Лабораторная работа 7.

Замыкания.

Платформа:

<https://tio.run/>

Цели:

* Знакомство с замыканиями и их областью применения.
* Безопасность кода

Синопсис, тезисно:

* Эволюция возвращаемых функцией значений
  + Один параметр фиксированного типа
  + Массив или ассоциативный массив (хэш-массив) на входе функции
  + Функция

Описание:

Замыкание (Closure) – механизм, реализуемый с помощью вспомогательной функции, которая при вызове возвращает новую функцию, использующую набор данных, недоступных для изменения из кода за её пределами.   
Такие данные (например, переменные, массивы, функции и т.д.) можно рассматривать как «замкнутые», либо «изолированные».

Пример:

// функция, генерирующая функции, которые при каждом вызове будут возвращать очередное, возрастающее число.

function generateNumbers(fromN) {

let n = fromN

return function() {

return n++

}

}

counter1 = generateNumbers(0)

counter2 = generateNumbers(10)

console.log(counter1()) // вызываем первый генератор, он считает от нуля и "помнит" своё состояниее, т.к. использует "замкнутую" переменную n

console.log(counter1()) // теперь ее значение = 1

console.log(counter1()) // теперь ее значение = 2

console.log(counter2()) // вторая функция помнит собственное значение n, тоже "замкнутое" в ней - это 10

console.log(counter2()) // 11

console.log(counter1()) // при этом функция counter1 по-прежнему возвращает собственные значения, сейчас это 3

Характерно, что если в функции generateNumbers убрать слово let, сделав n по отношению к ней не ЛОКАЛЬНОЙ, а захватываемой из внешнего контекста переменной, то замыкания у нас не произойдёт.

Замыкания могут использоваться как счетчики, в логику которых нельзя вмешаться, как генераторы криптопоследовательностей, относительно безопасно хранящие в себе секретный ключ, а также использоваться везде, где нам необходимо снизить вероятность «сломать» работу функций, случайно объявив во внешнем контексте имя переменной, используемой в нашей функции.

Задание:

1. Написать функцию, создающую генераторы последовательностей чисел Фиббоначи с произвольными стартовыми числами (обычно это две единицы) – которую можно было бы использовать как  
   fibA = createFib(1,1)  
   fibA() // = 2  
   fibA() // = 3  
   fibB = createFib(0,2)  
   fibB() // = 2  
   fibB() // = 4  
   fibB() // = 6
2. Написать функцию, которая создаёт функции-генераторы случайных целых чисел в диапазоне от 0 до 10, таким образом, что ни одно сгенерированное отдельным экземпляром генератора число не повторяется. После исчерпания такими функциями всех доступных чисел функция должна возвращать false  
   rnd = createRandom()  
   rnd() // = 1  
   rnd() // = 7  
   …  
   rnd() // = 8 – это был 11-й вызов  
   rnd() // = false

Варианты:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | Java | 4 | Haskell | 7 | Python |
| 2 | Go | 5 | Lua | 8 | C(любой) |
| 3 | Ruby | 6 | JavaScript | 9 | PHP |