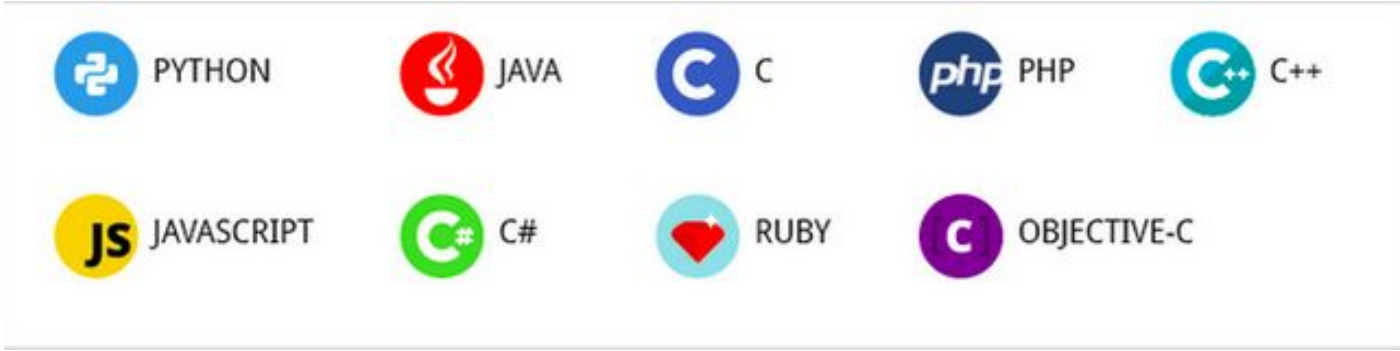
Современные веб-технологии

Презентацию подготовили
Шестакова Полина и Велиева Лейла
ОИС 4 курс

Языки для веб-разработки



В тройку самых популярных входят Javascript, Java и Python

Javascript

Javascript это язык программирования, который поддерживает объектно-ориентированный, императивный и функциональный стили программирования.

В основном используется для создания интерактивных веб-страниц, но с распространением среды Node.js, которая компилирует javascript в машинный код, его стало возможно использовать как **язык общего назначения**, писать на нем серверные и десктопные приложения

Javascript. Особенности

- Динамическая типизация -- одна и та же переменная может принимать значения разных типов
- Автоматическое управление памятью -- js освобождает память за вас
- Прототипное программирование -- вместо классов тут прототипы, которые имеют ряд отличий от привычного по C++ и Java ООП
- Функции как объекты первого класса -- они могут быть переданы как параметр, возвращены из функции, присвоены переменной

Внутренние особенности Javascript

Три важных понятия: call stack (стек вызовов), где хранятся вызовы функций, task queue (очередь заданий), где хранятся коллбэки, и event loop (цикл событий), который выполняет специфическую работу. Он смотрит на стек вызовов и если видит, что он пуст, то начинает брать первую задачу из очереди заданий и добавляет ее в стек вызовов.

Javascript код выполняется в порядке очереди, то есть Last In, First Out. Все вызовы функций добавляются в стек вызовов и выполняются друг за другом.

```
function bar() { console.log('bar'); }
function baz() { console.log('baz'); }
function foo() {
    console.log('foo');
    bar();
    baz();
}
foo();
```


Call Stack
foo()
Call Stack
console.log('foo')
foo()
Call Stack
foo()
Call Stack
bar()
foo()
Call Stack
console.log('bar')
bar()
foo()
Call Stack
bar()
foo()
Call Stack
foo()
Call Stack
baz()
foo()
Call Stack
console.log('baz')
baz()
foo()
Call Stack
baz()
foo()
Call Stack
foo()
Call Stack
Empty

Внутренние особенности Javascript

1. Javascript использует различные сторонние API, например, setTimeout и setInterval. Эти функции занимают много много времени, поэтому JS выполняет, когда стек вызовов пуст, а хранит эти функции в очереди задач.

```
function foo(){console.log(“foo”);}                    foo
setTimeout(function(){console.log(“Hello”)}, 1000); --->    bar
function bar(){console.log(“bar”);}                    Hello
```

Javascript. Callback

Коллбэк – это функция, которая должна быть выполнена после того, как другая функция завершила выполнение

```
function doHomework(subject, callback) {
	console.log(`Starting my ${subject} homework.`);
	callback();
}
function alertFinished(){
	alert('Finished my homework');
}
doHomework('math', alertFinished);
// “Starting my math homework.”
// “Finished my homework!”
```

Javascript. Callback hell

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

Javascript. Promise

Promise – это специальный объект, который содержит своё состояние.
Вначале pending, затем – одно из: fulfilled или rejected.

```
let promise = new Promise((resolve, reject) => {
	If (true)
	setTimeout(() => { resolve(“OK"); }, 1000);
	setTimeout(() => { reject(new Error("error")); }, 2000);
});
promise
	.then(
		result => alert("Fulfilled: " + result), // “Fulfilled: OK”
		error => alert("Rejected: " + error) // “Rejected: error”
	);
```

Javascript. async/await

async и **await** это удобный синтаксис для работы с промисами

Функция, идущая после ключевого слова async, всегда возвращает промис (и даже если она не была промисом, async обернет ее в промис)

```
async function f() {
    return 1;
}

f().then(alert); // 1
```

===>

```
async function f() {
    return Promise.resolve(1);
}

f().then(alert); // 1
```

Javascript. async/await

Ключевое слово **await** работает только внутри async функций. await блокирует исполнение кода внутри async функции и ждет конца выполнения промиса, возвращая его результат.

```
async function f() {
	let promise = new Promise((resolve, reject) => {
		setTimeout(() => resolve("done!"), 1000)
	});
	let result = await promise;
	alert(result); // "done!"
}

f();
```

Javascript. async/await

```
async function showAvatar(username) {
	// read github user
	let githubResponse = await fetch(`https://api.github.com/users/${username}`);
	let githubUser = await githubResponse.json();

	// show the avatar
	let img = document.createElement('img');
	img.src = githubUser.avatar_url;
	img.className = "promise-avatar-example";
	document.body.append(img);
}

showAvatar(username);
```

JS библиотеки и фреймворки для фронтенд-разработки

JS-фреймворки — это инструменты для построения динамических веб/мобильных/настольных приложений на языке Javascript. Они сильно облегчают жизнь разработчику.

Наиболее популярные библиотеки и фреймворки:

- React
- Vue.js
- Angular
- Jquery

Юнит тесты и функциональные тесты

Тест -- это процедура, которая позволяет либо подтвердить, либо опровергнуть работоспособность кода. Тесты нужны, чтобы быть уверенным, что когда вы что-то изменили в своей программе, все остальное работает как и раньше. Бывают юнит тесты и функциональные тесты.

Каждый **юнит тест** проверяет правильность работы одного метода или функции

Функциональные тесты нужны для проверки определенного функционала программы. Это взгляд на программу со стороны пользователя, а не изнутри, как в юнит тестировании. В функциональном тесте может быть задействовано сразу несколько функций или методов.

Зачем тратить на написание тестов время, которое я могу потратить на разработку новой фичи??????

1. Тесты экономят время
2. Код, для которого легко написать юнит тест, это хороший код
3. Тесты это аналог документации

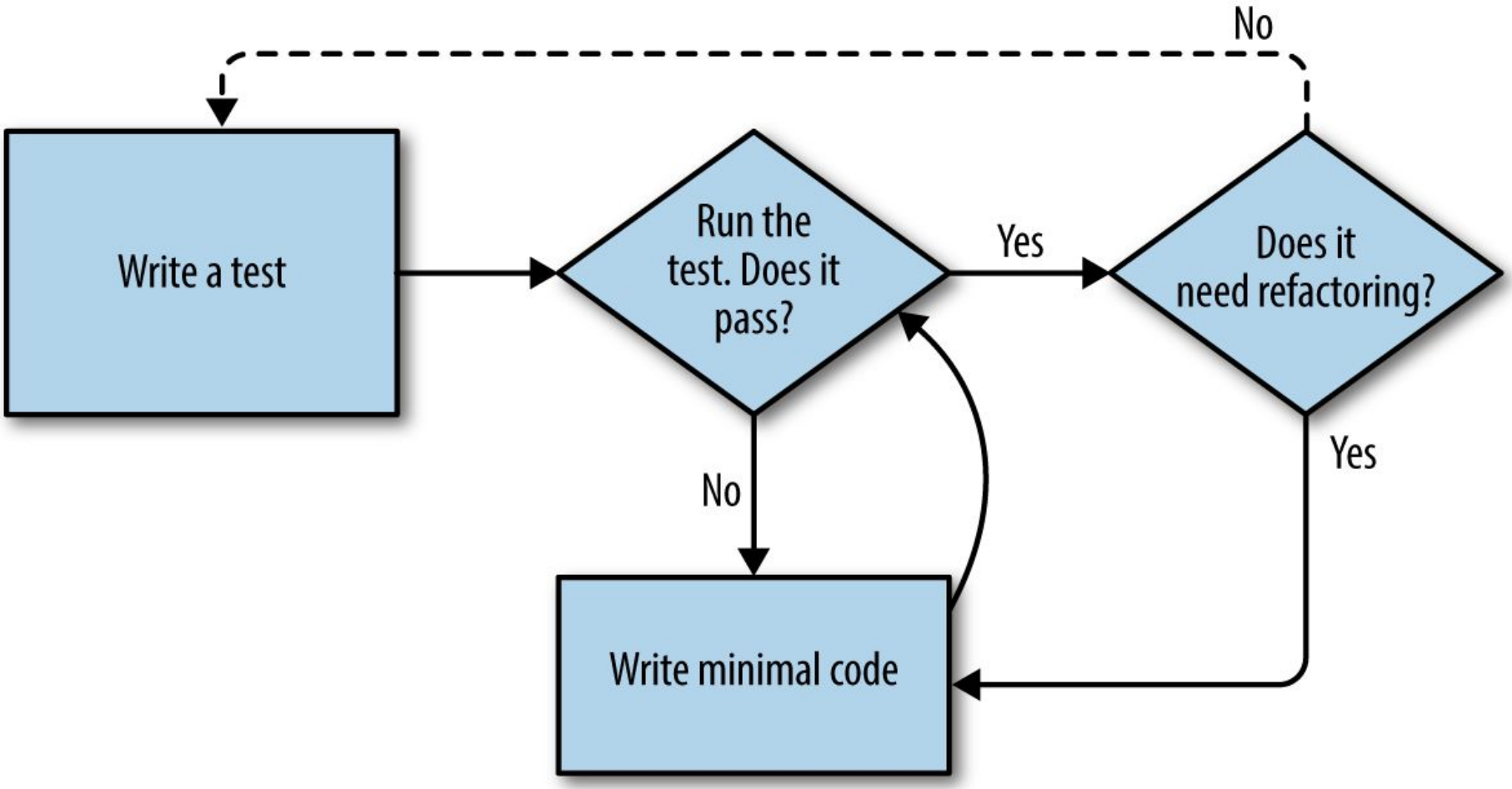
Test-driven development

Разработка через тестирование -- техника разработки

Если вы хотите внести изменение или дополнение в свой код, вы сначала пишете тест, который покрывает это изменение, и только затем пишете сам код, который будет что-то делать.

Принцип TDD “Fake it till you make it”

Когда мы пишем код, который позволит нам пройти тест, мы не пытаемся сделать его идеальным, мы пытаемся сделать его простым. А вот когда тест пройдет, можно будет заняться рефакторингом.


No
Write a test
Run the test. Does it pass?
Yes
Does it need refactoring?
Yes
No
Write minimal code

Test-driven development

Такой подход помогает убедиться, что приложение

1. Пригодно для тестирования, поскольку разработчику придется с самого начала обдумать то, как приложение будет тестироваться.
2. Способствует тому, что тестами будет покрыта вся функциональность. Когда функциональность пишется до тестов, разработчики и организации склонны переходить к реализации следующей функциональности, не протестировав существующую.

Mocha/Chai. Тестирование на javascript

Mocha -- это фреймворк для тестирования кода на javascript

Chai -- это библиотека, расширяющая возможности mocha для написания assert тестов. Можно спокойно писать тесты на чистом Mocha, но chai предлагает несколько разных и гибких подходов

Mocha/Chai

Структура теста выглядит таким образом:

```
describe('Basic Mocha Test', function () {
    it('should return number of characters in a string', function () {
        assert.equal('Hello'.length, 5);
    });
});
```

- **describe** -- это функция, содержащая коллекцию тестов. Первым аргументом является название этой коллекции, а вторым -- функция, в которой содержатся отдельные тесты
- **it** -- это функция, принимающая как аргументы название теста и функцию, которая содержит “тело” теста
- **assert** -- определяет, прошел тест или нет, возвращая true или false

Babel.js

ECMAScript -- стандарт для javascript, новая версия которого выходит практически каждый год. Разработчики не успевают внедрить новый стандарт в браузеры, и из-за этого мы не можем сразу же применять все, что нам понравилось, из новой спецификации.

Здесь может пригодиться babel.js, который транслирует код, написанный с использованием новых стандартов, в код, совместимый с большинством живых на текущий момент браузеров.

Put in next-gen JavaScript

```
[1, 2, 3].map(n => n ** 2);
```

Get browser-compatible JavaScript out

```
[1, 2, 3].map(function (n) {
  return Math.pow(n, 2);
});
```

Node.js

Node.js – это среда для серверной разработки на JavaScript.

- быстрый
- прост в освоении
- работает асинхронно
- предоставляет кучу различных пакетов

Где используется?

- VK
- Paypal
- LinkedIn
- Yahoo
- Netflix

Подробнее можно прочитать тут https://brainhub.eu/blog/9-famous-apps-using-node-js/

Что можно писать на Node.js

- Серверы веб-приложений
- Серверы сообщений и трансляция событий (чаты, игры, интерактив)
- Любые веб-API (JSON, REST)
- Одностраничные приложения (SPA)
- Программирование микроконтроллеров (arduino, espruino, tessel)

Node Addon API (N-API)

Для Node.js можно писать аддоны на C и C++, если использовать для этого специальное API (или с помощью библиотек. Причиной для написания аддона может быть:

- Перенос сложного алгоритма на C++ для повышения производительности
- Желание использовать библиотеку, написанную на C++, в среде Node.js

Freeling аддон для Node.js

FreeLing -- это библиотека на C++, предоставляющая средства для обработки естественного языка, такие как морфологический анализ, распознавание именованных сущностей, разметка частей речи, парсинг, разрешение лексической многозначности, присвоение семантических ролей и т.д. Кроме английского и русского поддерживается еще около 10 языков.

В аддоне для Node.js на данный момент поддерживается морфологический анализ и разметка частей речи.

Репо: https://github.com/Pauline-sh/freeling-nodejs-addon

Полезные статьи. Javascript

30 идей для небольших проектов на native javascript
https://javascript30.com/
Что такое прототипы
https://yehudakatz.com/2011/08/12/understanding-prototypes-in-javascript/
Event loop
https://flaviocopes.com/javascript-event-loop/
“this” в javascript
https://yehudakatz.com/2011/08/11/understanding-javascript-function-invocation-and-this/

Полезные статьи. Node.js

Веб сервер на node.js без зависимостей
https://blog.bloomca.me/2018/12/22/writing-a-web-server-node.html
Особенности node.js по сравнению с браузерным javascript
https://blog.bloomca.me/2018/06/21/nodejs-guide-for-frontend-developers.html
Введение в node.js
https://nodejs.dev/

Полезные статьи. Тестирование

Test-driven-development на Python
http://www.obeythetestinggoat.com/book/praise.harry.html
Unit-testing для javascript
https://codeburst.io/javascript-unit-testing-using-mocha-and-chai-1d97d9f18e71

Полезные статьи. Babel и фронтэнд для динозавров

Официальный сайт babel.js с отличным демо
https://babeljs.io/
Объяснение в комплексе экосистемы javascript для фронтэнд разработки
https://medium.com/the-node-js-collection/modern-javascript-explained-for-dinosaurs-f695e9747b70