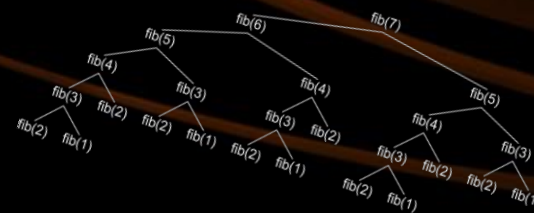


Рекурсия



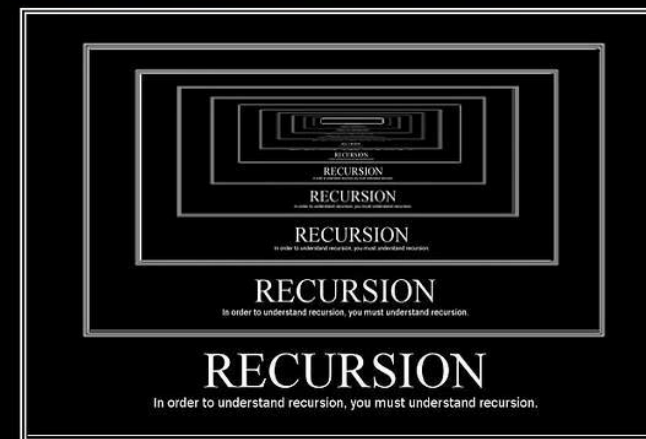
Увод в алгоритмите



Учителски екип

Обучение за ИТ кариера

<https://it-kariera.mon.bg/e-learning/>



RECURSION

In order to understand recursion, you must understand recursion.

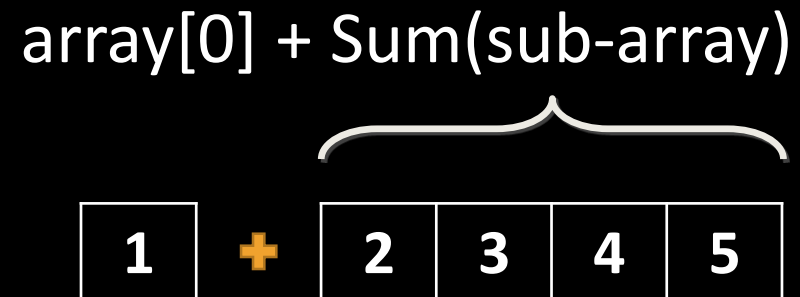
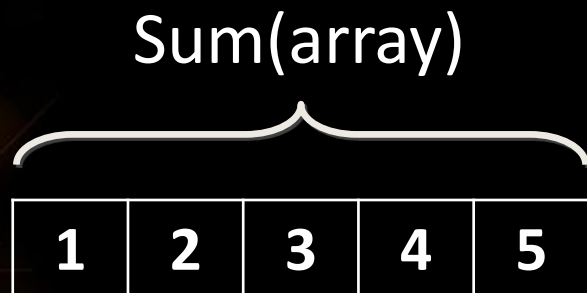
Съдържание

1. Какво е **рекурсия**
2. Рекурсивни решения на познати задачи
3. **Пряка** и **косвена** рекурсия
4. **Предварително** и **последващо** действие
5. Рекурсивно чертане
6. Рекурсия или итерации



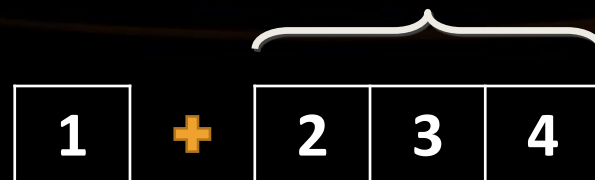
Какво е рекурсия?

- Техника за решаване на задачи чрез разделянето ѝ на **подзадачи от същия тип**
 - Включва **самоизвикване на функция**
 - Функцията трябва да има **основен случай (край, дъно)**
 - **Всяка стъпка** трябва да води **към основния случай**

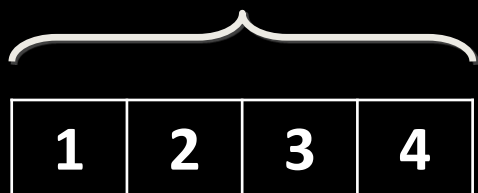


Сбор (сума) на елементите на масив – Пример

$\text{Sum}(n - 1)$



$\text{Sum}(n)$



$\text{Sum}((n - 1) - 1)$

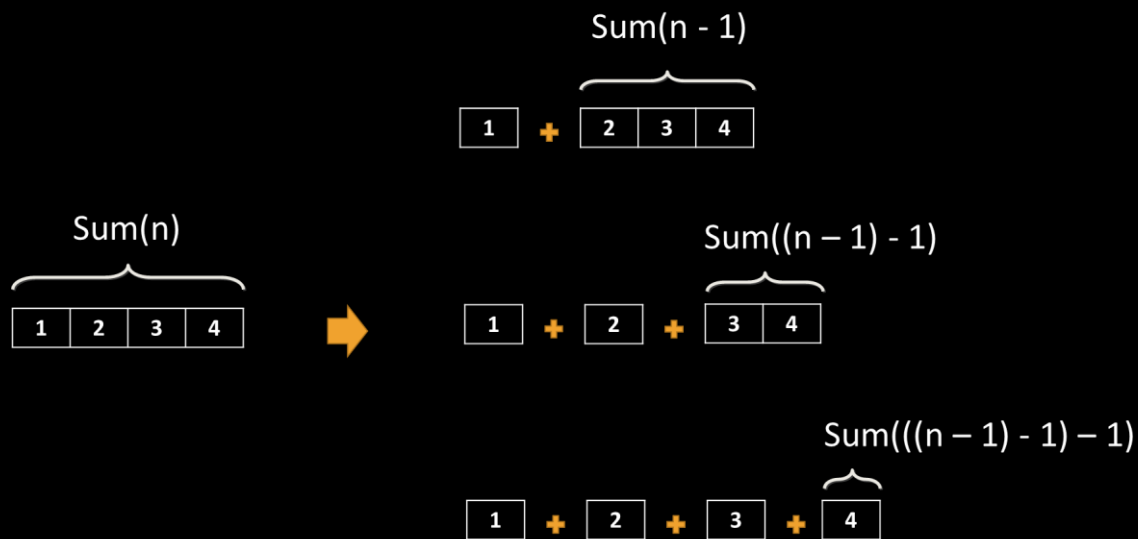


$\text{Sum}(((n - 1) - 1) - 1)$



Задача: Сума на масив

- Създайте **рекурсивен метод**, който:
 - Намира сбора на всички числа, съхранявани в **int[] array**
 - Позволява въвеждане на числа от клавиатурата (конзолата)



Решение: Сума на масив

```
static int Sum(int[] array, int index)
{
    if (index == array.Length - 1)
    {
        return array[index];
    }

    return array[index] + Sum(array, index + 1);
}
```

Основен случай

Рекурсивен факториел – Пример

- Рекурсивна дефиниция на **n!** (n факториел):

$$\begin{aligned} n! &= n * (n-1)! \text{ for } n > 0 \\ 0! &= 1 \end{aligned}$$

- $5! = 5 * 4!$

- $4! = 4 * 3!$

- $3! = 3 * 2!$


- $2! = 2 * 1!$

- $1! = 1 * 0!$

- $0! = 1$

Рекурсивни
извиквания

Основен случай (дъно)



factorial

$$n! = [1*2*3*4* \dots *n]$$

n! is "n factorial"

Задача: Рекурсивен факториел

- Създайте **рекурсивен метод**, който изчислява **$n!$**
 - Въведете n от клавиатурата

5 → 120

10 → 3628800

 **factor!al**

$n! = [1*2*3*4* \dots *n]$

$n!$ is "n factorial"

Решение: Рекурсивен факториел

```
static long Factorial(int num)
```

```
{
```

```
    if (num == 0)
```

```
    {
```

```
        return 1;
```

```
    }
```

```
    return num * Factorial(num - 1);
```

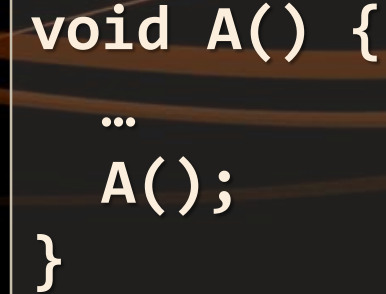
```
}
```

основен случай, край, дъно

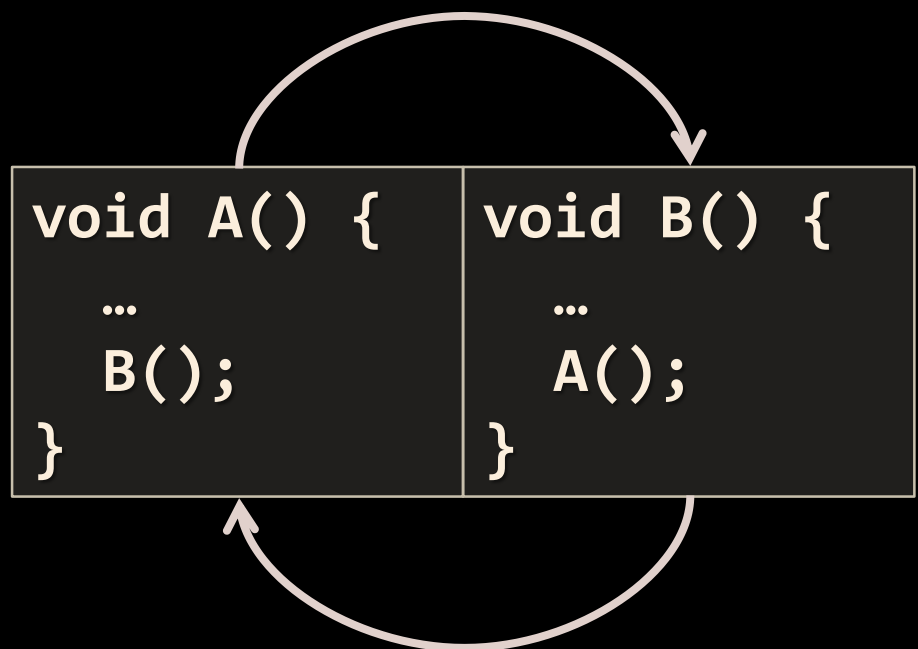
рекурсивно извикване

Пряка и косвена рекурсия

- Пряка рекурсия
 - Метод извиква себе си
- Непряка (косвена) рекурсия
 - Метод А извиква метод В, а Метод В извиква Метод А
 - Или $A \rightarrow B \rightarrow C \rightarrow A$



```
void A() {  
    ...  
    A();  
}
```



<pre>void A() { ... B(); }</pre>	<pre>void B() { ... A(); }</pre>
--	--

Предварително и последващо действие

- Рекурсивните методи имат 3 части:
 - **Предварително действие** (преди извикване на рекурсията)
 - **Рекурсивно извикване** (стъпка навътре)
 - **Последващо действие** (след връщане от рекурсията)

```
static void Recursion()  
{  
    // предварително действие  
    Recursion();  
    // последващо действие  
}
```

Задача: Рекурсивно чертаене

- Създайте **рекурсивен метод**, който чертае следната фигура

5



```
C:\Windows\system32\cmd.exe
*****
****
***
**
*
#
##
###
####
#####
#####
```

Предварителни и последващи действия – пример

```
static void PrintFigure(int n)
{
    if (n == 0) // дъно на рекурсията
        return;

    // предварително действие: отпечатва n звездички
    Console.WriteLine(new string('*', n));

    // рекурсивно извикване: отпечатва фигура с размер n-1
    PrintFigure(n - 1);

    // последващо действие: отпечатва n хештаг-а # (диез)
    Console.WriteLine(new string('#', n));
}
```


Производителност: Рекурсия срещу итерации (цикъл)

- Рекурсивните обръщания са **малко по-бавни** от итерациите
 - Параметрите и върнатите стойности минават през стека на всяка стъпка
 - Предпочита се за линейни изчисления (без разклонени обръщания)

Рекурсивен факториел:

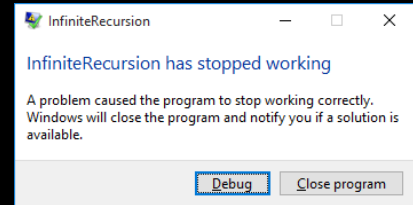
```
static long RecurFact(int n)
{
    if (n == 0)
        return 1;
    else
        return n * Fact(n - 1);
}
```

Итеративен факториел:

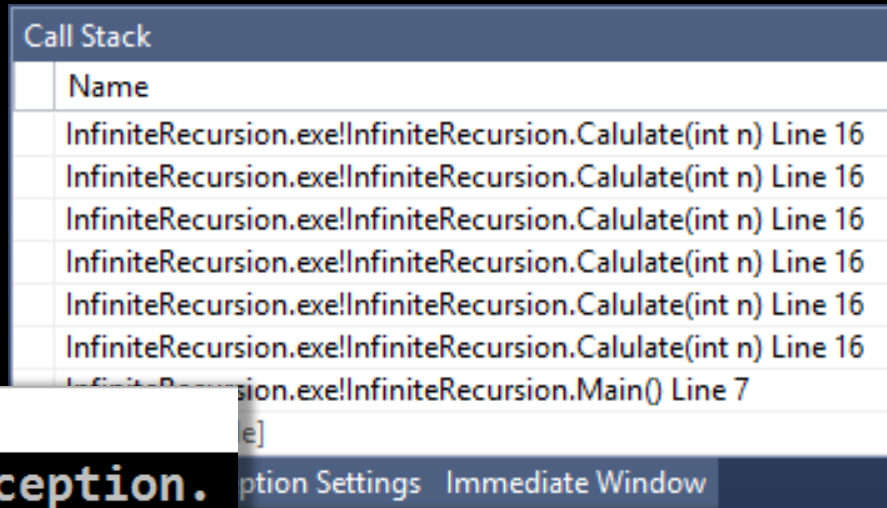
```
static long IterFact(int num)
{
    long result = 1;
    for (int i = 1; i <= num; i++)
        result *= i;
    return result;
}
```

Безкрайна рекурсия

- **Безкрайна рекурсия** == метод, извикващ себе си **безкрайно**
 - Обикновено, безкрайна рекурсия == грешка в програмата
 - Липсва край (дъно) на рекурсията или е грешно зададено
 - В C# / Java / C++ предизвиква грешка "**stack overflow**"



```
static long Calulate(int n)
{
    return Calulate(n + 1);
}
```



C:\Windows\system32\cmd.exe

Process is terminated due to StackOverflowException.

Рекурсията може да бъде и вредна!

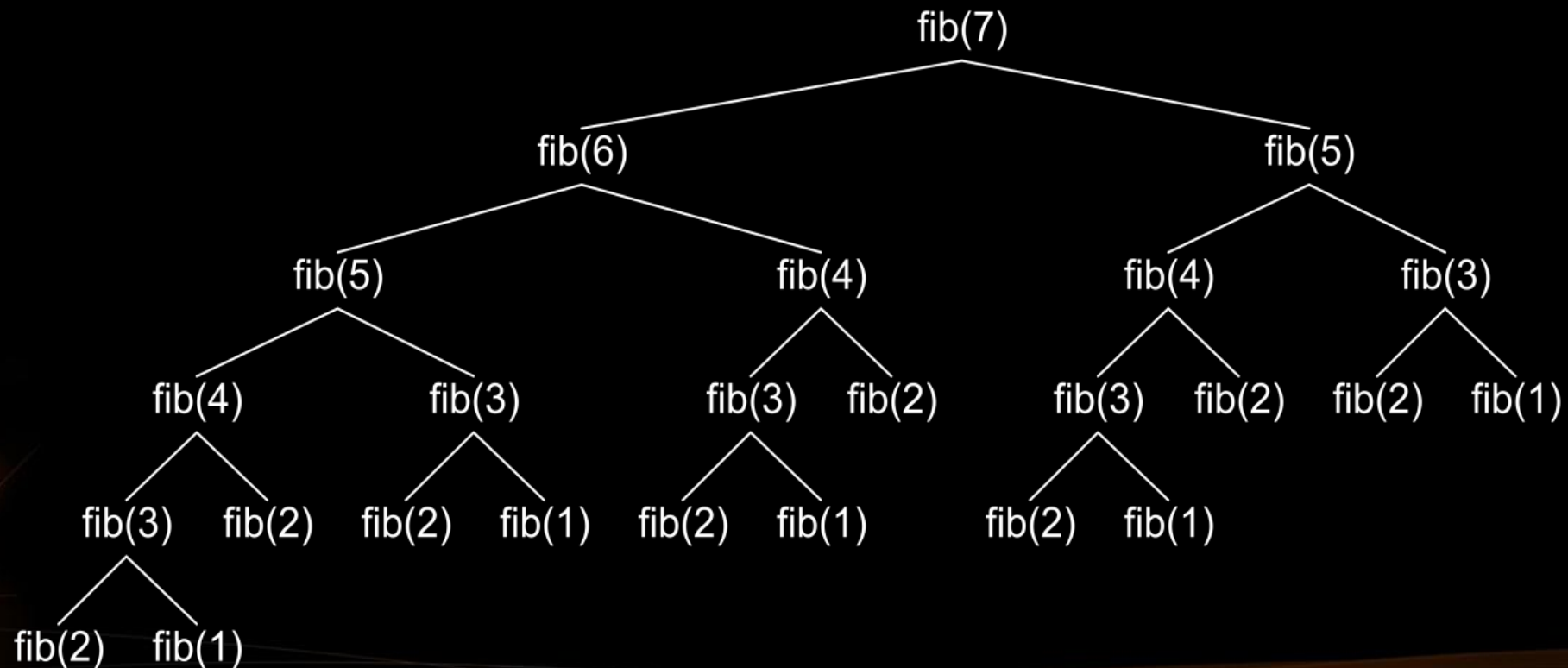
- Когато се използва неправилно, рекурсията може да отнеме прекалено много памет и изчислителна мощ

```
static decimal Fibonacci(int n)
{
    if ((n == 1) || (n == 2))
        return 1;
    else
        return Fibonacci(n - 1) + Fibonacci(n - 2);
}

static void Main()
{
    Console.WriteLine(Fibonacci(10)); // 89
    Console.WriteLine(Fibonacci(50)); // това ще увисне!
}
```

Как работи рекурсивното изчисляване на членовете на редицата на Фибоначи?

- **fib(n)** прави около **fib(n)** рекурсивни обръщания
- Една и съща стойност се изчислява многократно!



Кога да се ползва рекурсия?

- Избягвайте рекурсия, когато съществува очевиден итеративен алгоритъм
 - Примери: факториел, числа на Фибоначи

Обобщение

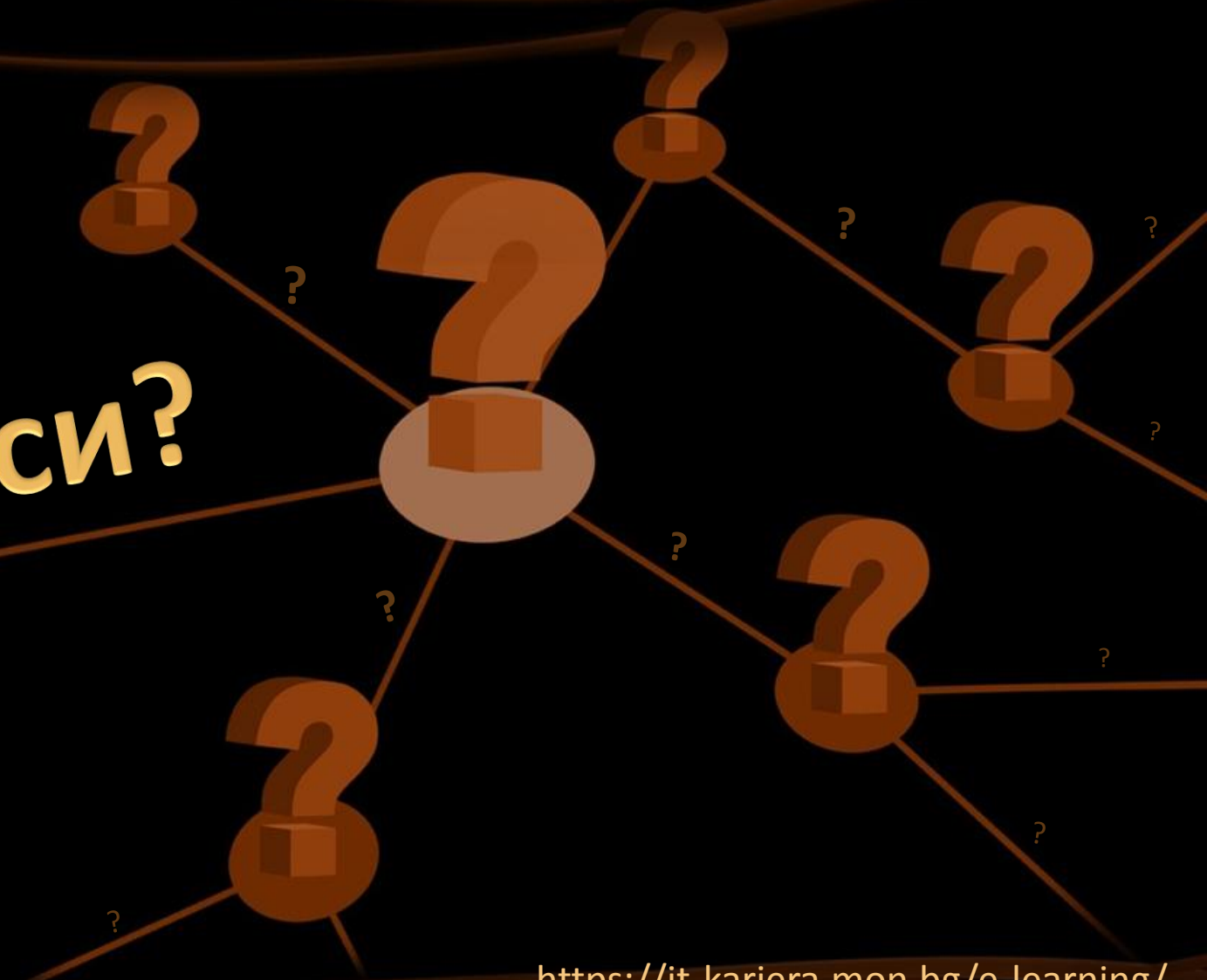
- Рекурсията имаме когато метод извиква сам себе си
 - Тя трябва да има **дъно(край)**, където спира
 - **Предварителни действия** са тези преди рекурсивното извикване - те се изпълняват в реда на извикване
 - **Последващи** са тези след него - те се изпълняват в обратен ред
- Рекурсията **може да бъде и вредна**, когато не се използва правилно



Рекурсия



Въпроси?



Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"



Министерство
на образованието
и науката



Национална
програма
„Обучение за
ИТ кариера“

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**



SoftUni
Foundation

