Most commonly used assertions in Cypress

Author: Anshita Bhasin
LinkedIn: https://www.linkedin.com/in/anshita-bhasin/

In Cypress, assertions are used to verify that the state of the application being tested meets the expected conditions.

An assertion typically consists of a target value (the actual value being tested) and an expected value (the expected result of the test). If the target value matches the expected value, the assertion passes. If the values do not match, the assertion fails and the test is considered to have failed.

Below are some of the most commonly used assertions in Cypress with examples:

1. Verify that an element exists on the page:
   Syntax=> .should('exist')

```
Example:
cy.get('#my-element').should('exist')
```

2. Verify that an element has a specific class:
   Syntax=> .should('have.class',classname)

```
Example:
cy.get('#my-element').should('have.class', 'active')
```

3. Verify that an element has a specific attribute:
Syntax=> .should('have.attr',attributeName)

```
Example:
cy.get('#my-element').should('have.attr', 'data-id', '123')
```

4. Verify that an element has a specific value:
Syntax=> .should('have.value',expectedValue)

```
Example:
cy.get('#my-input').should('have.value', 'My Input Value')
```

5. Verify that an element is visible:
Syntax=> .should('be.visible')

```
Example:
cy.get('#my-element').should('be.visible')
```

6. Verify that an element is not visible:
Syntax=> .should('not.be.visible")

```
Example:
cy.get('#my-element').should('not.be.visible')
```

7. Verify that an element is disabled:
Syntax=> .should('be.disabled')

```
Example:
cy.get('#my-button').should('be.disabled')
```

8. Verify if an element is enabled:
Syntax=> .should('be.enabled')

```
Example:
1)cy.get('.my-element').should('be.enabled')
2)cy.get('#my-button').should('not.be.disabled')
```

9. Verify if the element is focussed
Syntax=> .should('be.focused')

```
Example:

1)cy.get('#my-button').should('be.focused')
2)cy.get('#input-receives-focus').should('have.focus')
```

10. Verify if the length is as expected
Syntax=> .should('have.length',expectedLength)

```
Example:
cy.get('.div>li').should('have.length',10);
```

11. Verify if a specific element has a particular CSS property
Syntax=> .should('have.css',cssValue)

```
Example:
cy.get('.my-element').should('have.css', 'color', 'red')
```

12. Verify multiple assertions at a time
Syntax=> .and()

```
Example:

cy.get('.my-element')
.should('have.class', 'active')
.and('have.attr', 'href')
.and('include', 'cypress.io')
```

13. Verify Object assertions

Syntax => .should('have.property')

Example:

```
const expectedObject = {
name: 'John Doe',
age: 28,
city: 'New York'
}

cy.get('.my-element')
.should('have.property', 'name', expectedObject.name)
.should('have.property', 'age', expectedObject.age)
.should('have.property', 'city', expectedObject.city)
```

In the above example, we are using the cy.should() command to check that the element with the .my-element CSS class has the properties name, age, and city and that each of these properties has the expected value. If any of these assertions fail, Cypress will throw an error and stop the test.

14. Verify if an element has a specific property and that the value of that property is as expected

Syntax => .should('have.prop')

```
Example:

cy.get('.my-element')
.should('have.prop', 'name', 'John Doe')
```

In this example, we are using the cy.get() command to target the element with the .my-element CSS class, and then using the should() and *have.prop* commands to assert that the element has a name property with the value John Doe. If the assertion fails, Cypress will throw an error and stop the test.

15. Verify that a given value is NaN, or "not a number".
    Syntax => .should('be.a.NaN')
    Syntax => *.should('not.be.NaN')*

```
Examples:

1) cy.get('.my-element')
.should('have.prop', 'name')
.should('be.NaN')

2)cy.wrap(NaN).should('be.a.NaN')

3)cy.wrap(42).should('not.be.a.NaN')
```

16. Verify if an element or collection of elements is empty (i.e. that it contains no child elements).
Syntax => .should('be.empty')

```
Example:

cy.get('.my-list li')
.should('be.empty')
```

17. Verify if an element or collection of elements contains a specific piece of text.
verify if the element is present
Syntax => .should(contain,text)

```
Example:

cy.get('#my-element')
.should('contain', 'Hello World')
```

verify if a specific item is present in the list like this:
Syntax => .should(contain,value)

```
Example:

cy.get('.my-list li')
.should('contain', 'Item 1')
```

18. Verify if a checkbox or radio button is checked
Syntax => .should('be.checked')

```
Examples:

1) cy.get('#my-checkbox')
.should('be.checked')

2) cy.get('input[name="my-radio-buttons"]')
.should('be.checked')
```

19. Verify if a checkbox or radio button is not checked
Syntax => .should('not.be.checked')

```
Examples:

1) cy.get('#my-checkbox')
.should('not.be.checked')

2) cy.get('input[name="my-radio-buttons"]')
.should('not.be.checked')
```

20. Verify if an element has a specific data attribute with a specific value.
Syntax => have.data

```
Example:

cy.get('#my-element')
.should('have.data', 'id', '123')
```

21. Verify if it is an array
Syntax => .should('be.an','array')

```
Examples:

1) cy.get('#my-element')
.should('be.an', 'array')

2) cy.wrap([11,20,38,88,56]).should('be.an', 'array').and('have.length', 5)
```

22. Verify if an object has specific keys.
Syntax => .have.keys

```
Examples:
1) cy.get('#my-element')
.should('have.keys', ['id', 'name', 'email'])

2) cy.wrap(address).should('have.keys', ['city', 'state', 'zip'])
```

23. Verify if a value is one of a specific set of values.
Syntax => be.oneOf

```
Examples:

1) cy.get('#my-element')
.should('be.oneOf', ['red', 'green', 'blue'])

2) cy.wrap(88).should('be.oneOf', [5, 42, 88])
```

24. Verify if an object has a specific method
Syntax => should('respondTo')

```
Examples:

1) cy.get('#my-element')
.should('respondTo', 'myMethod')

2) cy.wrap(person).should('respondTo', 'greeting')
```

We can also use the *respondTo* assertion on a variable that we have defined in our test. For example, if we have an object called myObject, we could use the *respondTo* assertion like this:

```
Example:
expect(myObject).to.respondTo('myMethod')
```

In this case, the *respondTo* assertion will check that the myObject object has a method called myMethod. If it does not, the test will fail.
The *respondTo* assertion is useful for ensuring that an object has the expected methods. This can help you prevent bugs and ensure that your application is functioning as intended.

25. Verify that a numeric value is within a certain range of another value.
    Syntax => 'be.closeTo'

```
Example:

cy.get('#my-element')
.should('be.closeTo', 50, 3)
```

In this example, the *be.closeTo* assertion checks that the numeric value of the #my-element element is within 3 units of the value 50. If the value is not within this range, the test will fail.

We can also use the *be.closeTo* assertion on a variable that we have defined in our test. For example, if we have a variable called myValue, we can use the *be.closeTo* assertion like below:

```
expect(myValue).to.be.closeTo(50, 3)
```

In this case, the *be.closeTo* assertion will check that the myValue variable is within 3 units of the value 50. If it is not, the test will fail.
The *be.closeTo* assertion is helpful in ensuring that numeric values are within a certain range.

26. Verify if an element has specific text.

Syntax => have.text

```
Examples:

1) cy.get('#my-element')
.should('have.text', 'Hello world!')

2) cy.get('.my-list li')
.should('have.text', 'Hello world!')
```

Conclusion:
The above-shared list is the most commonly used assertion in Cypress. Cypress includes a number of built-in assertions that you can use to verify the state of your application.
There is also a list of BDD, TDD, and chai Assertions which is supported in Cypress.

Thanks for reading. Happy Learning! AB
Ref: https://docs.cypress.io/guides/references/assertions

~Anshita Bhasin