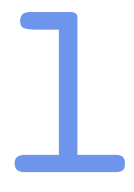
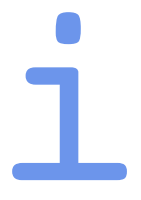
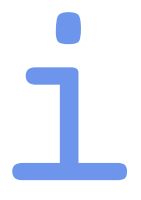
```
<Foo>
    <BarSubstitute />
</Foo>
```







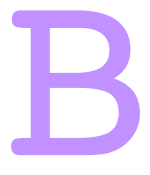
1

B

r

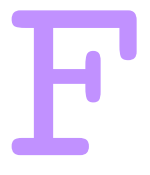
B


B


B

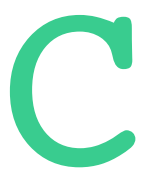

R


R

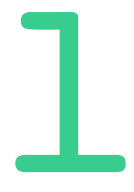


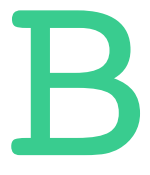

A

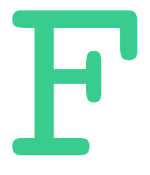

a

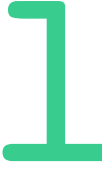


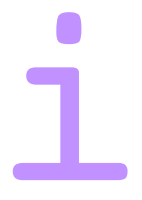

B

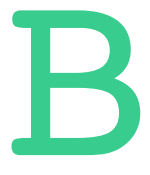


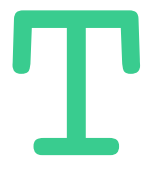












@code {

[Fact]

```
public void ShouldHaveBarSubstituteInsteadOfBar()
```

```
// Tell bUnit to replace <Bar /> with BarSubstitute
```

```
ComponentFactories.Add<Bar, BarSubstitute>();
```

```
// Render our component
```

```
IRenderedFragment cut = Render(@<Foo></Foo>);
```

```
// Assert that <Bar /> is not part of the render tree
```

```
cut.HasComponent<Bar>().Should().BeFalse();
```

```
// Assert that <BarSubstitute /> is part of the render tree
```

```
cut.HasComponent<BarSubstitute>().Should().BeTrue();
```

Stub a component

Create a replacement

```
@code {
    [Fact]
    public void ShouldHaveBarSubstituteInsteadOfBar()
    {
        // Tell bUnit to replace <Bar /> with BarSubstitute
        ComponentFactories.Add<Bar, BarSubstitute>();

        // Render our component
        IRenderedFragment cut = Render(@<Foo></Foo>);

        // Assert that <Bar /> is not part of the render tree
        cut.HasComponent<Bar>().Should().BeFalse();
        // Assert that <BarSubstitute /> is part of the render tree
        cut.HasComponent<BarSubstitute>().Should().BeTrue();
    }
}
```

```
<Foo>
    <BarSubstitute />
</Foo>
```

- With ComponentFactories we can tell bUnit to replace <Bar> with a component of our choice
- Making setup easier or certain behaviour easier to assert

TODO

If there is time: How to assert JavaScript calls