Języki programowania i GUI

Lista 2 - 2021

0. Ze strony `https://javascript.info/` przeczytaj rozdziały:
   - `https://javascript.info/symbol`
   - `https://javascript.info/object-toprimitive`
   - `https://javascript.info/iterable`
   - `https://javascript.info/destructuring-assignment`

   a potem zrozum i wypróbuj poniższy kod:

```
function zakres(a,b){this.a=a;this.b=b}
zakres.prototype[Symbol.iterator]=
      function*(){
         for(let i=this.a;i<=this.b;i++) yield i;
      }
zakres.prototype[Symbol.toPrimitive]=
      function(hint){return hint=="number"?
                                  (this.a+this.b)*(this.b-this.a+1)/2:
                                  this.a+"..."+this.b;}
z=new zakres(10,15); console.log(z);
for(let x of z)
    console.log(x);
console.log("suma("+z+")="+ +z)
console.log(Array.from(z))
```

   Następnie zapisz ten kod w postaci definicji klasy `zakres` i wypróbuj ponownie.

1. Zapisz funkcję gwiazdkową `Fibonacci()` zwracająca iterator na wszystkie liczby Fibonacciego zwracane jako typ `BigInt`, czyli z dowolnie dużą ilością cyfr. Dla przetestowania wywołaj 200 razy metodę `next()` wynikowego operatora, i wypisuj pole `value`. Napisz drugą wersję programu, która za pomocą `setInterval` lub `setTimeout` będzie uruchamiała `next()` co pół sekundy.

2. Napisz funkcję `Fibo()` – konstruktor oraz `Fibo.prototype.next=function(){...}` tak, aby obiekty `let z1=Fibonacci()` z zadania 1, oraz `let z2=new Fibo()` z tego zadania zachowywały się identycznie w trakcie testów z zadania 1.

3. Zapisz `function* fragment(iter,skip,limit=1)`, która zwróci iterator, który z argumentu `iter`, który też jest iteratorem, pobiera kolejne wartości za pomocą `for( of )`, ale pomija ilość = `skip` początkowych wartości i zwraca (przez `yield`) ilość = `limit` następnych, a potem kończy działanie. Zastosuj ją tak:
   `for(let x of fragment(Fibonacci(),100,3)) console.log(x)`

4. Napisz `Array.prototype.wspak=function*(){...}`, która zwraca iterator idący po elementach tablicy od końca. Użyj pętli po indeksach tablicy oraz `yield`. Nie możesz użyć `Array.reverse()`. Zastosowanie: `for(let x of [2,3,4,5].wspak()) ....`

5. Napisz funkcję `function BST(key,left,right)`, która zwraca węzeł drzewa BST o poddrzewach `left` i `right` i kluczu `key`. Zainicjuj `BST.prototype[Symbol.iterator]` taką funkcją gwiazdkową, by pętla `for( of )` dla drzew `BST` pokazywała ich klucze w porządku `in order`, czyli rosnącym. Aby zademonstrować działanie utwórz jednym poleceniem drzewo o co najmniej 7 kluczach na nie więcej niż 4 poziomach.

6. (2pkt) Napisz funkcję `arytmetyczny(dane)`, która zwraca obiekt, będący implementacją ciągu arytmetycznego, zawierający metody:

`a(i)`$= a_i$ wartość `i`-tego wyrazu ciągu.
`suma(i)` $= a_1 + a_2 + ... + a_i$ - suma i początkowych wyrazów ciągu.
`get r()` $= r$ - różnica ciągu arytmetycznego
`*[Symbol.iterator]()` - funkcja gwiazdkowa `yield`-ująca kolejne wyraz ciągu arytmetycznego: a_1, a_2 ...

Uwaga: wewnętrznie obiekt może przechowywać dwie liczby (np. a_1 i r lub a_0 i r). Cała trudność zadania polega na takim napisaniu funkcji (lub konstruktora), by była ona „inteligentna“, w stopniu wystarczającym, by prawidłowo obliczać a_0 i r dla najróżniejszych danych, a w szczególności:

- wyraz i różnica: `arytmetyczny({a7:9,r:2})` daje „ciąg” (-3,-1,1,..)
- dwa wyrazy: `arytmetyczny({a3:8,a5:2})` daje ciąg (14,11,8,5,2,...)
- dowolna suma i różnica: `arytmetyczny({suma5:15,r:1})` - daje ciąg (1,2,3,4,5...)
- dowolne dwie sumy: `arytmetyczny({suma3:12,suma6:42})` - daje ciąg (2,4,6,8,..)
- suma i dowolny wyraz : `arytmetyczny({suma5:20,a2:13})`

W przypadku, gdy dane są niewystarczające do wyznaczenia ciągu funkcje `a()` i `sum()` powinny zwracać `null`, a funkcja gwiazdkowa ma nic nie robić.

Można (choć nie jest to konieczne) zrobić tak, by zmienne a_1 i r były w funkcjach `a(), sum() i get r()` wychwycone z funkcji `arytmetyczny` i nie były przechowywane w zwracanym obiekcie.

W pracy nad zadaniem może się przydać wiedza ze strony:
`https://javascript.info/destructuring-assignment#smart-function-parameters`
która pokazuje, jak pisać funkcje z dużą ilością argumentów i jak stosować destrukturyzację dla uproszczenia kodu.

7. (2pkt) Podobną funkcjonalność zrealizuj z pomocą klasy z trzema funkcjami `a()`, `suma()`, `r()` takimi, że:
`a(i)` - zwraca a_i
`a(i,x)` - ustawia $a_i = x$
`suma(i)` - zwraca S_i
`suma(i,x)` - ustawia $S_i = x$
`r()` - zwraca r
`r(x)` - ustawia $r = x$

8. * (3pkt) Zapoznaj się z tym jak działa `Proxy` oraz `Reflect`. Zaimplementuj takie `Proxy`, by dane o ciągu arytmetycznym można było wprowadzać i pobierać za pomocą fejkowych właściwości, o nazwach postaci: `"a${i}"` , `"suma${i}"`, oraz `"r"`, gdzie i jest liczbą naturalną:

```
Ary = new Proxy (......);
Ary.a5 = 7;
Ary.a8 = 13   //   (-1,1,3,5,7,9,11,13)
console.log(Ary.a3 + Ary.a7); // 3+11=14
console.log(Ary.suma3); // 3
```