Разработчики JavaScript пользуются большим спросом в мире IT. Если это та сфера, в которой у вас достаточно знаний, у вас есть много возможностей изменить место своей работы и повысить заработную плату. Но прежде чем вы будете приняты на работу, вам необходимо будет пройти собеседование. В этой статье мы покажем вам 5 типичных вопросов, которые вы должны пройти при приёме на работу и показать свои умения в JavaScript.

**Вопрос 1: Возможности**

Рассмотрим следующий код:

|  |  |
| --- | --- |
| 1 | (function() { |
| 2 | var a = b = 5; | |

|  |  |  |
| --- | --- | --- |
| 3 | })(); | |
| 4 |  |

|  |  |
| --- | --- |
| 5 | console.log(b); |

Что будет написано на консоли?

**Ответ**

Код выше напишет 5.

Хитрость этого вопроса заключается в том, что в IIFE есть два задания, но переменная a объявляется ключевым словом var. Это означает, что a является переменной функции. b же присвоена глобальной области.

Другой хитростью этого вопроса является то, что он использует строгий режим ('use strict';) в функции. Если был включен строгий режим, код покажет ошибку “Uncaught ReferenceError: b не определён”. Помните, что строгий режим требует, чтобы вы ссылались на глобальные области. Таким образом, вы должны написать:

|  |  |
| --- | --- |
| 1 | (function() { |
| 2 | 'use strict'; | |

|  |  |  |
| --- | --- | --- |
| 3 | var a = window.b = 5; | |
| 4 | })(); |

|  |  |
| --- | --- |
| 5 |  |
| 6 | console.log(b); | |

**Вопрос 2: Создание “родных” методов**

Определите повторяющуюся функцию в строке объекта. Функция подразумевает целое число, определяющее, сколько раз строка должна повториться. Функция повторяет указанную строку указанное количество раз. Например:

|  |  |
| --- | --- |
| 1 | console.log('hello'.repeatify(3)); |

Должно получиться "hellohellohello".

**Ответ**

Возможная реализация показана ниже:

|  |  |  |
| --- | --- | --- |
| 1 | String.prototype.repeatify = String.prototype.repeatify || function(times) { | |
| 2 | var str = ''; |

|  |  |
| --- | --- |
| 3 |  |
| 4 | for (var i = 0; i < times; i++) { | |

|  |  |  |
| --- | --- | --- |
| 5 | str += this; | |
| 6 | } |

|  |  |
| --- | --- |
| 7 |  |
| 8 | return str; | |

|  |  |
| --- | --- |
| 9 | }; |

Вопрос проверяет знания разработчиков о наследовании в JavaScript и свойстве прототипа. Так же проверяется и способность разработчика расширять функциональность собственных типов данных (хотя это не обязательно).

Ещё одним важным моментом является демонстрация знаний о том, как не отменить уже определенные функции. Это делается путём тестирования, подтверждающего, что функция не существует самостоятельно до ее определения.

|  |  |  |
| --- | --- | --- |
| 1 | String.prototype.repeatify = String.prototype.repeatify || function(times) | |
| 2 | {/\* code here \*/}; |

Этот метод особенно полезен, когда вас попросили заклинить функцию JavaScript.

**Вопрос 3: Подъём**

Каков результат выполнения и почему?

|  |  |
| --- | --- |
| 01 | function test() { |
| 02 | console.log(a); | |

|  |  |  |
| --- | --- | --- |
| 03 | console.log(foo()); | |
| 04 |  |

|  |  |
| --- | --- |
| 05 | var a = 1; |
| 06 | function foo() { | |

|  |  |  |
| --- | --- | --- |
| 07 | return 2; | |
| 08 | } |

|  |  |
| --- | --- |
| 09 | } |
| 10 |  | |

|  |  |
| --- | --- |
| 11 | test(); |

**Ответ**

Результаты кода – underfined и 2.

Причина в том, что обе переменные функции подняты (помещены в верхней части функции), но переменные не удерживают присвоены значения. В то время, когда переменная a печатается, она существует в функции (это декларируется), но она все ещё undefined. Другими словами, код выше эквивалентен следующему:

|  |  |  |
| --- | --- | --- |
| 01 | function test() { | |
| 02 | var a; |

|  |  |  |
| --- | --- | --- |
| 03 | function foo() { | |
| 04 | return 2; |

|  |  |  |
| --- | --- | --- |
| 05 | } | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | console.log(a); |
| 08 | console.log(foo()); | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | a = 1; | |

|  |  |  |
| --- | --- | --- |
| 11 | } | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | test(); |

**Вопрос 4: Как это работает в JavaScript**

Каков результат следующего кода? Обоснуйте свой ответ.

|  |  |  |
| --- | --- | --- |
| 01 | var fullname = 'John Doe'; | |
| 02 | var obj = { |

|  |  |  |
| --- | --- | --- |
| 03 | fullname: 'Colin Ihrig', | |
| 04 | prop: { |

|  |  |  |
| --- | --- | --- |
| 05 | fullname: 'Aurelio De Rosa', | |
| 06 | getFullname: function() { |

|  |  |  |
| --- | --- | --- |
| 07 | return this.fullname; | |
| 08 | } |

|  |  |  |
| --- | --- | --- |
| 09 | } | |
| 10 | }; |

|  |  |
| --- | --- |
| 11 |  |
| 12 | console.log(obj.prop.getFullname()); | |

|  |  |
| --- | --- |
| 13 |  |
| 14 | var test = obj.prop.getFullname; | |

|  |  |
| --- | --- |
| 15 |  |
| 16 | console.log(test()); | |

**Ответ**

Код выдаст Aurelio De Rosa и John Doe. Причина в том, что контекст функции, вызываемый ключевым this словом, в JavaScript зависит от того, как именно вызывается функция, а не от того, как она определена.  
Вызов первых console.log ( ), getFullname ( ) вызывается функцией объекта obj.prop. Таким образом, контекст относится к последнему и возвращает “fullname” как свойство объекта. В противном случае, когда getFullname ( ) присваивается переменной, контекст относится к глобальному объекту. Это происходит потому, что тест устанавливается как свойство глобального объекта (window). По этой причине функция возвращает значение свойства fullnameиз window, которая в данном случае является кодом, устанавливаемым в первой строке фрагмента.

**Вопрос 5: call ( ) и apply ( )**

Исправьте предыдущий вопрос так, чтобы последний console.log ( ) стал Aurelio De Rosa.

**Ответ**

Вопрос может быть изменен, повлияв на контекст функции, используя функции call ( ) или apply ( ). Если вы не знаете их и их отличия, вам стоит прочесть статью. В коде ниже я буду использовать call ( ), но в этом случае применяется apply ( ), результат будет тот же:

|  |  |
| --- | --- |
| 1 | console.log(test.call(obj.prop)); |