Custom Services with .provider()

STEP 1: Define Provider Function

```
function ServiceProvider() {
  var provider = this;
  provider.config = {…};

  provider.$get = function () {
    var service =
        new Service(provider.config.prop);

    return service;
  };
}
```

Provide some config object. Usually with defaults

Factory function, just like the one provided to .factory()

STEP 2: Register Provider Function With Module

```
angular.module('app', [])
.controller('ctrl', Ctrl)
.provider('Service', ServiceProvider);
```

Name of service as it will be injected into other services, controllers, etc.

Name of this function does NOT matter

STEP 3: Inject it As Usual

```
Ctrl.$inject =
   ['$scope', 'Service'];

function Ctrl($scope,
              Service) {
  Service.someMethod();
};
```

Inject with .provider defined name

STEP 4a (Optional): Register Config Function

```
angular.module('app', [])
.controller('ctrl', Ctrl)
.provider('Service', ServiceProvider)
.config(Config);
```

Guaranteed to run before any services, factories, or controllers are created

STEP 4b (Optional): Inject Provider into Config

Function name irrelevant

Registered name +Provider

```
…
.provider('Service', ServiceProvider);
```

```
Config.$inject = ['ServiceProvider'];

function Config(ServiceProvider) {
   ServiceProvider.config.prop = 'value';
};
```

STEP 4b (Optional): Inject Provider into Config

```
…
.provider('Service', ServiceProvider);
```

```
Config.$inject = ['ServiceProvider'];

function Ctrl(ServiceProvider) {
   ServiceProvider.config.prop = 'value';
};
```

Summary

- .provider() - most verbose, but most flexible
  - Configure factory not just at time of use, but at app bootstrapping
- .provider('name', function)
  - Whatever the 'name' is – that's what gets injected into other components
- .config() function gets called *before* any service, factory, or controller is instantiated
  - Therefore, we can't inject any regular components into .config
  - We CAN inject the provider of service with name*Provider*