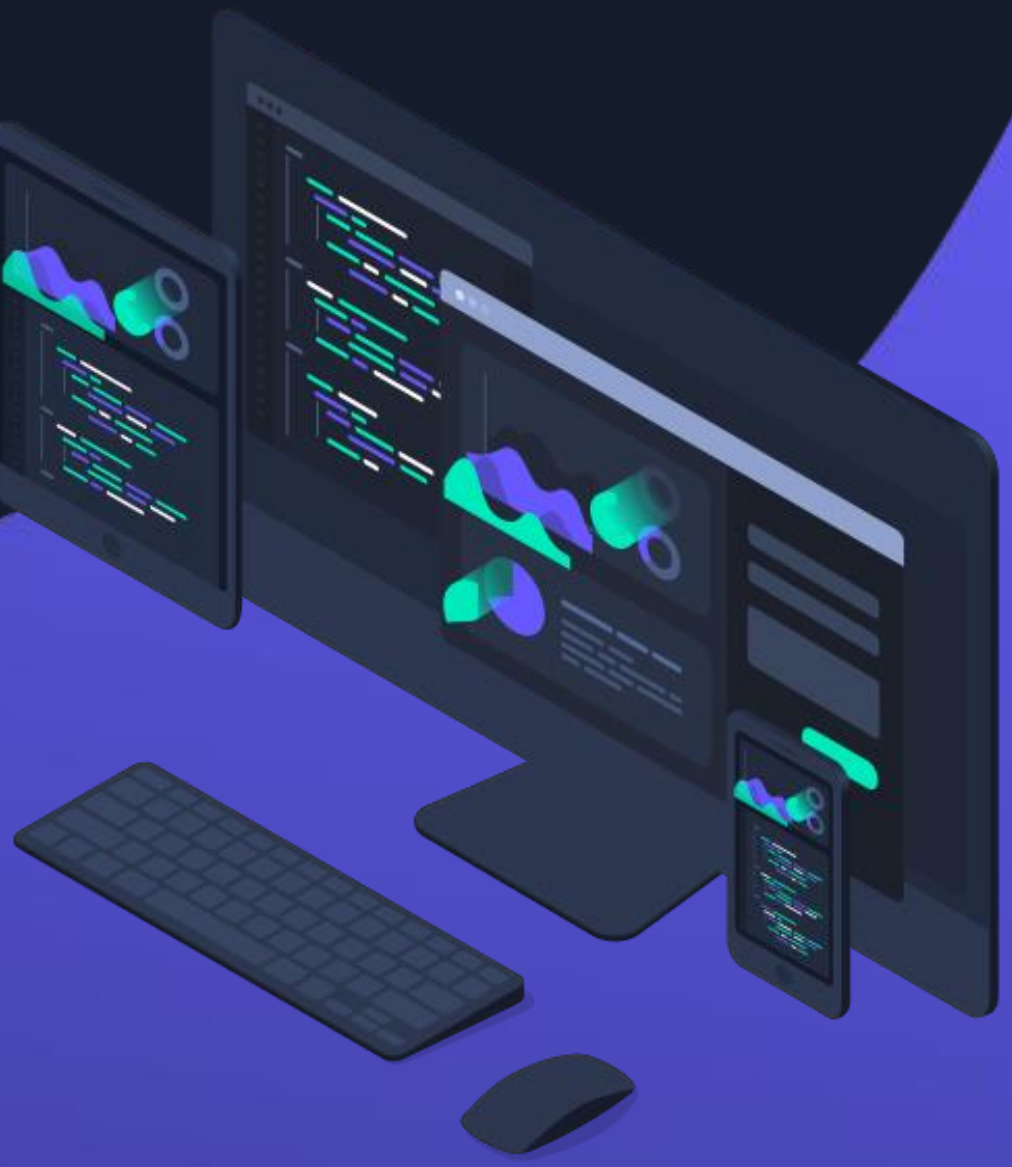
ITEA

ONLINE



ITEA ONLINE

# На сегодняшнем занятии:

1. Функция-конструктор
2. Создание методов в конструкторе
3. Object.keys, values, entries
4. Копирование объектов и ссылки
5. Ключевое слово «this» в методах
6. У стрелочных функций нет «this»
7. setTimeout и setInterval
8. Обработка ошибок, "try..catch"
9. window.location
10. window.history

ITEA ONLINE

# Функция-конструктор

Функции-конструкторы являются обычными функциями. Но есть два соглашения:

- Имя функции-конструктора должно начинаться с большой буквы.
- Функция-конструктор должна вызываться при помощи оператора "new".

```
function User(name) {
  this.name = name;
  this.isAdmin = false;
}
let user = new User("Вася");
```

```
let user = {
  name: "Вася",
  isAdmin: false
};
```

Когда функция вызывается как `new User(...)`, происходит следующее:

- Создаётся новый пустой объект, и он присваивается this.
- Выполняется код функции. Обычно он модифицирует this, добавляет туда новые свойства.
- Возвращается значение this.

ITEA ONLINE

# Функция-конструктор

Другими словами, вызов `new User(...)` делает примерно вот что:

```
function User(name) {
  // this = {};  (неявно)

  // добавляет свойства к this
  this.name = name;
  this.isAdmin = false;

  // return this;  (неявно)
}
```

Теперь, когда нам необходимо будет создать других пользователей, мы можем использовать `new User("Маша")`, `new User("Даша")` и т.д. Данная конструкция гораздо удобнее и читабельнее, чем каждый раз создавать литерал объекта. Это и является основной целью конструкторов – удобное повторное создание однотипных объектов.

ITEA ONLINE

# Создание методов в конструкторе

Использование конструкторов для создания объектов даёт большую гибкость. Можно передавать конструктору параметры, определяющие, как создавать объект, и что в него записывать.

```
function User(name) {
  this.name = name;

  this.sayHi = function() {
    alert( "Меня зовут: " + this.name );
  };
}

let vasya = new User("Вася");
vasya.sayHi(); // Меня зовут: Вася
```

ITEA ONLINE

# Object.keys, values, entries

Это универсальные методы, и существует общее соглашение использовать их для структур данных. Для простых объектов доступны следующие методы:

```
Object.keys(obj) – возвращает массив ключей.
Object.values(obj) – возвращает массив значений.
Object.entries(obj) – возвращает массив пар [ключ, значение].
```

```
let user = {
  name: "John",
  age: 30
};
```

```
Object.keys(user) = ["name", "age"]
Object.values(user) = ["John", 30]
Object.entries(user) = [ ["name","John"], ["age",30] ]
```

ITEA ONLINE

# Копирование объектов и ссылки

Одним из фундаментальных отличий объектов от примитивных типов данных является то, что они хранятся и копируются «по ссылке». Примитивные типы: строки, числа, логические значения – присваиваются и копируются «по значению».

```
let message = "Привет!";
let phrase = message; // копируется значение

console.log(message);     // "Привет!"
console.log(phrase);      // "Привет!"

phrase += " Как дела?";
console.log(message);     // "Привет!"
console.log(phrase);      // "Привет! Как дела?"
```

В результате мы имеем две независимые переменные, каждая из которых хранит строку "Привет!". Если мы изменим одну из переменных – то вторая не изменится.

Объекты ведут себя иначе.

I T E A
ONLINE

# Копирование объектов и ссылки

Переменная хранит не сам объект, а его «адрес в памяти», другими словами «ссылку» на него.

```
let user = { name: "Иван" };
let admin = user; // копируется ссылка

console.log(user.name);   // "Иван"
console.log(admin.name);  // "Иван"

admin.name = "Саша";
console.log(user.name);   // "Саша"
console.log(admin.name);  // "Саша"
```

Сам объект хранится где-то в памяти. А в переменной user лежит «ссылка» на эту область памяти. Когда переменная объекта копируется – копируется ссылка, сам же объект не дублируется.

Если мы представляем объект как ящик, то переменная – это ключ к нему. Копирование переменной дублирует ключ, но не сам ящик.

I T E A
ONLINE

# Копирование объектов и ссылки

Если мы все таки хотим продублировать объект, то нам нужно создавать новый объект и повторять структуру дублируемого объекта, перебирая его свойства и копируя их

```
let user = { name: "Vlad", age: 30 };

let clone = {}; // новый пустой объект

// скопируем все свойства user в него
for (let key in user) {
  clone[key] = user[key];
}

// теперь в переменной clone находится абсолютно независимый клон объекта
clone.name = "Sasha"; // изменим в нём данные

console.log(user.name); // в оригинальном объекте значение свойства `name` осталось прежним
```

I T E A
ONLINE

# Копирование объектов и ссылки

Еще один способ, это использовать метод Object.assign:

```
Object.assign(dest, [src1, src2, src3...])
```

Первый аргумент dest — целевой объект.

Остальные аргументы src1, ..., srcN (может быть столько, сколько нужно) являются исходными объектами

Метод копирует свойства всех исходных объектов src1, ..., srcN в целевой объект dest. То есть, свойства всех перечисленных объектов, начиная со второго, копируются в первый объект.

Возвращает объект dest.

# Копирование объектов и ссылки

Например, объединим несколько объектов в один:

```
let user = { name: "Vlad" };

let src1 = { age: 30, additional: { surname: 'Sydorchuk'} };
let src2 = { isDeveloper: true };
let src3 = { habits: ["run", "youtube"] };

// копируем все свойства из src1...src3 в user
Object.assign(user, src1, src2, src3);

console.log(user);
```

```
{
  additional: {
    surname: "Sydorchuk"
  },
  age: 30,
  habits: ["run", "youtube"],
  isDeveloper: true,
  name: "Vlad"
}
```

Если принимающий объект (user) уже имеет свойство с таким именем, оно будет перезаписано
Если копируется свойство-объект, то оно будет скопировано по ссылке (!)

I T E A
ONLINE

# Ключевое слово «this» в методах

Как правило, методу объекта необходим доступ к информации, которая хранится в объекте, чтобы выполнить с ней какие-либо действия (в соответствии с назначением метода).

```
let user = {
  name: "Джон",
  age: 30,

  sayHi() {
    // this - это "текущий объект"
    alert(this.name);
  }
};

user.sayHi(); // Джон
```

Например, коду внутри `user.sayHi()` может понадобиться имя пользователя, которое хранится в объекте `user`.

Для доступа к информации внутри объекта метод может использовать ключевое слово `this`.

Значение `this` – это объект «перед точкой», который использовался для вызова метода.

ITEA ONLINE

# У стрелочных функций нет «this»

Стрелочные функции особенные: у них нет своего «собственного» `this`. Если мы используем `this` внутри стрелочной функции, то его значение берётся из внешней «нормальной» функции. Например, здесь `arrow()` использует значение `this` из внешнего метода `user.sayHi()`:

```
let user = {
  firstName: "Илья",
  testFunc: () => console.log(this.firstName),
  sayHi() {
    let arrow = () => alert(this.firstName);
    arrow();
  }
};

user.sayHi(); // Илья
user.testFunc(); // undefined
```

Это является особенностью стрелочных функций. Они полезны, когда мы на самом деле не хотим иметь отдельное значение `this`, а хотим брать его из внешнего контекста.

ITEA ONLINE

# setTimeout и setInterval

Мы можем вызвать функцию не в данный момент, а позже, через заданный интервал времени.
Это называется «планирование вызова».

Для этого существуют два метода:

- `setTimeout` позволяет вызвать функцию один раз через определённый интервал времени.
- `setInterval` позволяет вызывать функцию регулярно, повторяя вызов через определённый интервал времени.

ITEA ONLINE

# setTimeout

Синтаксис:

```
let timerId = setTimeout(func|code, [delay], [arg1], [arg2], ...)
```

Параметры:

- `func|code` - функция или строка кода для выполнения. Обычно это функция. По историческим причинам можно передать и строку кода, но это не рекомендуется.
- `delay` - задержка перед запуском в миллисекундах (1000 мс = 1 с). Значение по умолчанию – 0.
- `arg1`, `arg2`… - аргументы, передаваемые в функцию

ITEA ONLINE

# setTimeout

Например, данный код вызывает `sayHi()` спустя одну секунду:

```
function sayHi() {
  alert('Привет');
}

setTimeout(sayHi, 1000);
```

С аргументами:

```
function sayHi(phrase, who) {
  alert( phrase + ', ' + who );
}

setTimeout(sayHi, 1000, "Привет", "Джон"); // Привет, Джон
```

ITEA ONLINE

# setTimeout

Передавайте функцию, но не запускайте её
Начинающие разработчики иногда ошибаются, добавляя скобки `()` после функции:

```
// не правильно!
setTimeout(sayHi(), 1000);
```

Это не работает, потому что `setTimeout` ожидает ссылку на функцию. Здесь `sayHi()` запускает выполнение функции, и результат выполнения отправляется в `setTimeout`.

ITEA ONLINE

# Отмена через clearTimeout

Вызов `setTimeout` возвращает «идентификатор таймера» `timerId`, который можно использовать для отмены дальнейшего выполнения.
Синтаксис для отмены:

```
let timerId = setTimeout(...);
clearTimeout(timerId);
```

К сожалению, нет единой спецификации на эти методы, поэтому такое поведение является нормальным.

ITEA ONLINE

# setInterval

Метод `setInterval` имеет такой же синтаксис как `setTimeout`:

```
let timerId = setInterval(func|code, [delay], [arg1], [arg2], ...)
```

Все аргументы имеют такое же значение. Но отличие этого метода от `setTimeout` в том, что функция запускается не один раз, а периодически через указанный интервал времени.

Чтобы остановить дальнейшее выполнение функции, необходимо вызвать `clearInterval(timerId)`

ITEA ONLINE

# Обработка ошибок, "try..catch"

Обычно скрипт в случае ошибки «падает» (сразу же останавливается), с выводом ошибки в консоль.
Но есть синтаксическая конструкция `try..catch`, которая позволяет «ловить» ошибки и вместо падения делать что-то более осмысленное.
Конструкция `try..catch` состоит из двух основных блоков: `try`, и затем `catch`:

```
try {
  // код...
} catch (err) {
  // обработка ошибки
}
```

I T E A
ONLINE

# Обработка ошибок, "try..catch"

Когда возникает ошибка, JavaScript генерирует объект, содержащий её детали. Затем этот объект передаётся как аргумент в блок `catch`:

```
try {
  lalala;
} catch (err) { // <-- объект ошибки, можно использовать другое название вместо err
  console.log(err.name);
  console.log(err.message);
}
```

- `name` - имя ошибки. Например, для неопределённой переменной это "ReferenceError".
- `message` - текстовое сообщение о деталях ошибки.

I T E A
ONLINE

# window.location

Window Location

- `window.location.href` возвращает href (URL) текущей страницы
- `window.location.hostname` возвращает доменное имя веб-хоста
- `window.location.pathname` возвращает путь и имя файла текущей страницы
- `window.location.protocol` возвращает используемый веб-протокол (http: или https :)
- `window.location.assign` загружает новый документ

I T E A
ONLINE

# window.history

Window History

Объект `window.history` содержит историю браузеров.
Для защиты конфиденциальности пользователей существуют ограничения на то, как JavaScript может получить доступ к этому объекту.
Некоторые методы:

- `history.back()` - то же самое, что и щелчок назад в браузере
- `history.forward()` - то же самое, что и щелчок вперед в браузере

ITEA ONLINE

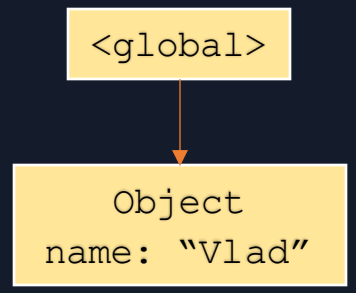
# Сборщик мусора

Управление памятью в JavaScript выполняется автоматически и незаметно. Мы создаём примитивы, объекты, функции… Всё это занимает память. Основной концепцией управления памятью в JavaScript является принцип достижимости. Если упростить, то «достижимые» значения – это те, которые доступны или используются. Они гарантированно находятся в памяти. В интерпретаторе JavaScript есть фоновый процесс, который называется сборщик мусора. Он следит за всеми объектами и удаляет те, которые стали недостижимы..

# Сборщик мусора

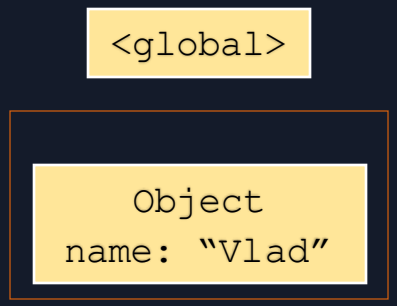
Простой пример:

```
// в person находится ссылка на объект
let person = {
  name: "Vlad"
};
```




Здесь стрелка обозначает ссылку на объект. Глобальная переменная person ссылается на объект {name: “Vlad"}. В свойстве "name" объекта person хранится примитив, поэтому оно нарисовано внутри объекта.

```
person = null;
```




Ссылки больше нет значит и доступа к объекту нет. Объект стал недостижимым и будет удален сборщиком мусора

ITEA ONLINE

# Дата и время

Объект: Date содержит дату и время, а также предоставляет методы управления ими. Например, его можно использовать для хранения времени создания/изменения, для измерения времени или просто для вывода текущей даты.

Для создания нового объекта Date нужно вызвать конструктор new Date() с одним из следующих аргументов:

`new Date()-` без аргументов – создаст объект Date с текущими датой и временем:

```
let now = new Date();
console.log(now); // показывает текущие дату и время
```

ITEA ONLINE

# Дата и время

Конструктор объекта Date может принимать до 7 параметров

```
new Date(year, month, date, hours, minutes, seconds, ms)
```

Обязательны только первые два аргумента.

- year должен состоять из четырёх цифр: значение 2013 корректно, 98 – нет.
- month начинается с 0 (январь) по 11 (декабрь).
- Параметр date здесь представляет собой день месяца. Если параметр не задан, то принимается значение 1.
- Если параметры hours/minutes/seconds/ms отсутствуют, их значением становится 0.

```
new Date(2011, 0, 1, 0, 0, 0, 0); // // 1 Jan 2011, 00:00:00
new Date(2011, 0, 1); // то же самое, так как часы и проч. равны 0
```

ITEA ONLINE

# Дата и время

Получение компонентов даты

`getFullYear()` – получить год (4 цифры)
`getMonth()` – получить месяц, от 0 до 11.
`getDate()` – получить день месяца, от 1 до 31, что несколько противоречит названию метода.
`getHours()`, `getMinutes()`, `getSeconds()`, `getMilliseconds()` – получить, соответственно, часы, минуты, секунды или миллисекунды.
`getDay()` – получить день недели от 0 (воскресенье) до 6 (суббота). Несмотря на то, что в ряде стран за первый день недели принят понедельник, в JavaScript начало недели приходится на воскресенье.

ITEA ONLINE

# Дата и время

Установка компонентов даты

`setFullYear(year, [month], [date])` – установить год
`setMonth(month, [date])` – установить месяц
`setDate(date)` – установить день месяца
`setHours(hour, [min], [sec], [ms])` – установить часы
`setMinutes(min, [sec], [ms])` – уставиновить минуты
`setSeconds(sec, [ms])` – установить секунды
`setMilliseconds(ms)` – установить миллисекунды

```
let today = new Date();
today.setHours(0);
console.log(today); // выводится сегодняшняя дата, но значение часа будет 0
today.setHours(0, 0, 0, 0);
console.log(today); // всё ещё выводится сегодняшняя дата, но время будет ровно 00:00:00.
```

ITEA ONLINE

# Дата и время

Автоисправление даты – это очень полезная особенность объектов Date. Можно устанавливать компоненты даты вне обычного диапазона значений, а объект сам себя исправит.

Неправильные компоненты даты автоматически распределяются по остальным. Предположим, нам требуется увеличить дату «28 февраля 2016» на два дня. В зависимости от того, високосный это год или нет, результатом будет «2 марта» или «1 марта». Нам об этом думать не нужно. Просто прибавляем два дня.

```
let date = new Date(2016, 1, 28);
date.setDate(date.getDate() + 2);

console.log(date); // 1 Mar 2016
```

ITEA ONLINE

# Q&A

ITEA

ONLINE