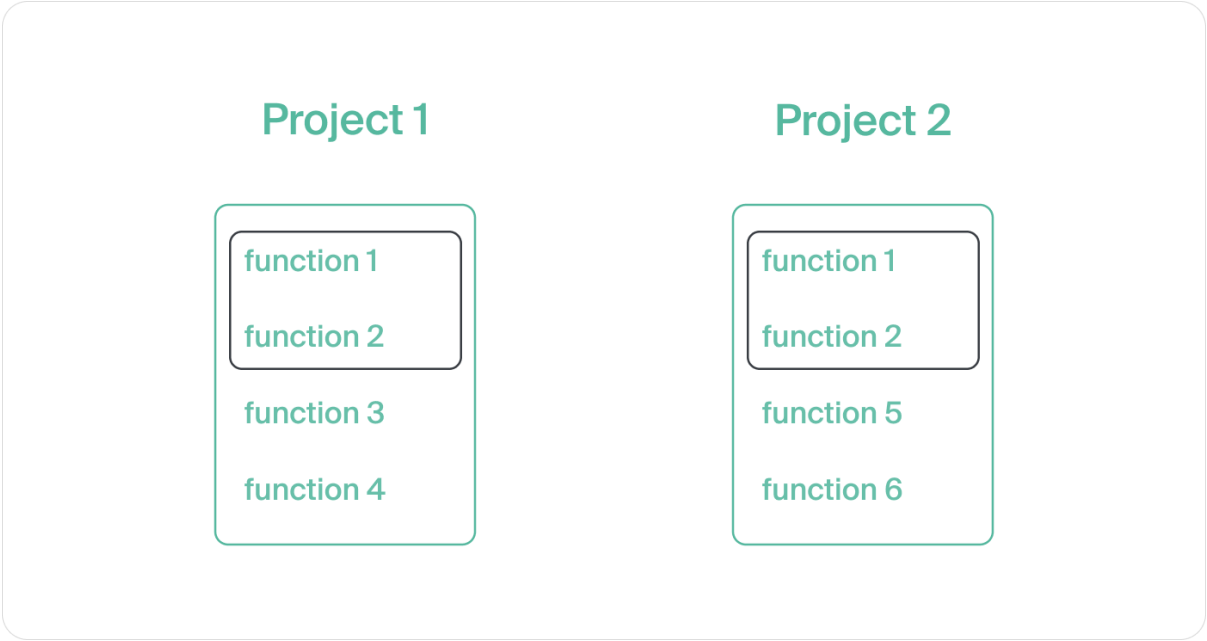
Шпаргалка: автоматизация

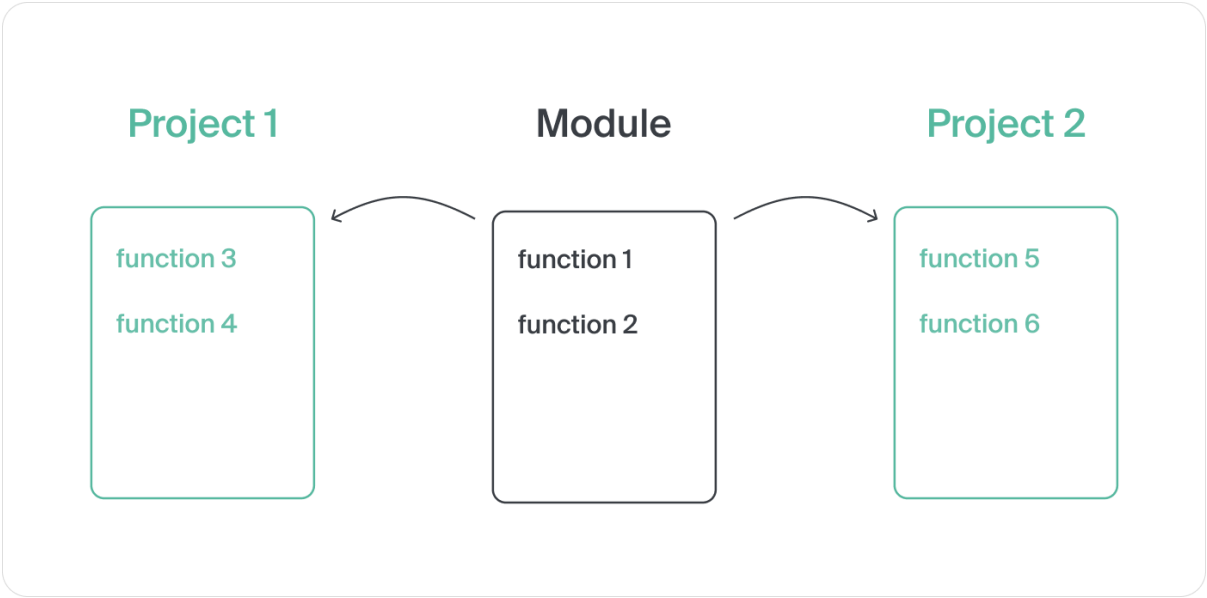
Модули

Модуль — это отдельный файл с кодом. В нём можно объявить переменные, константы или функции, а в другом файле — их применить.

Например: перед тобой — два проекта, Project 1 и Project 2. В обоих задействованы одинаковые функции, `function 1` и `function 2`:



Эти функции можно выделить в отдельный модуль `module`, а потом подключить его Project 1 и Project 2:



Как работает модульная система

Чтобы подключить модуль, примени команду `require`. В аргументе передают название модуля или путь к файлу.

```
const testLib = require('./test-lib.js');
```

Также можно использовать просто имя модуля, если он входит в NodeJS. Чтобы подключить модуль `http` из стандартной библиотеки, напиши команду:

```
const http = require('http');
```

По этой команде модуль подключился. Он позволяет работать с HTTP-запросами. У него есть поле `METHODS`. Если к нему обратиться, ты получишь список всех HTTP-методов: например, POST или GET.

```
const http = require('http');
console.log(http.METHODS); // обратились к METHODS в модуле http
```

Библиотеки

Библиотека — это набор полезных функций, которые можно применить в коде. Часто библиотека посвящена одной тематике.

В NodeJS есть два типа библиотек:

- Внутренняя или стандартная библиотека: встроена в платформу.
- Внешние — сторонние библиотеки: надо устанавливать отдельно.

Список встроенных библиотек можно посмотреть здесь: https://nodejs.org/api/modules.html.

Проверить версию NPM:

```
npm -v
```

`npm install` — установить библиотеку:

```
npm install puppeteer
```

`require` — подключить библиотеку:

```
const puppeteer = require('puppeteer');
```

Структура автотеста

Самый простой автотест, который протестирует интерфейс веб-приложения, структурно похож на тест-кейс. Какие этапы проходит автотест:

1. запуск браузера;
2. выполнение шагов, которые приводят к результату;
3. сравнение ОР и ФР;
4. закрытие браузера.

Запустить браузер методом `launch`:

```
console.log('Запуск браузера');
const browser = await puppeteer.launch(); // запуск браузера
```

В примере результат сохраняется в константу `browser` — она поможет пользоваться запущенным браузером.

Для запуска вызови метод `launch` библиотеки `puppeteer`. Здесь операция `launch` не получит никаких аргументов, поэтому браузер запустится по умолчанию: в режиме headless и с разрешением 800x600.

В `launch` можно передать две полезные опции: отключение безголового режима и замедление выполнения действий — так можно будет увидеть автоматическое выполнение действий в браузере:

```
const browser = await puppeteer.launch({
    headless: false, // выключен безголовый режим
    slowMo: 100, // замедление, чтобы отслеживать выполнение теста
});
```

Закрыть браузер операцией `close()`:

```
console.log('Закрытие браузера');
await browser.close();
```

Создать новую вкладку операцией `newPage()`:

```
console.log('Создание новой вкладки в браузере');
const page = await browser.newPage();
```

Перейти по ссылке командой `goto`:

```
console.log('Переход по ссылке');
await page.goto('<https://qa-routes.praktikum-services.ru/>');
```

Сохранять скриншоты командой `screenshot`:

```
await page.screenshot({path: 'testTaxiResult.png'}); // снятие скриншота
```

Селекторы

Селектор, или локатор — это специальный текстовый запрос. Он помогает найти элемент на веб-странице.

CSS-селектор — это часть языка CSS, которая сообщает браузеру, к каким элементам веб-страницы применить стиль.

HTML-код поля ввода поисковой строки ya.ru выглядит так:

```
<input class="input__control input__input" tabindex="2" autocomplete="off" autocorrect="
off" autocapitalize="off" spellcheck="false" aria-autocomplete="list" aria-label="Запрос
" id="text" maxlength="400" name="text" />
```

ID

`id` — уникальный идентификатор элемента на странице.

```
<input id="text" />
```

Чтобы написать селектор, нужно добавить знак `#` перед названием `id`.

```
#text
```

Классы

В отличие от `id`, один и тот же класс может применяться к нескольким элементам. Чтобы выбрать все элементы с определённым классом, добавь точку перед его названием.

```
.input__control
```

Обрати внимание: у этого элемента два класса. Они разделяются пробелом:

```
<input class="input__control input__input" />
```

Чтобы найти элемент, которому присвоено несколько классов, нужно их «склеить». Тогда селектор будет выглядеть так:

```
.input__control.input__input
```

В отличие от `id`, по классу можно выбрать сразу много элементов.

Другие селекторы

CSS-селекторы — гибкий инструмент, у него много правил. Например, чтобы выбрать все элементы `input` на странице, у которых есть атрибут `type` со значением "text", нужно применить такой селектор:

```
input[type="text"]
```

Как найти элемент

Вызови метод `$` у объекта `page`:

```
const element = await page.$(/* здесь укажи селектор */);
```

```
const minuteInput = await page.$('#form-input-minutes');
```

Type

Эмулировать ввод текста можно методом `type`.

```
const input = page.$(/* здесь укажи селектор поля ввода */);
input.type('Текст'); // ввод текста в поле ввода input
```

Как кликнуть на элемент

Вызови метод `click` у объекта:

```
await element.click();
```

Поиск и **клик** на элемент выбора режима «Свой» может быть выполнен так:

```
const routeMode = await page.$('#form-mode-custom'); // поиск элемента выбора режима «Свой»
await routeMode.click();
```

Метод `waitForSelector` ждёт, пока элемент с указанным селектором появится на странице.

```
await page.waitForSelector('selector');
```

Метод `$eval` отдаёт текст выбранного элемента

```
const text = await page.$eval(<селектор>, <функция>);
```

Функция вызывается с объектом, который Puppeteer сформировал по селектору.

```
const text = await page.$eval('#result-time-price', element => element.textContent);
```

В случаях, когда в аргументе одной функции нужно передать другую функцию, можно применять стрелочные функции.

Посмотри и сравни два варианта.

Классический:

```
function getTextContent(element) {
    return element.textContent;
}
const text = await page.$eval('#result-time-price', getTextContent);
```

Со стрелочной функцией:

```
const text = await page.$eval('#result-time-price', element => element.textContent);
```

Метод `startsWith()` проверяет, что текст начинается с определённой строки. Например, код `'Тест-кейс'.startsWith('Тест')` вернёт `true`.

```
// дождаться, когда элемент с результатом появится на странице
await page.waitForSelector('#result-time-price')

// получить текст этого элемента
const text = await page.$eval('#result-time-price', element => element.textContent);

// проверить, что текст начинается с подстроки «Такси»
if (text.startsWith('Такси')) {
  console.log('Успех. Текст содержит: ' + text);
} else {
  console.log(`Ошибка. Текст не начинается со слова 'Такси'`)
}
```

Если объединить материалы выше, получится такой код:

```
const puppeteer = require('puppeteer');

const URL_TEST = '<https://qa-routes.praktikum-services.ru/>';

async function testTaxiResult() {
    console.log('Запуск браузера');
    const browser = await puppeteer.launch({headless: false, slowMo: 100});

    console.log('Создание новой вкладки в браузере');
    const page = await browser.newPage();

    console.log('Переход по ссылке');
    await page.goto(URL_TEST);

    console.log('Шаг 1: ввод часов и минут');
    const hoursInput = await page.$('#form-input-hour');
    await hoursInput.type('08');
```

```
    const minutesInput = await page.$('#form-input-minute');
    await minutesInput.type('00');

    console.log('Шаг 2: заполнение поля Откуда');
    const fromInput = await page.$('#form-input-from');
    await fromInput.type('Усачева, 3');

    console.log('Шаг 3: заполнение поля Куда');
    const toInput = await page.$('#form-input-to');
    await toInput.type('Комсомольский проспект, 18');

    console.log('Шаг 4: выбор режима Свой');
    const routeMode = await page.$('#form-mode-custom');
    await routeMode.click();

    console.log('Шаг 5: выбор вида транспорта');
    const typeOfTransport = await page.$('#from-type-taxi');
    await typeOfTransport.click();

    console.log('Ожидание элемента с результатом');
    await page.waitForSelector('#result-time-price')

    console.log('Получение строки с результатом');
    const text = await page.$eval('#result-time-price', element => element.textContent);

    console.log('Проверка условия тест-кейса');
    if (text.startsWith('Такси')) {
        console.log('Успех. Текст содержит: ' + text);
    } else {
        console.log(`Ошибка. Текст не начинается со слова 'Такси'`)
    }

    console.log('Закрытие браузера');
    await browser.close();
}

testTaxiResult();
```