



```
→ Язык JavaScript → Продвинутая работа с функциями
```

Глобальный объект

Глобальный объект предоставляет переменные и функции, доступные в любом месте программы. По умолчанию это те, что встроены в язык или среду исполнения.

В браузере он называется window, в Node.js — global, в другой среде исполнения может называться иначе.

Hegabho globalThis был добавлен в язык как стандартизированное имя для глобального объекта, которое должно поддерживаться в любом окружении. Он поддерживается во всех основных браузерах.

Далее мы будем использовать window, полагая, что наша среда – браузер. Если скрипт может выполняться и в другом окружении, лучше будет globalThis.

Ко всем свойствам глобального объекта можно обращаться напрямую:

```
1 alert("Привет");
2 // это то же самое, что и
3 window.alert("Привет");
```

В браузере глобальные функции и переменные, объявленные с помощью var (не let/const!), становятся свойствами глобального объекта:

```
1 var gVar = 5;
2
3 alert(window.gVar); // 5 (становится свойством глобального объекта)
```

То же самое касается функций, объявленных с помощью синтаксиса Function Declaration (выражения с ключевым словом function в основном потоке кода, не Function Expression)

Пожалуйста, не полагайтесь на это. Такое поведение поддерживается для совместимости. В современных проектах, использующих JavaScript-модули, такого не происходит.

Если бы мы объявили переменную при помощи let, то такого бы не произошло:

```
1 let gLet = 5;
2
3 alert(window.gLet); // undefined (не становится свойством глобального объекта)
```

Если свойство настолько важное, что вы хотите сделать его доступным для всей программы, запишите его в глобальный объект напрямую:

```
1 // сделать информацию о текущем пользователе глобальной, для предоставления дс
2 window.currentUser = {
3    name: "John"
4 };
5
6 // где угодно в коде
7 alert(currentUser.name); // John
8
9 // или, если у нас есть локальная переменная с именем "currentUser",
10 // получим её из window явно (безопасно!)
11 alert(window.currentUser.name); // John
```

При этом обычно не рекомендуется использовать глобальные переменные. Следует применять их как можно реже. Дизайн кода, при котором функция получает входные параметры и выдаёт определённый результат, чище, надёжнее и удобнее для тестирования, чем когда используются внешние, а тем более глобальные переменные.

Использование для полифилов

Глобальный объект можно использовать, чтобы проверить поддержку современных возможностей языка.

Например, проверить наличие встроенного объекта **Promise** (такая поддержка отсутствует в очень старых браузерах):

```
1 if (!window.Promise) {
2 alert("Ваш браузер очень старый!");
3 }
```

Если нет (скажем, используется старый браузер), мы можем создать полифил: добавить функции, которые не поддерживаются окружением, но существуют в современном стандарте.

```
1 if (!window.Promise) {
2  window.Promise = ... // собственная реализация современной возможности языка
3 }
```

Итого

- Глобальный объект хранит переменные, которые должны быть доступны в любом месте программы.
 - Это включает в себя как встроенные объекты, например, Array, так и характерные для окружения свойства, например, window.innerHeight высота окна браузера.
- Глобальный объект имеет универсальное имя globalThis.
 - ...Но чаще на него ссылаются по-старому, используя имя, характерное для данного окружения, такое как window (браузер) и global (Node.js).
- Следует хранить значения в глобальном объекте, только если они действительно глобальны для нашего проекта. И стараться свести их количество к минимуму.
- В браузерах, если только мы не используем модули, глобальные функции и переменные, объявленные с помощью var, становятся свойствами глобального объекта.

• Для того, чтобы код был проще и в будущем его легче было поддерживать, следует обращаться к свойствам глобального объекта напрямую, как window.x.

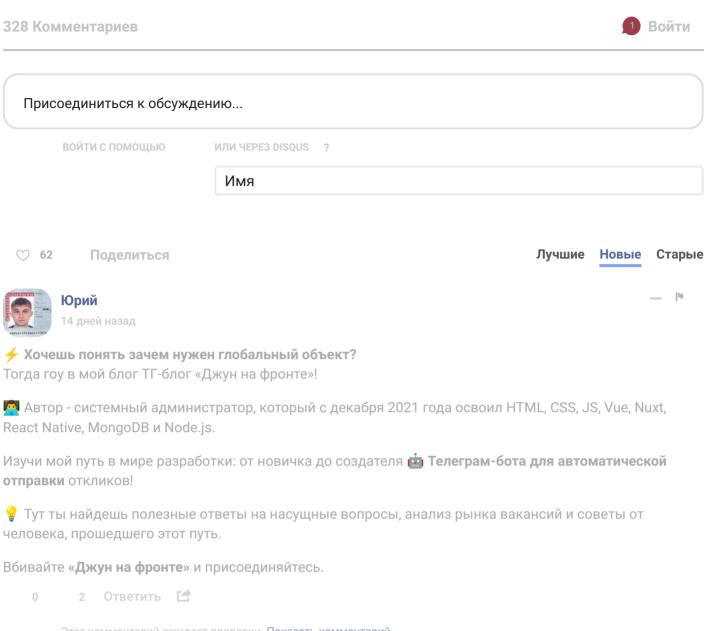


Проводим курсы по JavaScript и фреймворкам.

X

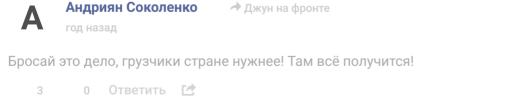
Комментарии

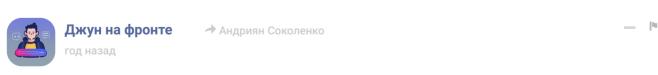
- Если вам кажется, что в статье что-то не так вместо комментария напишите на GitHub.
- Для одной строки кода используйте тег <code> , для нескольких строк кода тег pre> , если больше 10 строк ссылку на песочницу (plnkr, JSBin, codepen...)
- Если что-то непонятно в статье пишите, что именно и с какого места.





Этот комментарий ожидает проверки. Показать комментарий.





По себе людей не судят

12 Ответить 🗠



удались, пожалуйста, ты только негатив тут создаешь Программирование это не твое



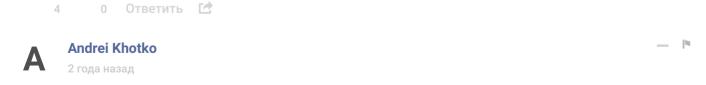
Тут сказано: "То же самое касается функций, объявленных с помощью синтаксиса Function Declaration", то есть Function Delcaration на уровне global lexical environment попадают также в глобальный объект. Но этот код вызывает ошибку, и не видит функцию в объекте (видит undefined).

Ваша функция ничего не возвращает, поэтому если записать в console.log() она вернёт undefined. вместо console.log(a + 100) напишите: return a + 100 И функция заработает



Нет там ошибки. Первый раз дает 99 из console.log(a + 100);, потом твоя функция не возвращает значения из console.log(globalThis.globalFunc(-1)); и дальше собственно сама функция из console.log(globalThis.globalFunc);.

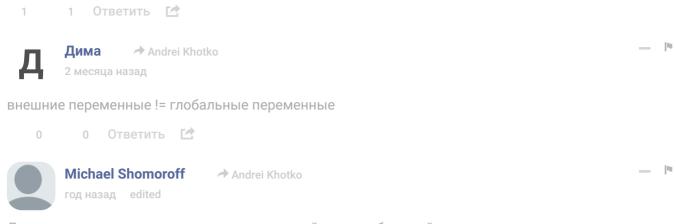
Бессмысленно много выдачи в консоль. Если в твоей функции прописать, чтобы она возвращала значение return a + 100;, то будет нормально работать



Из текущей статьи:

Дизайн кода, при котором функция получает входные параметры и выдаёт определённый результат, чище, надёжнее и удобнее для тестирования, чем когда используются внешние, а тем более глобальные переменные.

тогда такой вопрос: в теме замыкания только так, налево и направо, использовались внешние переменные в замыкающих функциях. Так стоит ли писать такие функции, использующие переменные из внешнего лексического окружения, или стоит писать "чистые функции", принимающие все необходимые значения извне?



Да простят меня джаваскриптоносцы за такой питонообразный код.

В замыкания можно использовать для написания функций обёрток для других функций, которые выполняются вместо них, делают что-то а потом возвращают оборачиваемую функцию.

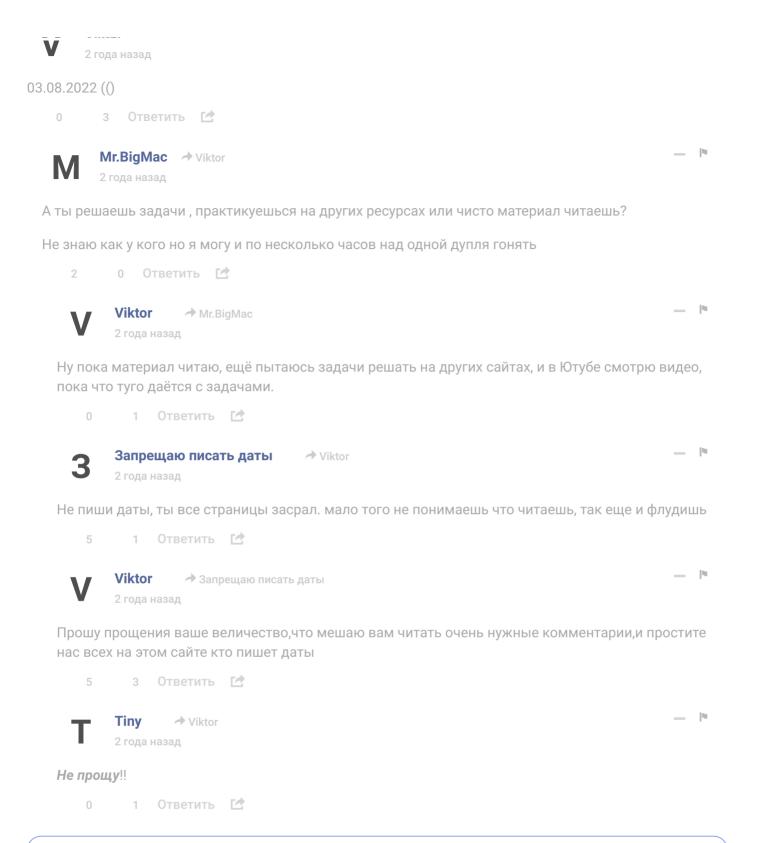


так там суть во вложенности функций и основная функция часто ничего просто не делает так что не думаю что к этому относится данный коммент



Ну, в главе "Замыкания" объясняется внутренняя структура и как работают замыкания. Автор дает понять, что эта возможность используется очень часто, но никаких советов по поводу стоит или не стоит, там нет.

2 1 Ответить 🗠



Загрузить ещё комментарии

Подписаться

О защите персональных данных

Не продавайте мои данные

