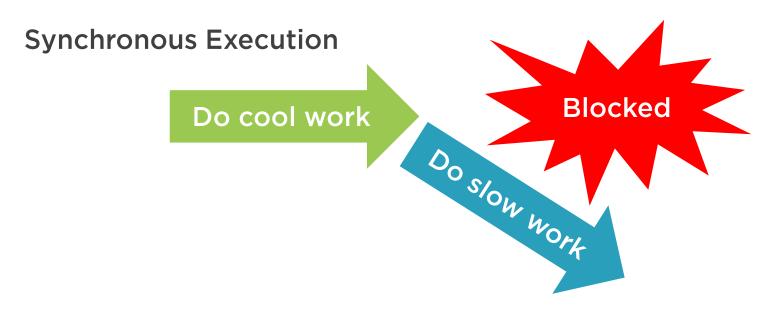
Do cool work



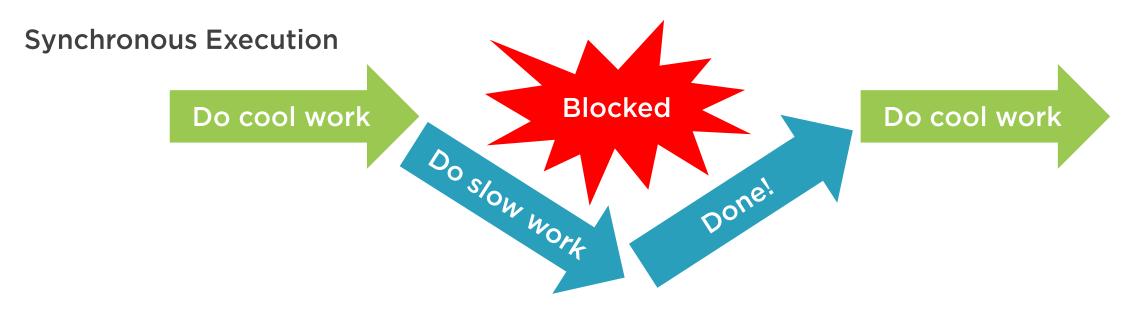
Synchronous Execution



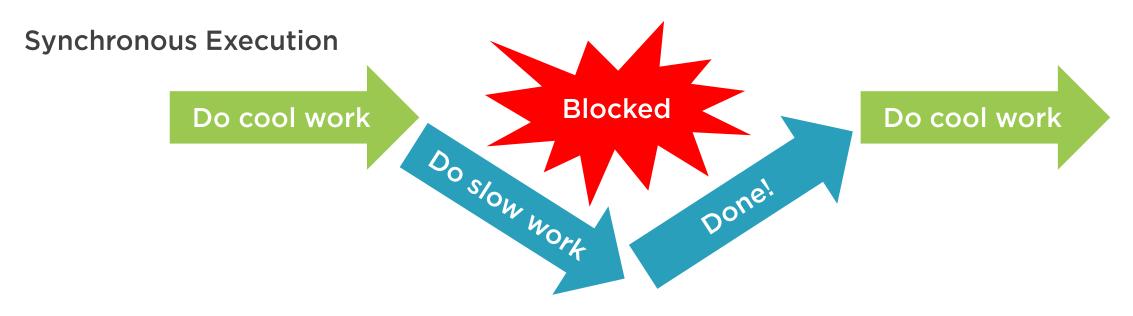






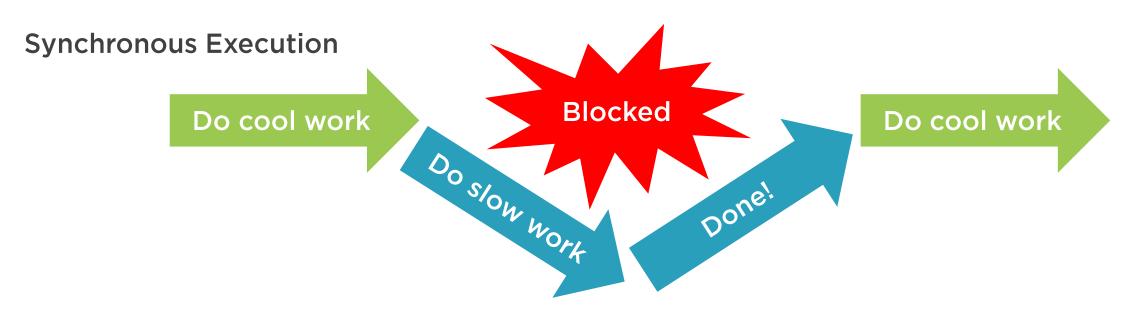






Asynchronous Execution

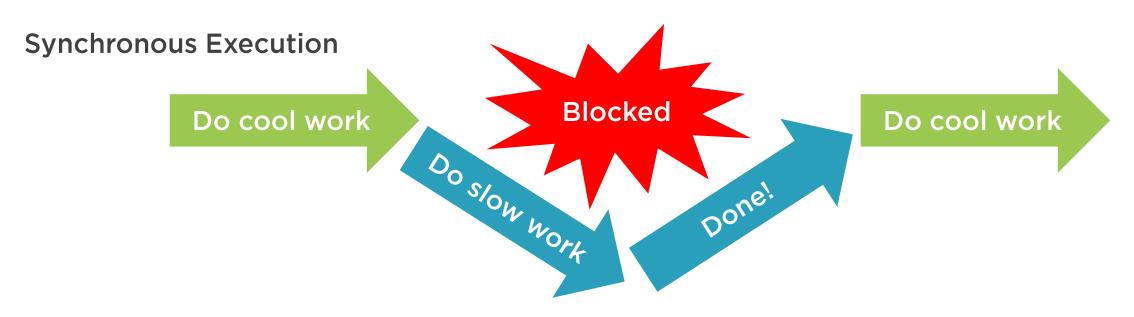




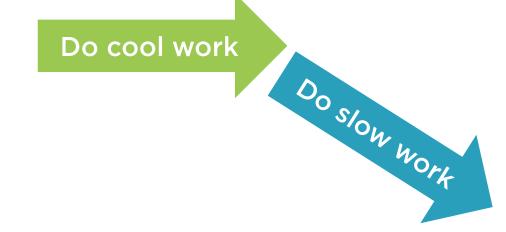
Asynchronous Execution

Do cool work

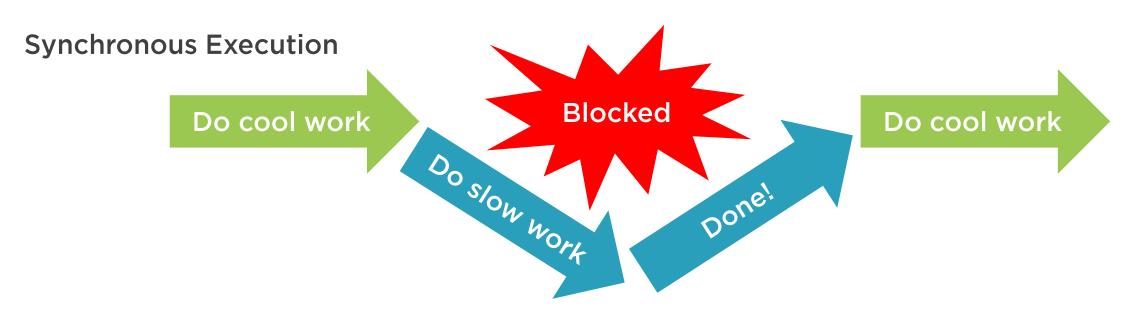




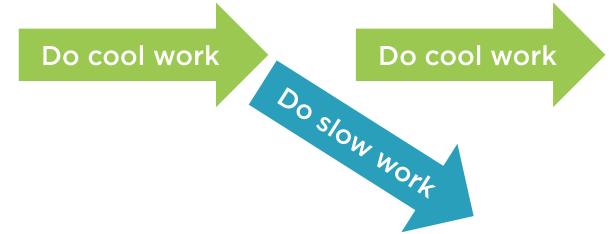
Asynchronous Execution



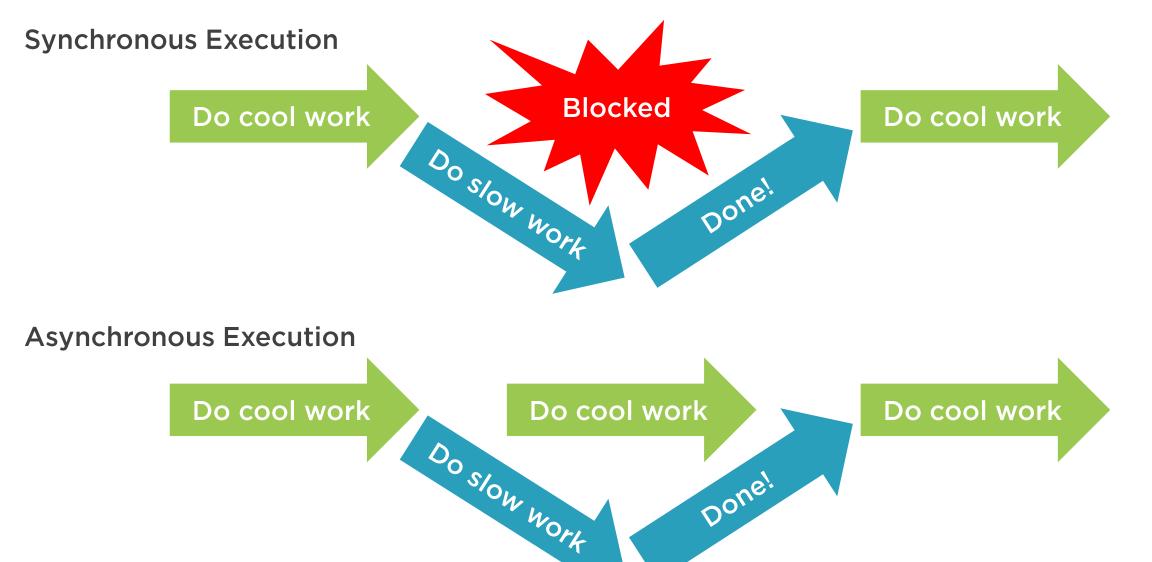




Asynchronous Execution









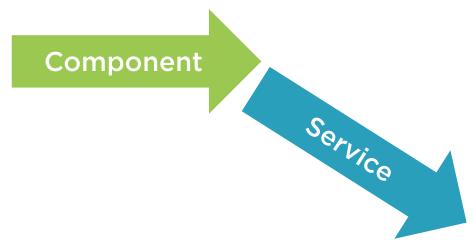


Synchronous Execution

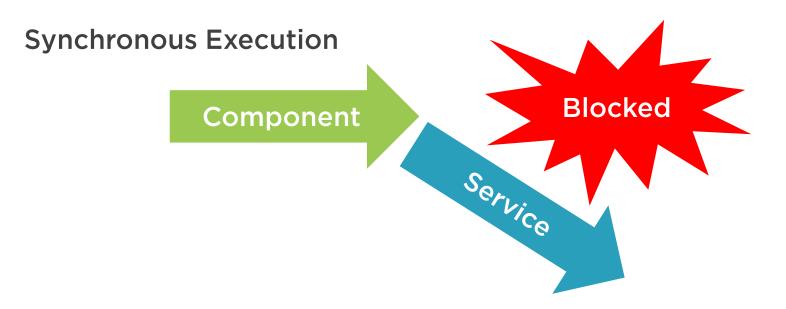
Component



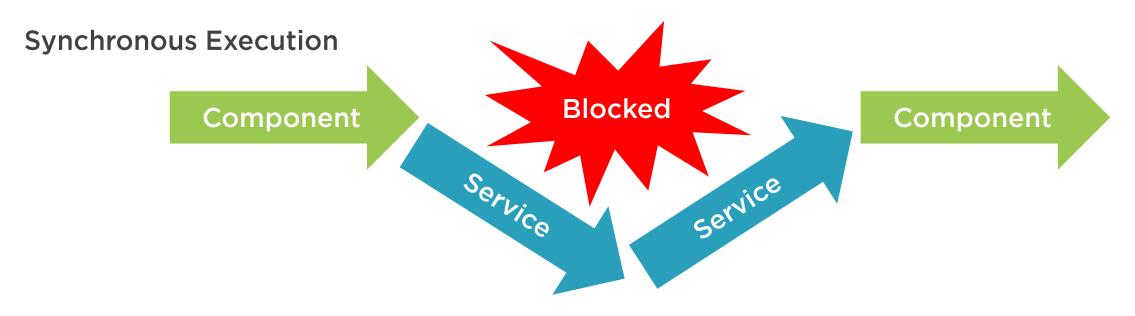
Synchronous Execution



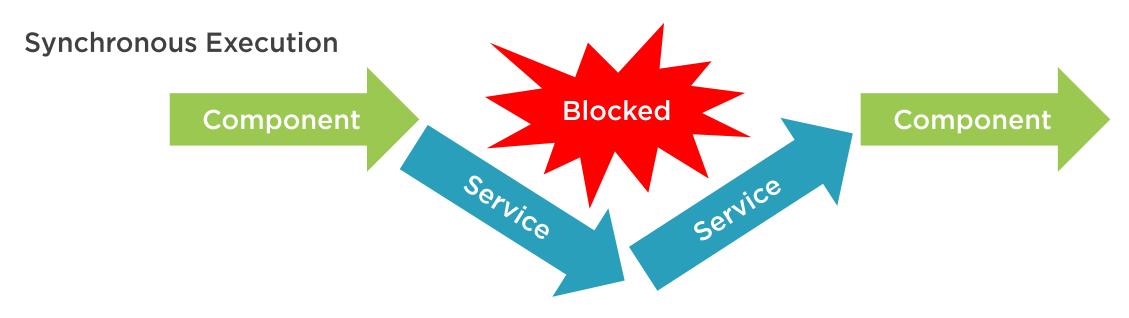








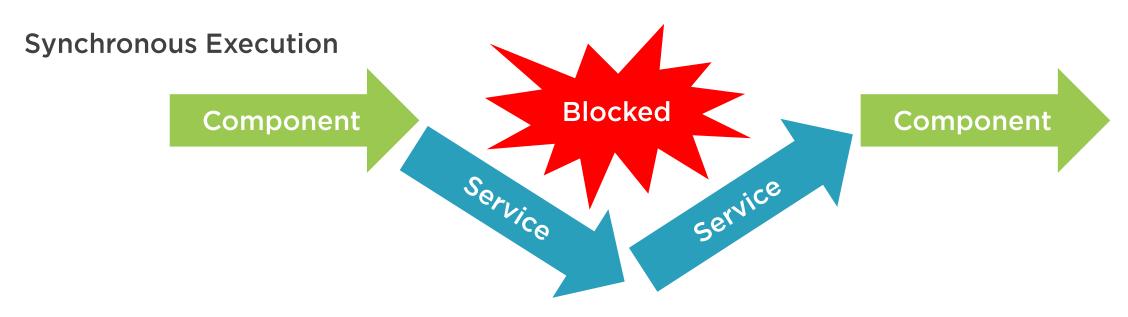




Asynchronous Execution

Component

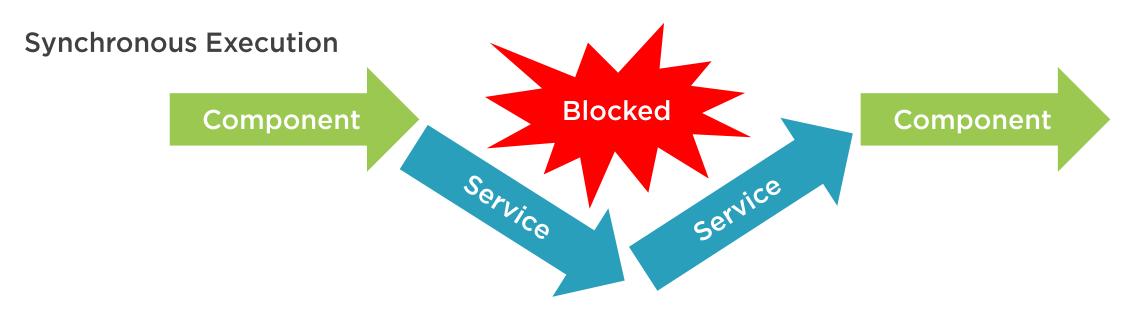


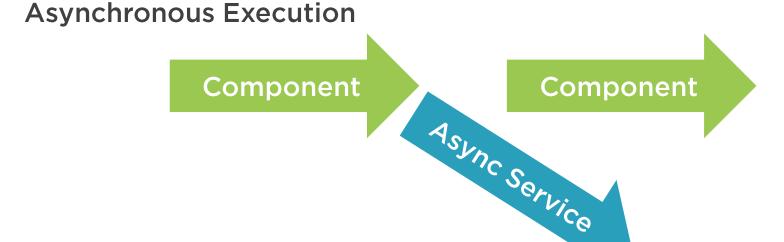


Asynchronous Execution

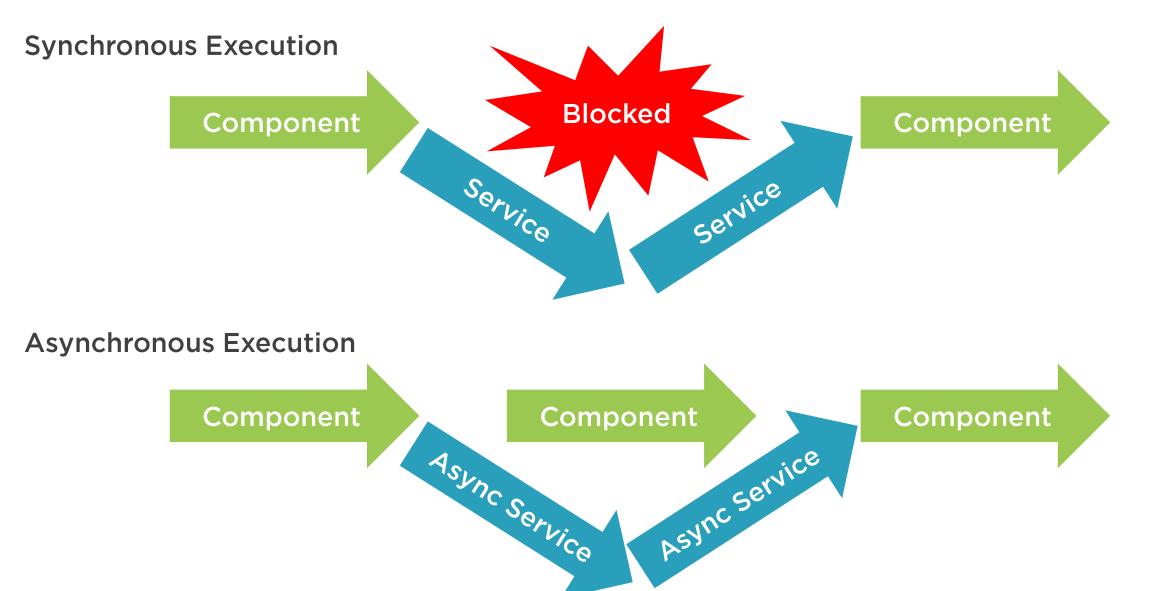
Component Async Service













Observables



Observables

Part of RxJS

Returned by methods on Angular's HttpClient

Can be used with much more than http requests

Very large API





```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Reader } from 'app/models/reader';
```



```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Reader } from 'app/models/reader';
getReaderById(id: number): Observable<Reader> {
  return this.http.get<Reader>(`/api/readers/${id}`);
```



```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Reader } from 'app/models/reader';
getReaderById(id: number): Observable<Reader> {
  return this.http.get<Reader>(`/api/readers/${id}`);
```



```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Reader } from 'app/models/reader';
getReaderById(id: number): Observable<Reader> {
  return this.http.get<Reader>(`/api/readers/${id}`);
```





```
// edit-reader.component.ts
```



```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
```



```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
```



```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
    .subscribe(
```





```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
    .subscribe(
```



```
// edit-reader.component.ts
ngOnInit() {
  let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
    .subscribe(
    data => this.selectedReader = data,
```



```
// edit-reader.component.ts
ngOnInit() {
 let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
    .subscribe(
      data => this.selectedReader = data,
      err => console.log(err),
```



```
// edit-reader.component.ts
ngOnInit() {
 let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
    .subscribe(
      data => this.selectedReader = data,
      err => console.log(err),
        => console.log('All done!')
```

```
// edit-reader.component.ts
ngOnInit() {
 let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataService.getReaderById(readerID)
    .subscribe(
      data => this.selectedReader = data,
      err => console.log( r),
      () => console.log(' loge!')
```

```
// edit-reader.component.ts
ngOnInit() {
 let readerID: number = parseInt(this.route.snapshot.params['id']);
  // this.selectedReader = this.dataService.getReaderById(readerID);
  this.dataServi ... getReaderById(readerID)
    .subscribe(
      data => this.selectedReader = data,
      err => console.log( r),
      () => console.log(' ll done!')
```

Demo



Processing an asynchronous HTTP request with an Observable



Demo



Abstracting away HTTP errors



Promises



Promises

Part of ES2015 specification

Polyfills shipped with Angular allow them to be used with ES5

General-purpose solution for performing asynchronous work

Results processed with callback functions

Will either be "resolved" or "rejected"





```
updateSchedule(empID: number): Promise<string> {
}
```



```
updateSchedule(empID: number): Promise<string> {
   return new Promise(this.doWork);
}
```



```
updateSchedule(empID: number): Promise<string> {
  return new Promise(this.doWork);
}
```



```
updateSchedule(empID: number): Promise<string> {
  return new Promise(this.doWork);
}
```



```
updateSchedule(empID: number): Promise<string> {
   return new Promise(this.doWork);
}
```





```
doWork(resolve, reject): void {
```



```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
```



```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
  if (result === 'success') {
```



```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
  if (result === 'success') {
```



```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
  if (result === 'success') {
    resolve('Done updating schedule.');
  }
```



```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
   if (result === 'success') {
     resolve('Done updating schedule.');
  else {
     reject('Unable to update schedule.');
```



```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
   if (result === 'success') {
     resolve('Done updating schedule.');
   else
     reject('Unable to update schedule.');
```



```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
   if (result === 'success') {
     resolve('Done updating schedule.');
   else {
     reject('Unable to update schedule.');
```

```
doWork(resolve, reject): void {
  let result: string = this.processCalendar();
   if (result === 'success') {
     resolve('Done updating schedule.');
  else {
     reject('Unable to update schedule.');
```



```
updateSchedule(empID: number): Promise<string> {
```



```
updateSchedule(empID: number): Promise<string> {
   return new Promise((resolve, reject) => {
```



```
updateSchedule(empID: number): Promise<string> {
    return new Promise((resolve, reject) => {
        let result: string = this.processCalendar();
```



```
updateSchedule(empID: number): Promise<string> {
    return new Promise((resolve, reject) => {
        let result: string = this.processCalendar();
        if (result === 'success') {
            resolve('Done updating schedule.');
```



```
updateSchedule(empID: number): Promise<string> {
   return new Promise((resolve, reject) => {
    let result: string = his.promiseCalendar();
   if (result === 'succe's') {
      resolve('Done updating schedule.');
}
```



```
updateSchedule(empID: number): Promise<string> {
   return new Promise((resolve, reject) => {
     let result: string = this.processCalendar();
     if (result === 'success') {
       resolve('Done updating schedule.');
     else {
       reject('Unable to update schedule.');
   });
```



```
ngOnInit() {
```



```
ngOnInit() {
    this.dataService.updateSchedule(10)
```



```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
    data => console.log(`Resolved: ${data}`),
    reason => console.log(`Rejected: ${reason}`)
  )
```



```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
      data => console.log(`Resolved: ${data}`),
      reason => console.log(`Rejected: ${reason}`)
    .catch(
      err => console.log(`ERROR: ${err}`)
```

```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
      data => console.log(`Resolved: ${data}`),
      reason => console.log(`Rejected: ${reason}`)
    .catch(
      err => console.log(`ERROR: ${err}`)
```

```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
      data => console.log(`Resolved: ${data}`),
      reason => console.log(`Rejected: ${reason}`)
    .catch(
      err => console.log(`ERROR: ${err}`)
```

```
ngOnInit() {
  this.dataService.updateSchedule(10)
    .then(
      data => console.log(`Resolved: ${data}`),
      reason => console.log(`Rejected: ${reason}`)
    .catch(
      err => console.log(`ERROR: ${err}`)
```

Demo



Asynchronously executing a task with a **Promise**



async/await



async/await

Work with Promises

Write code in a more linear style

Functions declared with the "async" keyword

Execution will "await" the resolution of a Promise



Demo



Handling a Promise with async/await



Summary



Network calls and long-running tasks should be in services

Asynchronous services are important

Not difficult to implement

Do it for your users!!!

