

# Операторы цикла, функции и объекты

Василий Петров  
Разработчик Python, JavaScript



# Василий Петров

О спикере:

- Стаж работы в ИТ более 25 лет
- Разрабатывал корпоративные приложения
- Руководил проектами и ИТ-подразделениями
- Руководил собственным бизнесом
- В настоящее время участвую в различных проектах с применением Python и JavaScript



# Вспоминаем прошное занятие

**Вопрос:** какой тип в JavaScript аналогичен спискам в Python?



# Вспоминаем прошное занятие

**Вопрос:** какой тип в JavaScript аналогичен спискам в Python?

**Ответ:** аналог списков в JavaScript — массивы (**Array**)



# Вспоминаем прошрое занятие

**Вопрос:** какой оператор в JavaScript используется для множественного ветвления?



# Вспоминаем прошрое занятие

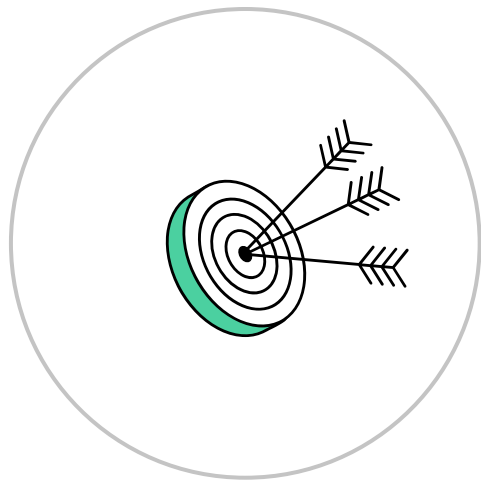
**Вопрос:** какой оператор в JavaScript используется для множественного ветвления?

**Ответ:** оператор **switch**. Аналогичный оператор отсутствует в Python



# Цели занятия

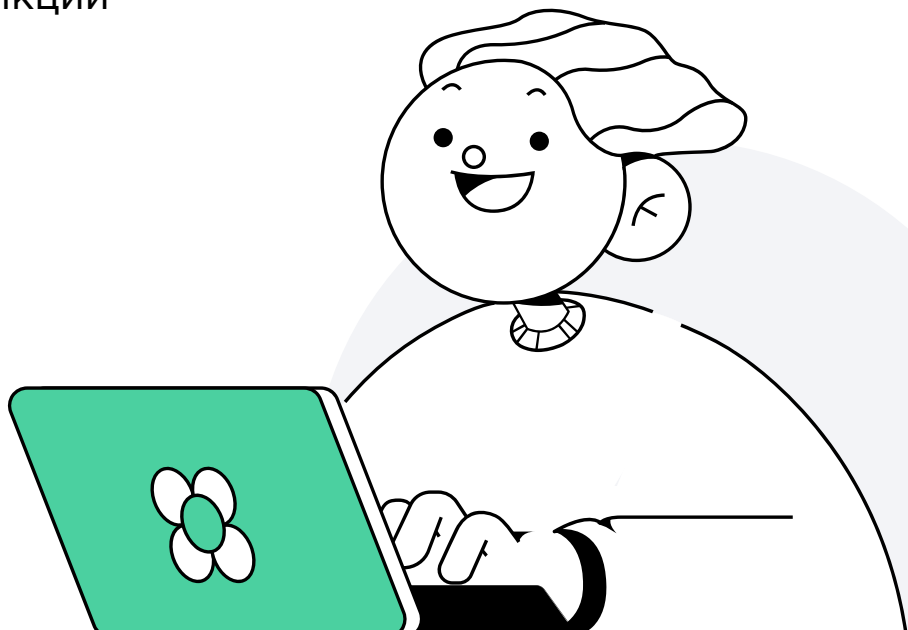
- Познакомимся с платформой NodeJS
- Изучим различные операторы цикла в JS
- Изучим синтаксис описания функций и их параметров
- Напишем программу, запускаемую в NodeJS
- Познакомимся с объектами и посмотрим, как можно их использовать в прототипе интернет-магазина



# План занятия

- 1 Платформа NodeJS. Установка и первые шаги
- 2 Операторы цикла. **For, while, do while**
- 3 Классический синтаксис описания функций
- 4 Объекты в JS
- 5 Итоги
- 6 Домашнее задание

\*Нажми на нужный раздел для перехода






# Платформа NodeJS

Установка и первые шаги



1



**NodeJS** — платформа для  
выполнения кода на JS

# NodeJS

Разработана в 2009 году Райаном Далем на основе движка **Google V8**.

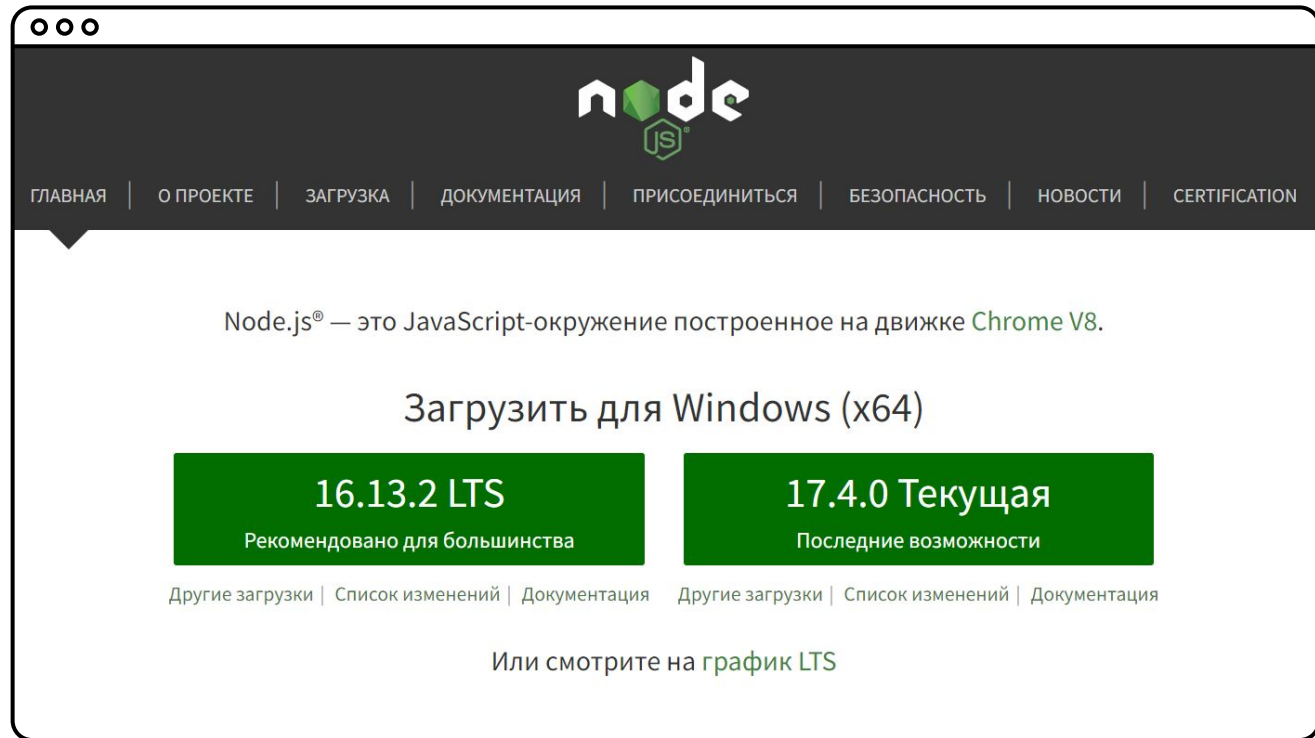
Node JS включает в себя библиотеку **libuv** для асинхронного выполнения функций ввода-вывода и организации цикла событий (**event-loop**).

Запуск программ в NodeJS аналогичен использованию **интерпретатора Python** в командной строке.


Существует под все основные операционные системы **Windows, MacOS, Linux** и аппаратные платформы

# Установка NodeJS

Инсталляционный пакет и инструкции по установке: [Node.js](https://nodejs.org/)



ooo



ГЛАВНАЯ | О ПРОЕКТЕ | ЗАГРУЗКА | ДОКУМЕНТАЦИЯ | ПРИСОЕДИНИТЬСЯ | БЕЗОПАСНОСТЬ | НОВОСТИ | CERTIFICATION

Node.js® — это JavaScript-окружение построенное на движке [Chrome V8](#).

Загрузить для Windows (x64)

<b>16.13.2 LTS</b> Рекомендовано для большинства	<b>17.4.0 Текущая</b> Последние возможности
---	--

[Другие загрузки](#) | [Список изменений](#) | [Документация](#) | [Другие загрузки](#) | [Список изменений](#) | [Документация](#)

Или смотрите на график [LTS](#)

# Дополнительные инструменты

Устанавливаются вместе с платформой:

- **npm** (node package manager) — утилита для управления зависимостями проекта. Аналогична **pip**, но имеет более широкий набор функций. С помощью npm создаётся среда проекта, аналогично **venv** в Python
- **npx** — утилита для запуска программ, написанных на JS без установки в систему или проект.

Дополнительные модули и утилиты публикуются на сайте [npmjs.com](https://npmjs.com) и могут быть подключены к проекту, используя команды утилиты **npm**

# Проверка установки и режим REPL

Проверяем установку:

```
> node --version  
v17.4.0
```

Запускаем режим интерактивного выполнения команд:

```
> node  
Welcome to Node.js v17.4.0.  
Type ".help" for more information.  
> console.log('Hello, World!')  
Hello, World!  
undefined  
> .exit
```

# Запуск файла .js

Запишем код в файл index.js

```
console.log('Hello, World');
```

Запустим:

```
> node index.js  
Hello, World!
```

Отлично, теперь мы умеем запускать код на JS в среде NodeJS

# Операторы цикла

For, while, do while



2



# Циклы в JavaScript

Набор вариантов циклов в JavaScript шире, чем в Python:

- Цикл со счетчиком **for (;;) {}**
- Цикл с предусловием **while () {}**
- Цикл с постусловием **do {} while ()**
- Циклы с итератором **for ( in ) {}** и **for ( of ) {}** — в этом курсе не рассматриваем

# Цикл со счетчиком for

Аналогичен циклу **for a in range(...)** в Python:

```
for (<инициализация>; <условие>; <действие>) {  
  <тело цикла>  
}
```

Пример для вывода чисел от 0 до 9 в консоль:

```
for (let i = 0; i < 10; i++) {  
  console.log(i);  
}
```

# Цикл с предусловием while

Аналогичен циклу **while** в Python:

```
while (<условие>) {  
    <тело цикла>  
}
```

Пример для вывода чисел от 0 до 9 в консоль:

```
let i = 0;  
while (i < 10) {  
    console.log(i);  
    i++;  
}
```

# Цикл с постусловием do while

Аналогичный вариант цикла в Python отсутствует:

```
while (<условие>) {  
    <тело цикла>  
}
```

Пример для вывода чисел от 0 до 9 в консоль:

```
let i = 0;  
do {  
    console.log(i);  
    i++;  
} while (i < 10)
```

Условие проверяется только после выполнения тела цикла

# Классический синтаксис описания функций



3

# Функции в JavaScript

Используются в JS аналогично функциям в Python для выделения блоков кода, которые могут быть повторно использованы путём вызова с различными аргументами.

В JS существует несколько вариантов описания функций:

- 1 Классический **function a(...) {}**.
- 2 Функциональное выражение **const a = function (...) {}**.
- 3 Стрелочная функция **(...) => {}**

# Функции в JavaScript

Функции в JS являются объектами первого класса (first-class objects), то есть могут быть:

- Записаны в переменные
- Переданы в качестве аргументов в другие функции и возвращены в качестве результата

В практике разработки на JS функциональное программирование используется значительно шире, чем на Python

# Классический синтаксис описания функций

```
function <имя> (<параметры>) {  
  <тело функции>  
  return <возвращаемое значение>  
}
```

Например, функция сложения двух чисел:

```
function sum(a, b) {  
  return a+b;  
}
```

Функции, описанные классическим синтаксисом, доступны для вызова в любом месте кода скрипта, даже до их описания



# Функциональное выражение

```
const <имя> = function (<параметры>) {  
  <тело функции>  
  return <возвращаемое значение>  
}
```

Например, функция сложения двух чисел:

```
const sum = function (a, b) {  
  return a+b;  
}
```

Так как в переменную записывается ссылка на создаваемую функцию, то использовать её можно будет только после присваивания

# Параметры функции

В отличие от Python соответствие количества описанных параметров и передаваемых аргументов не контролируется.

Следующий код не вызывает ошибку времени выполнения:

```
function sum(a, b) { return a+b; }  
const c = sum(); // отсутствующие аргументы = undefined  
const d = sum(0, 1, 2, 3, 4, 5); // лишние аргументы игнорируются
```

Все параметры в JS — позиционные, отсутствуют ключевые именованные параметры, аналогичные используемым в Python. В JavaScript для той же цели используется передача объектов с именованными полями.

# Параметры функции

Функция может принимать переменное количество параметров:

- 1 Внутри функции доступен псевдомассив **arguments**[] со всеми аргументами.
- 2 Можно объявить последний параметр с помощью оператора rest:

```
function sum(a, b, ...c) {}
```

В этом примере через c будет доступен массив переданных аргументов кроме первых двух.

Для последних параметров возможно задание значений по умолчанию:

```
function calc(a, b, operation = 'sum') {}
```

# Возвращаемый результат

Для возврата значения из функции используется оператор **return**:

```
return <значение>;
```

Может быть использован в любом месте кода функции, выполнение функции завершается выполнением оператора **return**.

В случае, если функция завершается без выполнения оператора **return**, возвращаемое значение равно **undefined**

# Рекурсивные вызовы функций

Функция может вызывать сама себя для выполнения рекурсивных алгоритмов. Например, вычисление факториала:

```
function factorial(n) {  
  if (n === 1) return 1;  
  return n * factorial(n-1);  
}
```

Если функция создается как функциональное выражение, то возможно в нём задать ей имя, которое может быть использовано только внутри этой функции:

```
const factorial = function f(n) {  
  if (n === 1) return 1;  
  return n * f(n-1);  
}
```

Рекурсию необходимо использовать с осторожностью, максимальный уровень вложенности ограничен движком JS

# Демонстрация

Давайте напишем программу для  
вычисления чисел Фибоначчи



# Объекты в JS

4



# Объекты в JavaScript

В JavaScript, аналогично Python, все данные так или иначе хранятся и обрабатываются в виде объектов.

Для примитивных типов это не всегда заметно, пока мы не используем методы объектов этих типов:

```
(1).toString();  
("HELLO").toLowerCase()
```



# Объекты в JavaScript

Объекты в JS похожи на словари в Python, но имеют существенные отличия:

- Именем свойства или метода может быть только строка. При использовании не строк в качестве имён они по возможности будут преобразованы в строковое значение
- Для доступа к свойствам и методам можно использовать два синтаксиса:

`<объект>['<имя>']`, например `client['name']` — используется для вычисляемых во время выполнения имен свойств и методов и для имен, содержащих специальные символы

`<объект>.<имя>`, например `client.name` — используется в случае, если имя свойства известно на этапе написания кода

# Объекты в JavaScript

В отличие от словарей Python, объекты используются в качестве экземпляров классов в объектно-ориентированном программировании.

Но устройство механизмов реализации ООП в JS значительно отличается от Python и других языков (через прототипы).

Эти механизмы будут рассмотрены на следующем вебинаре

# Создание объекта

Объекты можно создавать инициализируя их содержимое, например:

```
const client = {  
  name: 'Ivan Ivanov',  
  'e-mail': 'ivan@bigcorp.com',  
  password: 'qwerty',  
}
```

Обратите внимание на запятую после последнего свойства. Это разрешено в JS, более того, считается хорошим стилем, так как облегчает добавление свойств в коде.

В отличие от Python, имена свойств не обязательно заключать в кавычки, если они содержат только разрешённые для идентификаторов символы: 'e-mail' должно быть в кавычках

# Операции с объектами

Чтение и запись свойств:

```
client1.name = client2['name'];
```

Добавление в объект нового свойства:

```
client.position = 'CEO';
```

Удаление свойства из объекта:

```
delete client.position;
```

# Методы объектов, переменная **this**

В свойствах объектов могут храниться ссылки на функции, в этом случае они становятся методами объекта, которые можно вызывать и использовать для обработки данных объекта, для которого они вызываются.

В этом случае ссылка на объект в них доступна через специальную переменную **this**:

```
client.setPosition = function (position) {  
  this.position = position;  
};  
client.setPosition('CEO');
```

Но **this** в JS существенно отличается от **self** внутри методов в Python

# Работа со значениями и ссылками

Примитивные типы всегда обрабатываются в JS по значению.

```
let a = 1;  
let b = a;  
b = 3; // Значение в a не меняется
```

Сложные типы, такие как массивы и объекты, всегда по ссылке.

```
const a = { name: 'Ivan', };  
const b = a;  
a.name = 'Petr'; //Значение в b.name меняется, т.к. а и b ссылаются на один объект
```

В функции значения сложных типов также всегда передаются по ссылке, т. е. могут быть изменены в коде функции. Объявление через **const** не защищает от изменений внутри объекта или массива, только от замены ссылки на другой объект или массив.

# Демонстрация

Давайте создадим объекты для нашего  
будущего интернет-магазина



# Итоги

Сегодня мы:

- 1 Познакомились с платформой NodeJS
- 2 Изучили различные варианты оператора цикла
- 3 Изучили синтаксис описания функций и их параметров
- 4 Изучили базовые возможности создания объектов и операций с ними

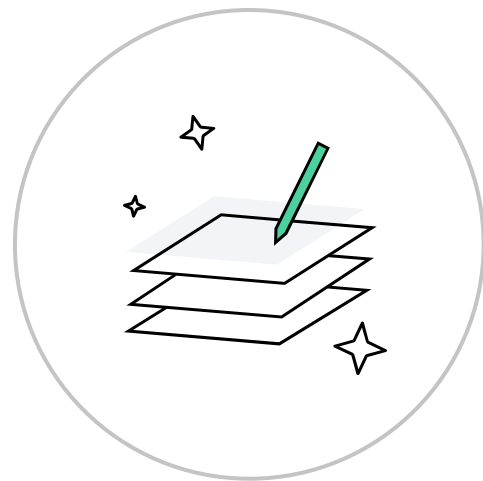




# Домашнее задание

Давайте посмотрим ваше домашнее задание.

- 1 Вопросы по домашней работе задавайте в чате группы
- 2 Задачи можно сдавать по частям
- 3 Зачёт по домашней работе ставят после того, как приняты все задачи



# Дополнительные материалы

- [Документация](#) по NodeJS
- [Циклы](#) в учебнике по JS
- [Функции](#) в учебнике по JS
- [Объекты](#) в учебнике по JS



# Задавайте вопросы и пишите отзыв о лекции

Василий Петров  
Разработчик Python, JavaScript

