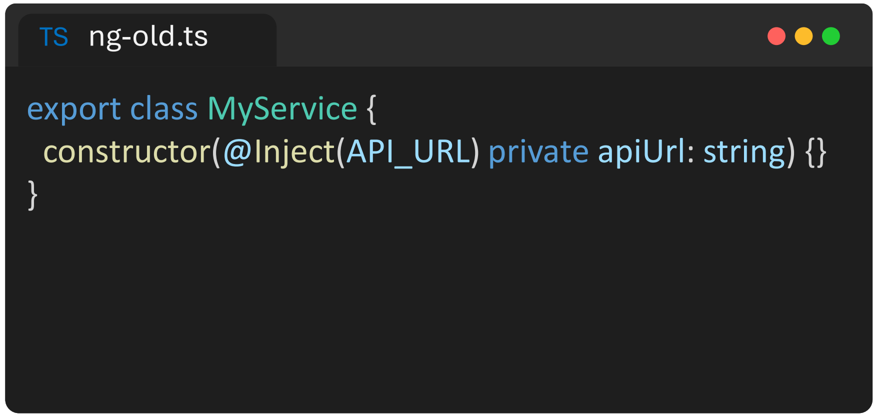
Angular

Renaissance

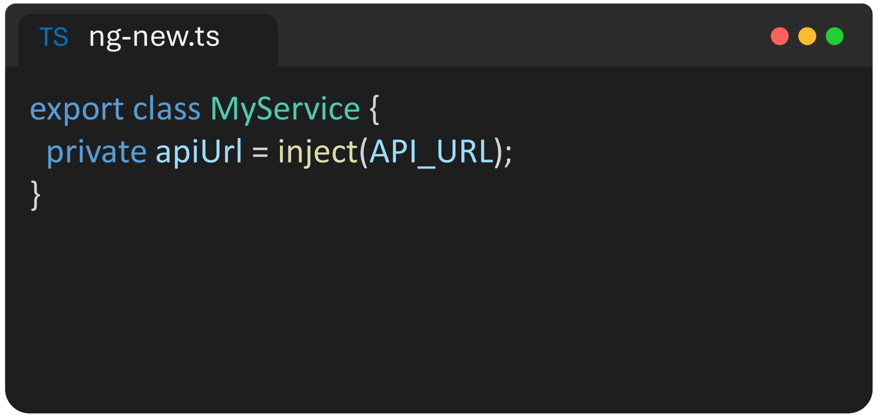
Angular Renaissance

`Inject` function replaces constructor injection and the `@Inject` decorator



ng-old.ts

```
export class MyService {
  constructor(@Inject(API_URL) private apiUrl: string) {}
}
```

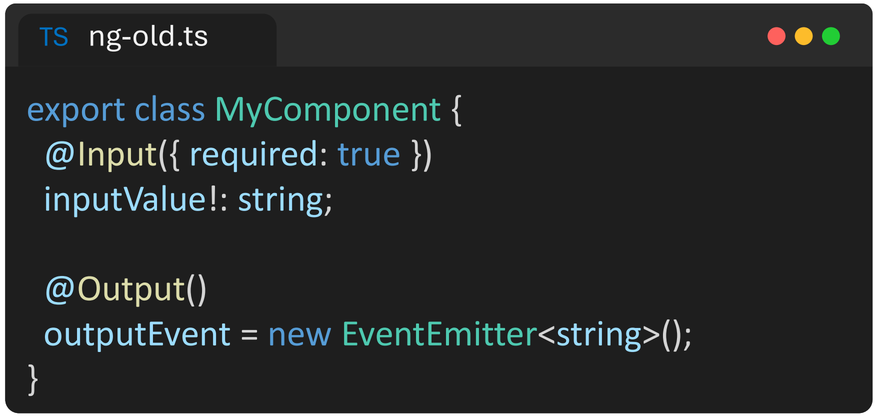


ng-new.ts

```
export class MyService {
  private apiUrl = inject(API_URL);
}
```

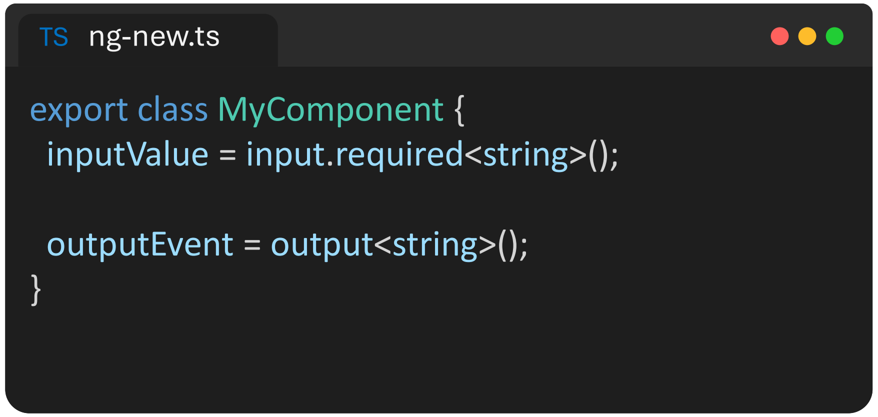
Angular Renaissance

`Input, output, model` functions replace `@Decorators`



```
export class MyComponent {
 @Input({ required: true })
 inputValue!: string;

 @Output()
 outputEvent = new EventEmitter<string>();
}
```

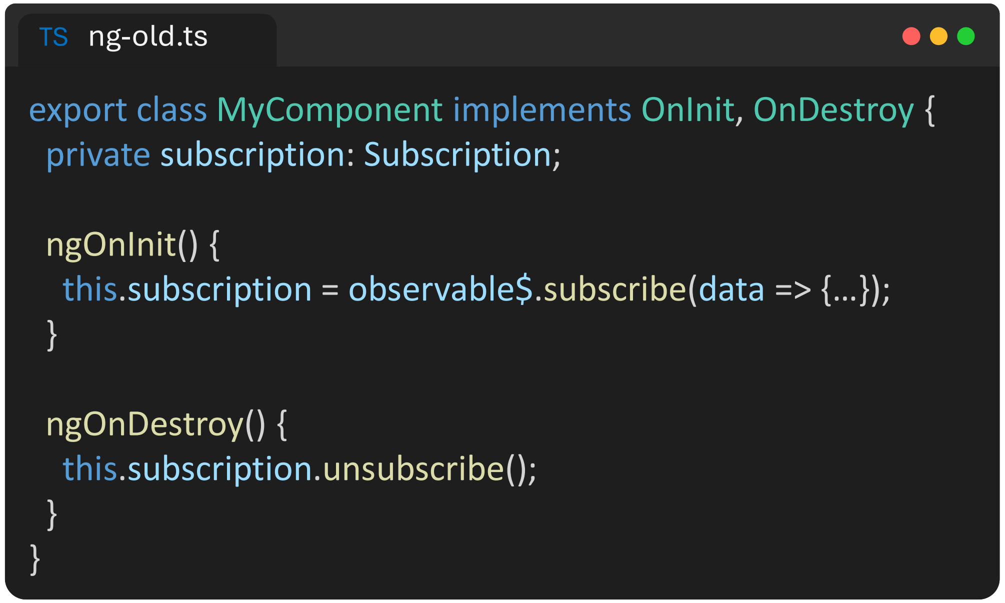


```
export class MyComponent {
 inputValue = input.required<string>();

 outputEvent = output<string>();
}
```

Angular Renaissance

`OnDestroy` interface
is replaced by
`DestroyRef.onDestroy`
higher order function

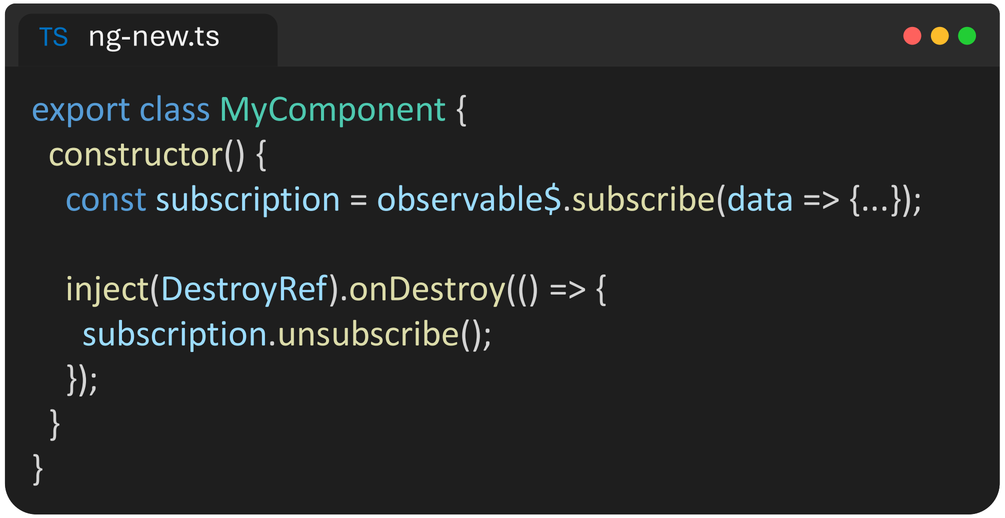


TS ng-old.ts

```
export class MyComponent implements OnInit, OnDestroy {
 private subscription: Subscription;

 ngOnInit() {
  this.subscription = observable$.subscribe(data => {...});
 }

 ngOnDestroy() {
  this.subscription.unsubscribe();
 }
}
```



TS ng-new.ts

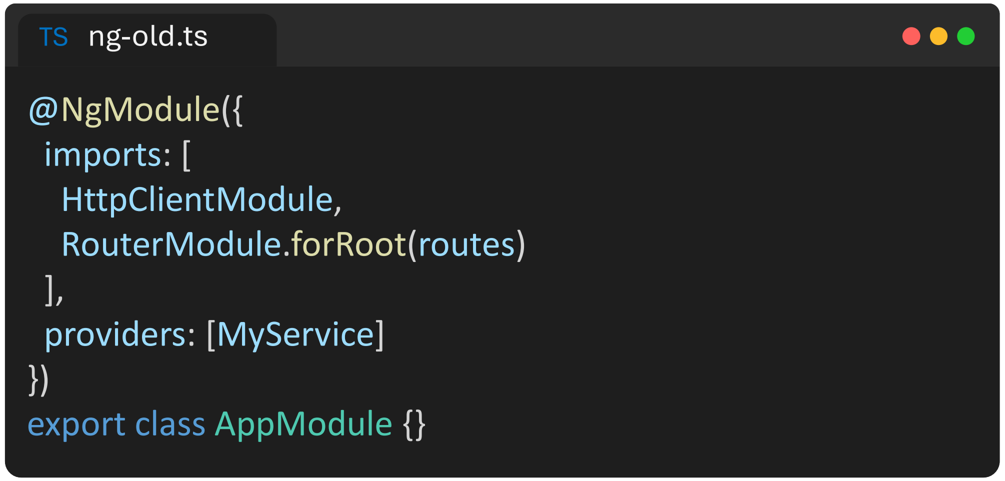
```
export class MyComponent {
 constructor() {
  const subscription = observable$.subscribe(data => {...});

  inject(DestroyRef).onDestroy(() => {
   subscription.unsubscribe();
  });
 }
}
```

WHOA!

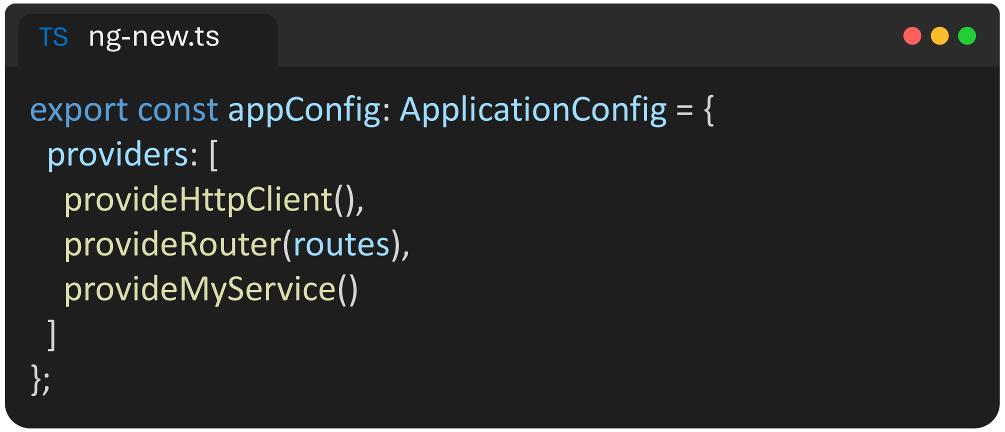
Angular Renaissance

`prodive` functions replace `@ngModule providers`



ng-old.ts

```ts
@NgModule({
  imports: [
    HttpClientModule,
    RouterModule.forRoot(routes)
  ],
  providers: [MyService]
})
export class AppModule {}
```

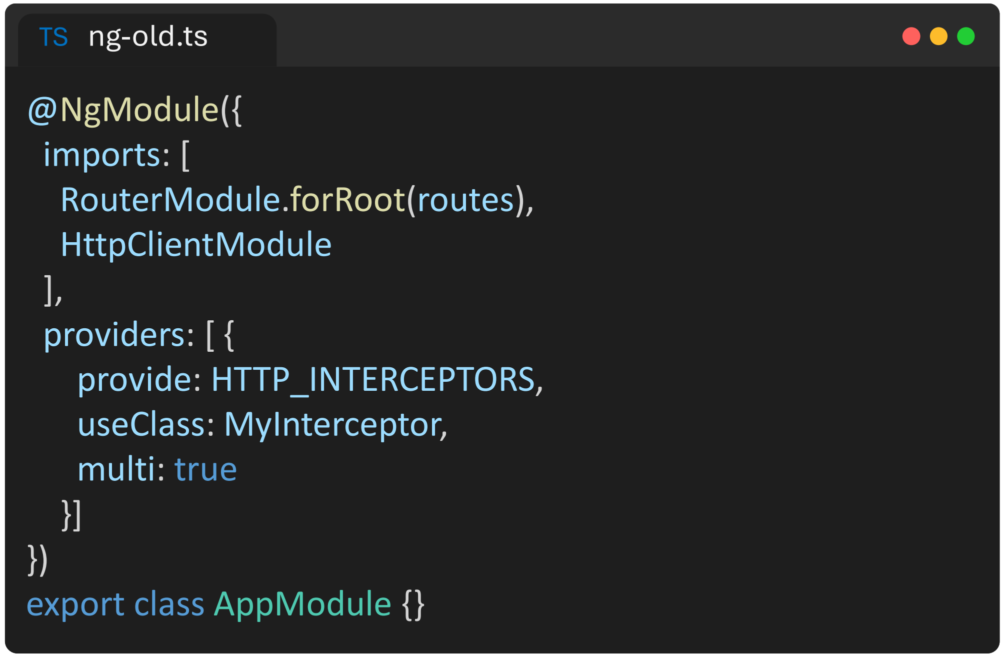


ng-new.ts

```ts
export const appConfig: ApplicationConfig = {
  providers: [
    provideHttpClient(),
    provideRouter(routes),
    provideMyService()
  ]
};
```

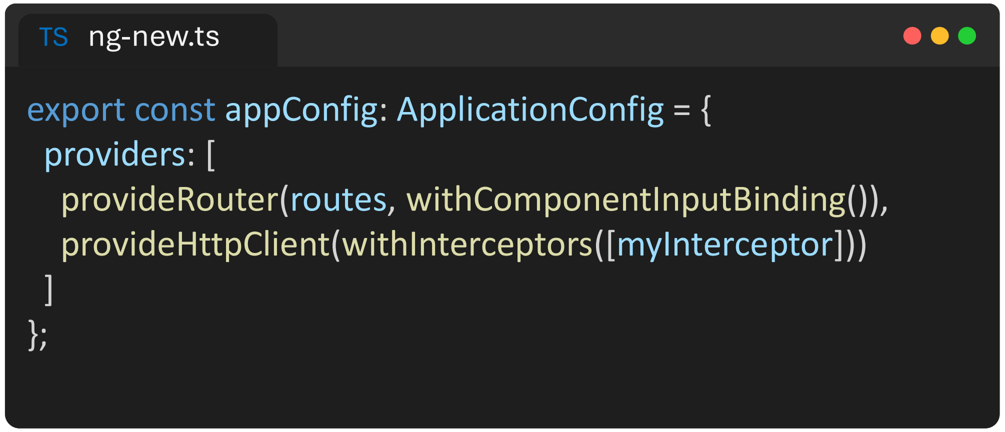
Angular Renaissance

`with[feature]` function parameters replace `modules and providers`



ng-old.ts

```ts
@NgModule({
 imports: [
  RouterModule.forRoot(routes),
  HttpClientModule
 ],
 providers: [ {
    provide: HTTP_INTERCEPTORS,
    useClass: MyInterceptor,
    multi: true
  }]
})
export class AppModule {}
```



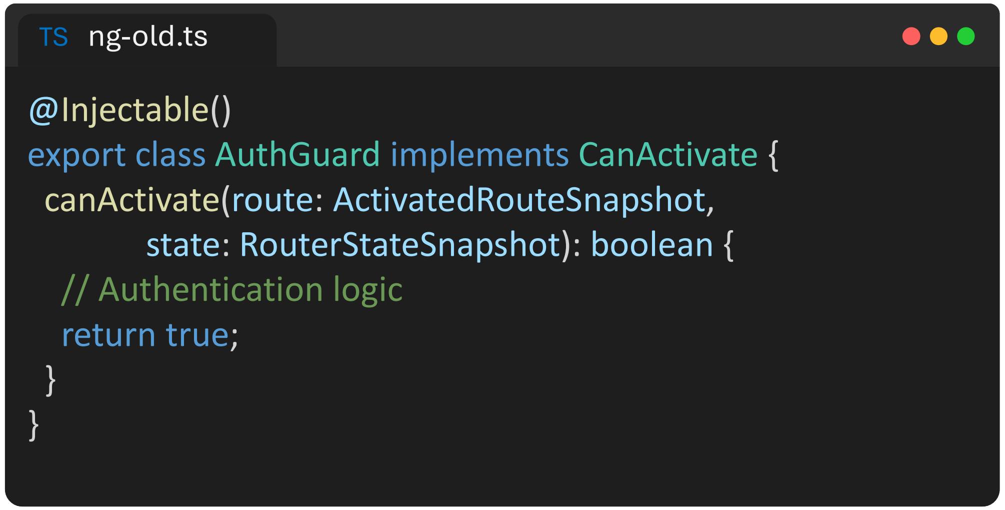
ng-new.ts

```ts
export const appConfig: ApplicationConfig = {
 providers: [
  provideRouter(routes, withComponentInputBinding()),
  provideHttpClient(withInterceptors([myInterceptor]))
 ]
};
```

Angular Renaissance

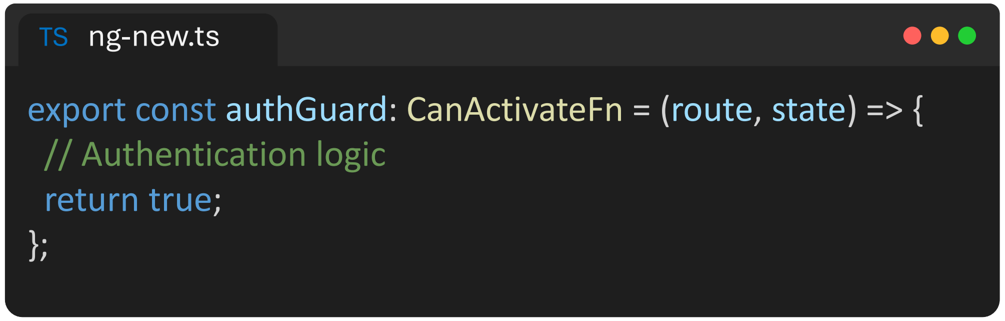
`guard` is a function constant instead of `@Injectable class` with `interface` implementation,

Same for `resolver, interceptor`



TS ng-old.ts

```
@Injectable()
export class AuthGuard implements CanActivate {
 canActivate(route: ActivatedRouteSnapshot,
          state: RouterStateSnapshot): boolean {
  // Authentication logic
  return true;
 }
}
```



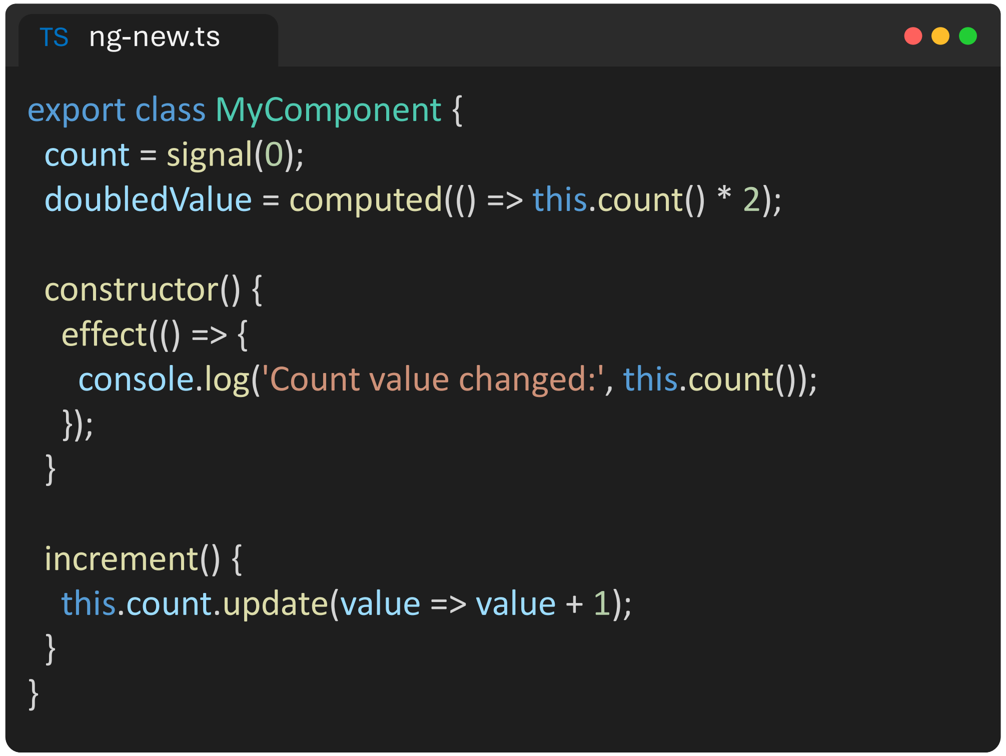
TS ng-new.ts

```
export const authGuard: CanActivateFn = (route, state) => {
 // Authentication logic
 return true;
};
```

Angular Renaissance

The functions:
`signal, signal.set`

and the higher order functions :
`computed, effect, signal.update`.

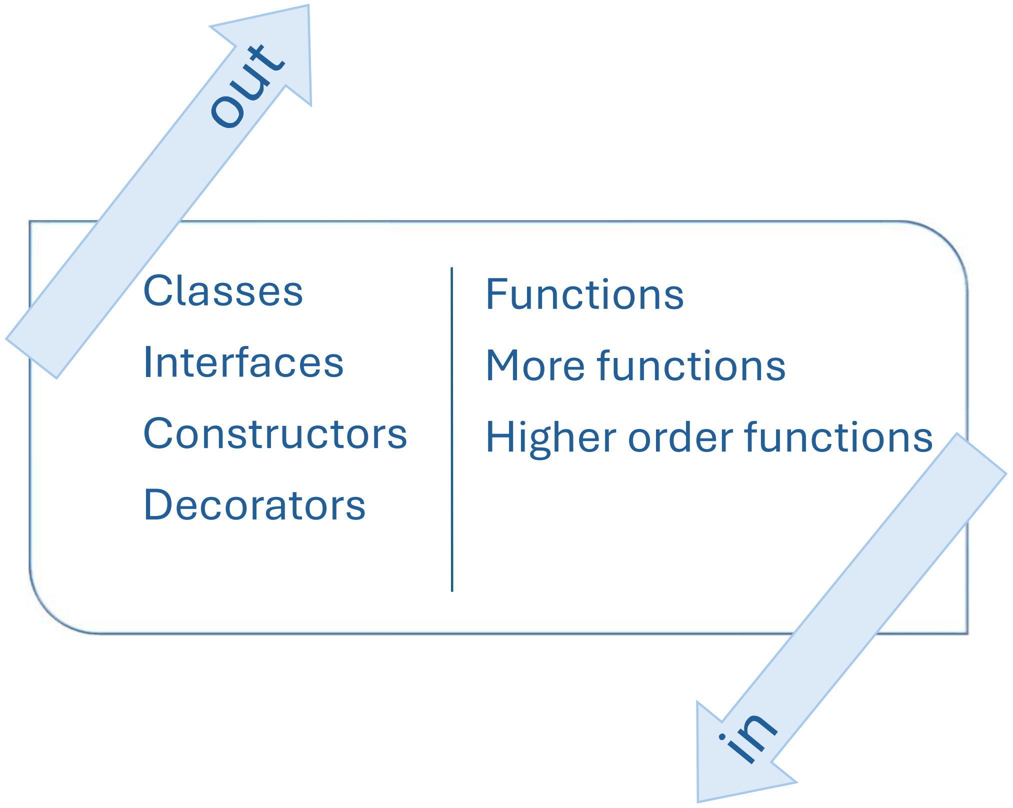


```
export class MyComponent {
  count = signal(0);
  doubledValue = computed(() => this.count() * 2);

  constructor() {
    effect(() => {
      console.log('Count value changed:', this.count());
    });
  }

  increment() {
    this.count.update(value => value + 1);
  }
}
```

Can you see a pattern?


out
Classes
Interfaces
Constructors
Decorators
Functions
More functions
Higher order functions
in