Работа с по-сложни цикли

Цикли със стъпка, While, Do-While

СофтУни

трейнърски екип

Софтуерен университет

http://softuni.bg

Have a Question?

SoftUni Foundation

sli.do

#pb-feb

Съдържание

- По-сложни конструкции за цикъл:
  - Цикъл със стъпка
  - Цикъл с намаляваща стъпка
  - Цикъл - **`while`**
  - Цикъл - **`do-while`**
  - Безкраен цикъл
    - Оператор **`break`**
    - Оператор **`continue`**

```
for (i = 1; i < 10; i += 2)
for (i = 100; i > 0; i -= 3)
```

Цикли със стъпка

Работа с по-сложни **for**-цикли

Числата от 1 до N през 3 - условие

- Напишете програма, която:
  - Прочита цяло число **n**
  - Отпечатва числата от **1** до **n** със стъпка **3**
- Примерен вход и изход:

10	→	1, 4, 7, 10
15	→	1, 4, 7, 10, 13

Числата от 1 до N през 3 – решение

```
function loopByStep3([arg1]) {
  let n = Number(arg1);
  for (let i = 1; i <= n; i+=3 ) {
    console.log(i);
  }
}
loopByStep3([15]);
```

Задаване на **стъпка**

DEBUG CONSOLE

```
Debugging with inspector
node advancedLoops.js
1
4
7
10
13
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#0

Числата от N до 1 в обратен ред - условие

- Напишете програма, която:
  - Прочита цяло число **n**
  - Отпечатва числата от **n** до **1** в обратен ред (стъпка **-1**)
- Примерен вход и изход:

100	→	100, 99, 98, …, 3, 2, 1
3	→	3, 2, 1

Числата от N до 1 в обратен ред – решение

```
function numbersNto1([arg1]) {
  let n = Number(arg1);
  for (let i = n; i >= 1; i-=1 ) {
    console.log(i);
  }
}
```

Отрицателна стъпка: -1

Обърнато условие:
`i >= 1`

BACKWARDS

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#1

Числата от 1 до 2^n с for-цикъл – условие

- Напишете програма, която:
  - Прочита цяло число **n**
  - Отпечатва числата от **1** до **2^n**
- Примерен вход и изход:

10	→	1, 2, 4, 8, 16, 32, …, 1024
5	→	1, 2, 4, 8, 16, 32

Числата от 1 до 2^n с for-цикъл – решение

```
function powersOfTwo([arg1]) {
  let n = Number(arg1);
  let num = 1;
  for (let i = 0; i <= n; i++) {
    console.log(num);
    num = num * 2;
  }
}
powersOfTwo([5]);
```

```
DEBUG CONSOLE
Debugging with inspector
node advancedLoops.js
1
2
4
8
16
32
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#2

Четни степени на 2 - условие

- Напишете програма, която:
  - Прочита цяло число **n**
  - Отпечатва четните степени на **2** до $\mathbf{2^n}$: 2^0, 2^2, 2^4, 2^8, …, $\mathbf{2^n}$
- Примерен вход и изход:

10	➡	1, 4, 16, …, 1024
3	➡	1, 4

Четни степени на 2 – решение

```
function evenPowersOfTwo([arg1]) {
  let n = Number(arg1);
  let num = 1;
  for (let i = 0; i <= n; i+=2 ) {
    console.log(num);
    num = num * 2 * 2;
  }
}
evenPowersOfTwo([7]);
```

Ползваме стъпка +2

DEBUG CONSOLE

```
Debugging with inspector
node advancedLoops.js
1
4
16
64
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#3

```
while (condition)
{ // code }
```

While цикъл

Повторение докато е в сила дадено условие

While цикъл

- Тялото на цикъла се изпълнява докато е вярно дадено условие

Условие (true/false)

```
while (...) {
    //code
}
```

Код за изпълнение (повторение)

Редица числа 2k+1 - условие

- Напишете програма, която:
  - Прочита цяло число **n**
  - Отпечатва всички числа ≤ **n** от редицата: 1, 3, 7, 15, 31, …
  - Всяко следващо число е равно на предишното * 2 + 1

```
1, (1*2)+1 = 3, (3*2)+1 = 7, (7*2)+1 = 15 …
```

  - Примерен вход и изход:

```
17 ➡ 1, 3, 7, 15
```

```
33 ➡ 1, 3, 7, 15, 31
```

Редица числа 2k+1 – решение

```
function sequence([arg1]) {
  let n = Number(arg1);
  let num = 1;
  while (num <= n) {
    console.log(num);
    num = 2 * num + 1;
  }
}
sequence([17]);
```

Повторение докато е в сила условието num ≤ n

```
DEBUG CONSOLE
Debugging with inspector
node advancedLoops.js
1
3
7
15
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#4

Число в диапазона [1…100] - условие

- Напишете програма, която:
  - Прочита цяло число
  - Проверява дали е в диапазона [1…100]
  - При:
    - Намиране на число в диапазона, прекратява изпълнение
    - Невалидно число прочита ново

Число в диапазона [1…100] – решение

```
function numberInRange(args) {
  let i = 0;
  let num = Number(args[i]);
  while (num < 1 || num > 100) {
    i++;
    console.log("Invalid number!");
    num = Number(args[i]);
  }
  console.log(`The number is: ${num}`);
}
numberInRange([-22, 401, 93]);
```

```
DEBUG CONSOLE
Debugging with inspector
node advancedLoops.js
Invalid number!
Invalid number!
The number is: 93
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#5

```
while (a <= b)
{ a += 10; }
```

Цикли със стъпка и while цикъл

Работа на живо в клас (лаб)

Най-голям общ делител (НОД)

Алгоритъм на Евклид

Най-голям общ делител (НОД)

- Най-голям общ делител (НОД) на две естествени числа **a** и **b** е най-голямото число, което дели едновременно **a** и **b** без остатък
  - НОД(24, 16) = 8
  - НОД(67, 18) = 1
  - НОД(12, 24) = 12
  - НОД(15, 9) = 3
  - НОД(10, 10) = 10
  - НОД(100, 88) = 4

Алгоритъм на Евклид за НОД - условие

- Напишете програма, която:
  - Прочита 2 цели числа **a** и **b**
  - Намира най-големия им общ делител - **НОД**(**a**, **b**)
- Насоки:
  - Докато не се достигне остатък 0:
    - Дели се по-голямото число на по-малкото
    - Взема се остатъка от делението

Псевдо код:

```
while b ≠ 0
  oldB = b;
  b = a % b;
  a = oldB;
print a;
```

Алгоритъм на Евклид за НОД - решение

```
function gcd([arg1, arg2]) {
  let a = Number(arg1);
  let b = Number(arg2);
  while (b != 0) {
    let oldB = b;
    b = a % b;
    a = oldB;
  }
  console.log("GCD = " + a);
}
```

```
a = 600, b = 136
600 / 136 = 4 (ост. 56)
136 / 56 = 2  (ост. 24)
56 / 24 = 2   (ост. 8)
24 / 8 = 3    (ост. 0)
НОД = 8
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#6

```
do { // code }
while (condition)
```

Do-While цикъл

Повторение докато е изпълнено условието

Do-while цикъл

- Тялото на цикъла се изпълнява докато е вярно дадено условие
- Изпълнява се минимум един път

```
do {
    //code
} while (...);
```

Код за изпълнение (повторение)

Условие (true/false)

Изчисляване на факториел - условие

- Напишете програма, която:
  - Прочита естествено число **n**
  - Изчислява факториел от n (**n!**)
- Примерен вход и изход:
  - **5!** = **1** * **2** * **3** * **4** * **5** = **120**

5	→	120
6	→	720

Изчисляване на факториел – решение

```
function factorial([arg1]) {
  let n = Number(arg1);
  let fact = 1;
  do {
    fact = fact * n;
    n--;
  } while (n > 1);
  console.log(fact);
}
factorial([10]);
```

```
DEBUG CONSOLE
Debugging with inspector
node advancedLoops.js
3628800
```

n!

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#7

Сумиране на цифрите на число - условие

- Напишете програма, която:
  - Прочита цяло положително число **n**
  - Сумира цифрите на **n**
- Примерен вход и изход:
  - **n** = 5634: 5 + 6 + 3 + 4 = 18

5634	→	18	1234	→	10

Сумиране на цифрите на число

```
function sumDigits([arg1]) {
  let n = Number(arg1);
  let sum = 0;
  do {
    sum = sum + (n % 10);
    n = Math.floor(n / 10);
  } while (n > 0);
  console.log("Sum of digits: " + sum);
}
```

n % 10 връща последната цифра на числото **n**

n / 10 премахва последната цифра на **n**

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#8

Безкрайни цикли и оператори break и continue

Безкраен цикъл

- Безкраен цикъл имаме когато:
  - Нямаме условие, което да прекрати цикъла
  - Нямаме команда, която да прекрати цикъла

```
while(true) {
   console.log("Infinite loop");
}
```

```
for (;;) {
 console.log("Infinite loop");
}
```

Условия за прекратяване на цикъл

Условие за прекратяване на цикъл

```
while(...) {
   console.log("Infinite loop");
}
```

Условие за прекратяване на цикъл

```
for (;...;) {
 console.log("Infinite loop");
}
```

Команда за прекратяване на цикъл

- Оператор break – прекъсване на цикъла

```
while(true) {
 console.log("Infinite loop");
   if (...) {
      break;
   }
}
```

Условие за прекратяване на цикъл

Команда за излизане от цикъл

Прости числа - условие

- Напишете програма, която:
  - Прочита цяло число **n**
  - Проверява да ли **n** е просто число
- Насоки:
  - Едно число **n** е просто, ако се дели единствено на **1** и **n** и е по-голямо от 1
  - Прости числа: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, …
  - Непрости (композитни) числа: 10 = 2 * 5, 21 = 3 * 7, 143 = 13 * 11

Прости числа – решение

```
function isPrime([arg1]) {
  let n = Number(arg1);
  let prime = true;
  for (let i = 2; i <= Math.sqrt(n); i++)
    if (n % i == 0) {
      prime = false;
      break;
    }
  if (prime && n > 2) console.log("Prime");
  else console.log("Not prime");
}
isPrime([2]);
```

break излиза от цикъла

```
DEBUG CONSOLE
Debugging with inspector
node advancedLoops.js
Not prime
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#9

Четно число - условие

- Напишете програма, която:
  - Прочита число **n**
  - Проверява дали **n** е четно
  - При невалидно число се връща към повторно въвеждане

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#10

Четно число – решение

```
function enterEvenNumber(args) {
  let i = 0;
  let num = 0;
  while (true) {
    num = Number(args[i]);
    if (num % 2 == 0)
      break;        // even number -> exit from the loop
    console.log("The number is not even.");
    i++;
  }
  console.log(`Even number entered: ${num}`);
}
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#10

SoftUni Foundation

Задачи с цикли

Работа на живо в клас (лаб)

Числа на Фибоначи - условие

- Напишете програма, която:
  - Прочита число **n**
  - Пресмята **n-тото** число на Фибоначи
- Числата на Фибоначи са следните: 1, 1, 2, 3, 5, 8, 13, 21, 34, …
- Примерен вход и изход:

15	→	987	3	→	2

Числа на Фибоначи

```
function fibonacci([arg1]) {
  let n = Number(arg1);
  let f0 = 1;
  let f1 = 1;
  for (let i = 0; i < n-1; i++) {
    let fNext = f0 + f1;
    f0 = f1;
    f1 = fNext;
  }
  console.log(f1);
}
fibonacci([10]);
```

```
DEBUG CONSOLE
Debugging with inspector
node advancedLoops.js
89
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#11

Пирамида от числа – условие

- Да се отпечатат числата 1…n в пирамида като в примерите:

```
n = 7
```

```
1
2 3
4 5 6
7
```

```
n = 10
```

```
1
2 3
4 5 6
7 8 9 10
```

```
n = 12
```

```
1
2 3
4 5 6
7 8 9 10
11 12
```

```
n = 15
```

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#12

Пирамида от числа – решение

```
function numberPyramid([arg1]) {
  let n = Number(arg1);
  let num = 1;
  let result = "";
  for (let row = 1; row <= n; row++) {
    for (let col = 1; col <= row; col++) {
      if (col > 1) result += " ";
      result += num;
      num++;
      if (num > n) break;
    }
    console.log(result);
    result = "";
    if (num > n) break;
  }
}
numberPyramid([7]);
```

```
DEBUG CONSOLE
Debugging with inspector
node advancedLoops.js
1
2 3
4 5 6
7
```

Таблица с числа – условие

- Да се отпечатат числата 1…n в таблица като в примерите:

n = 2	n = 3	n = 4	n = 5
1 2 2 1	1 2 3 2 3 2 3 2 1	1 2 3 4 2 3 4 3 3 4 3 2 4 3 2 1	1 2 3 4 5 2 3 4 5 4 3 4 5 4 3 4 5 4 3 2 5 4 3 2 1

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#13

Таблица с числа – решение

```
function numberTable([arg1]) {
  let n = Number(arg1);
  let result = "";
  for (let row = 0; row < n; row++) {
    for (let col = 0; col < n; col++) {
      let num = row + col + 1;
      if (num > n) num = 2 * n - num;
      result = result + num + " ";
    }
    console.log(result);
    result = "";
  }
}
numberTable([5]);
```

```
DEBUG CONSOLE
Debugging with inspector
node advancedLoops.js
1 2 3 4 5
2 3 4 5 4
3 4 5 4 3
4 5 4 3 2
5 4 3 2 1
```

Тестване на решението: https://judge.softuni.bg/Contests/Practice/Index/156#13

SoftUni Foundation

По-сложни задачи с цикли

Работа на живо в клас (лаб)

Какво научихме днес?

- Можем да ползваме **for**-цикли със стъпка:

```
for (let i = 1; i <= n; i+=3)
    console.log(i);
```

- Цикли **while** / **do-while** повтарят блок от код докато е в сила дадено условие:

```
let num = 1;
while (num <= n)
    console.log(num++);
```

Какво научихме днес? (2)

- Можем да създаваме безкрайни цикли и когато се наложи да излизаме от тях:

```
for (;;) {
  if (…)
    break;
}
```

```
while (true) {
  if (…)
    break;
}
```

Чертане с цикли

Въпроси?

https://softuni.bg/courses/programming-basics/

Лиценз

- Настоящият курс (слайдове, примери, видео, задачи и др.) се разпространяват под свободен лиценз "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International"

- Благодарности: настоящият материал може да съдържа части от следните източници
  - Книга "Основи на програмирането със C#" от Светлин Наков и колектив с лиценз CC-BY-SA

Безплатни обучения в СофтУни

- Фондация "Софтуерен университет" – softuni.org
- Софтуерен университет – качествено образование, професия и работа за софтуерни инженери
  - softuni.bg
- СофтУни @ Facebook
  - facebook.com/SoftwareUniversity
- СофтУни форуми – forum.softuni.bg