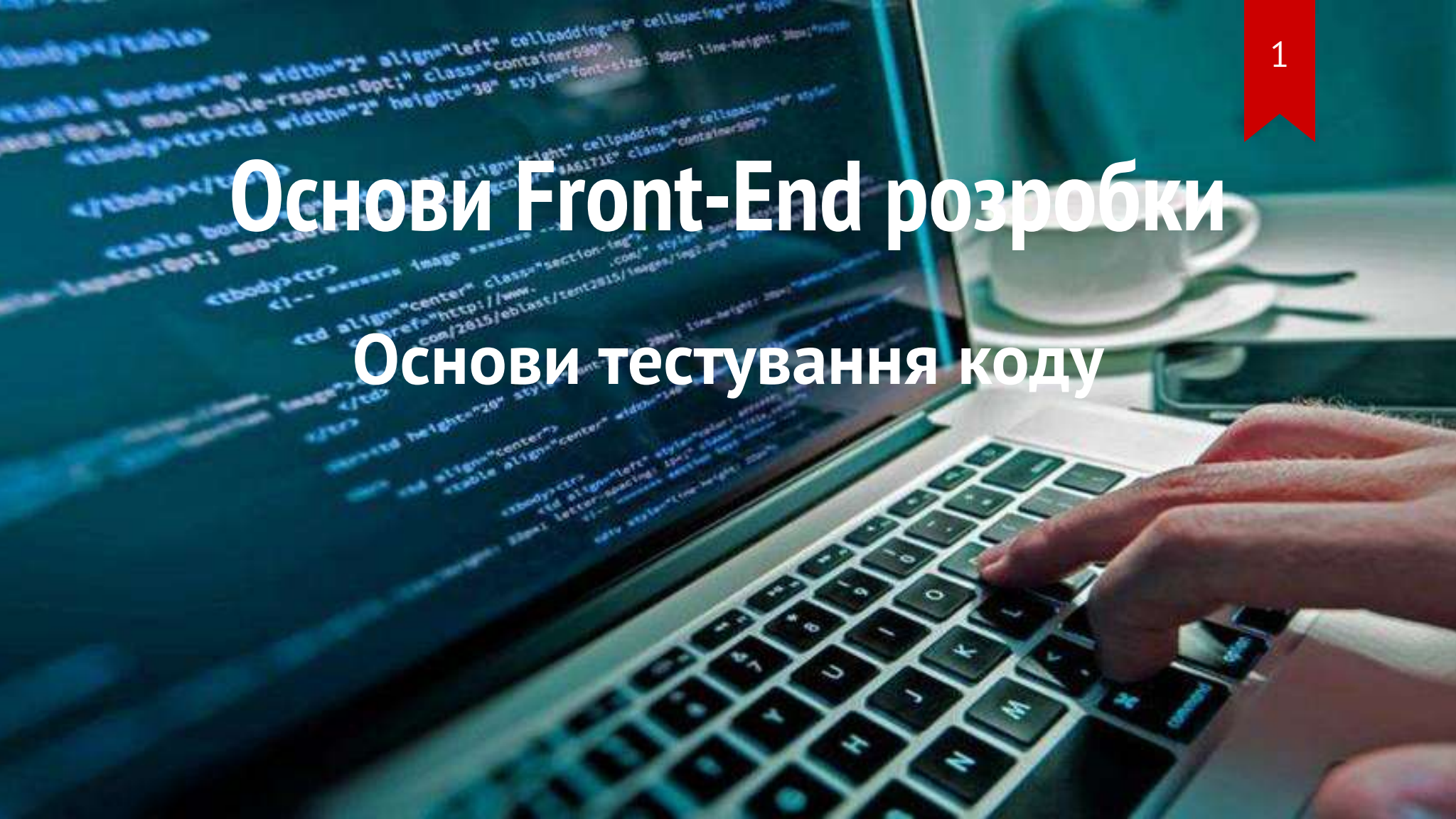
Основи Front-End розробки

Основи тестування коду



Що таке тестування?

Тестування - це процес перевірки роботи програмного забезпечення на наявність помилок.

Тестування може бути ручним або автоматизованим.

Ручне тестування виконується людиною, яка перевіряє роботу програмного забезпечення на наявність помилок.

Автоматизоване тестування - це процес перевірки роботи програмного забезпечення на наявність помилок, який виконується за допомогою спеціального програмного забезпечення.

Підходи до тестування

Smoke testing - у тестуванні програмного забезпечення означає мінімальний набір тестів на явні помилки.

Чорна скринька - тестування, при якому виконується перевірка роботи програмного забезпечення на наявність помилок, не знаючи його внутрушню будову.

Біла скринька - тестування, при якому виконується перевірка роботи програмного забезпечення на наявність помилок, спираючись на внутрішню будову програми.

Підходи до тестування (чорна скринька)

1. **Unit-тестування** полягає в тому, щоб тестувати окремі частини програмного коду, незалежно від інших частин. Наприклад, якщо у нас є функція, яка отримує два числа і повертає їх суму, то ми можемо написати тест, щоб перевірити, чи працює ця функція правильно.
2. **Інтеграційне** тестування використовується для тестування взаємодії різних частин системи. Наприклад, якщо у нас є веб-додаток, то ми можемо написати тест, щоб перевірити, чи правильно відображаються дані на сторінці після того, як користувач виконав певну дію.
3. **Функціональне** тестування використовується для тестування всієї системи як єдиного цілого. Наприклад, якщо у нас є онлайн-магазин, то ми можемо написати тест, щоб перевірити, чи правильно працює процес замовлення товару.
4. **Тестування залежностей** використовується для тестування коду, який використовується в нашій системі, але не написаний нами. Наприклад, якщо у нас є бібліотека, яка використовується в нашому проекті, то ми можемо написати тест, щоб перевірити, чи правильно працює ця бібліотека.

console.assert

`console.assert` - це метод, який приймає два аргументи: перший - умова, яка повинна бути виконана, а другий - повідомлення, яке буде виведено в консоль, якщо умова не буде виконана.

`console.assert` використовується для того, щоб перевірити, чи правильно працює програма.

```
console.assert(1 === 1, '1 не дорівнює 1');
```

```
console.assert(1 === 2, '1 не дорівнює 2');
```

Бібліотеки для тестування

- **Jest** - це один з найбільш популярних інструментів для тестування JavaScript коду. Він має багато корисних функцій, таких як автоматичне визначення тестів, генерація звітів, мокування (mocking) функцій та об'єктів, що дозволяє ефективно тестувати складні функції.
- **Mocha** - це ще один інструмент для тестування JavaScript коду, який має декілька варіантів використання. Mocha забезпечує широкі можливості для організації тестів, такі як передача параметрів у тести, включення / виключення тестів та запуск тестів в певному порядку.
- **Jasmine** - це ще один інструмент для тестування JavaScript коду, який дозволяє писати читабельний код тестів та легко відслідковувати проблеми під час їх виконання.

Бібліотеки для тестування

```
01. function add(a, b) {
02.     return a+b;
03. }
04.
05. console.assert(add(1, 2) === 3, '1 + 2 не дорівнює 3'); // перевірка за допомогою console
06.
07. describe("Test for sum", function() { // перевірка за допомогою бібліотеки
08.     it("should return 3", function() {
09.         expect(add(1, 2)).toBe(3);
10.     });
11. })
```

Бібліотека Jasmine (основні функції)

- **describe** - ця функція використовується для групування тестів, які перевіряють один або декілька пов'язаних функціональних блоків. Вона приймає два параметри: рядок з описом групи тестів та функцію, яка містить тести для цієї групи.
- **it** - ця функція використовується для опису окремих тестів. Вона також приймає два параметри: рядок з описом тесту та функцію, яка містить код для тестування.
- **expect** - ця функція використовується для перевірки очікуваного результату тесту. Вона приймає один параметр - значення, яке необхідно перевірити.
- **toBe** - ця функція використовується для перевірки, чи дорівнює отриманий результат очікуваному результату. Вона приймає один параметр - очікуване значення.
- **beforeEach** - ця функція використовується для виконання певного коду перед кожним тестом у групі. Вона приймає функцію, яка містить код, що необхідно виконати перед кожним тестом.
- **afterEach** - ця функція використовується для виконання певного коду після кожного тесту у групі. Вона приймає функцію, яка містить код, що необхідно виконати після кожного тесту.
- **spyOn** - ця функція використовується для створення шпигуна (spy) на функції або методі об'єкта. Вона приймає два параметри: об'єкт, на якому необхідно створити шпигуна, та назву функції або методу, я

Mock об’єкт

У тестуванні, **мок (mock)** - це об'єкт, який імітує реальний об'єкт і використовується для тестування коду, який залежить від цього об'єкту. Моки дозволяють замінити реальний об'єкт на контрольований об'єкт, що дозволяє ему виконувати очікувані дії та повертати очікувані значення.

Mock об’єкт

Функція `spyOn` в Jasmine дозволяє створювати моки на функції або методи об'єкта, що дозволяє перевіряти, чи викликали вони з певними параметрами та повернули очікувані значення.

Mock об’єкт

Наприклад, нехай ми маємо функцію `calculateSum`, яка використовує функцію `getNumbers` для отримання масиву чисел та обчислення їх суми

```
01. function getNumbers() {
02.     return [1, 2, 3];
03. }
04.
05. function calculateSum() {
06.     const numbers = getNumbers();
07.     return numbers.reduce((a, b) => a + b, 0);
08. }
```

Mock об’єкт

Ми можемо протестувати цю функцію з використанням моку на `getNumbers`, щоб перевірити, чи викликається вона з правильними параметрами:

```
01. describe('calculateSum', () => {
02.     it('should return sum of numbers', () => {
03.         spyOn(window, 'getNumbers').and.returnValue([1, 2, 3]);
04.         expect(calculateSum()).toBe(6);
05.         expect(getNumbers).toHaveBeenCalled();
06.     });
07. });
```

Домашнє завдання

Написати тести для перевірки функцій з уроків 23, 26, 28.

Для написання тестів використовуйте браузерне оточення

https://github.com/kondolovskiy/test-env.