Доклад

По Функционално програмиране

**1.Парадигми за програмиране**

Обектно-ориентирана парадигма - следва императивен програмен модел. Променливи и обекти (изменяеми данни). Функциите и стойностите са различни концепции. Странични ефекти при изпълнение.

Функционална парадигма - следва декларативен програмен модел. Функциите са стойности. Стойностите не се променят по време на изпълнение на програмата. Висока ефективност на изпълнение и отложено изпълнение на код. По-малко възможности за грешки.

**2.Функционални езици**

* Чисти функционални езици - Haskell , Mercury , Clean
* Нечисти функционални езици - Lisp , Scala , Clojure , F# , C#

\*Функционални програмни езици, като Hope и Rex, са използвани повече в академичните среди, отколкото в развитието на търговски софтуер.

**3.Входно/изходни операции -**  Потребителски вход и изход в C# с примери:

**Примери за отпечатване на низ в конзолата в C#**

using System;<font></font>

namespace FirstProgram<font></font>

{<font></font>

class Program<font></font>

{<font></font>

static void Main(string[] args)<font></font>

{<font></font>

Console.WriteLine("Hello World!");<font></font>

}<font></font>

}<font></font>

}<font></font>

**Пример за отпечатване на променливи и литерали с помощта на метода WriteLine() и Write() в C#.**

using System;<font></font>

namespace FirstProgram<font></font>

{<font></font>

class Program<font></font>

{<font></font>

static void Main(string[] args)<font></font>

{<font></font>

Console.WriteLine("Prints on ");<font></font>

Console.WriteLine("New line");<font></font>

<font></font>

Console.Write("Prints on ");<font></font>

Console.Write("Same line");<font></font>

}<font></font>

}<font></font>

}<font></font>

**Пример за получаване на въвеждане на низ от потребител в C#:**

using System;<font></font>

namespace FirstProgram<font></font>

{<font></font>

class Program<font></font>

{<font></font>

static void Main(string[] args)<font></font>

{<font></font>

int number1 = 15, number2 = 20, sum;<font></font>

sum = number1 + number2;<font></font>

Console.WriteLine("{0} + {1} = {2}", number1, number2, sum);<font></font>

}<font></font>

}<font></font>

}<font></font>

**4.Състояние на програма**

Една програма има състояние, когато е създадена да помни потребителски интеракции или предхождащи събития. Състоянието обикновено се пази в променливи, които представляват заделена компютърна памет.

* Глобално състояние на програма - достъпно от всеки контекст на програмата.
* Локално състояние на програма - достъпно само в рамките на определена функция/package.

Monad - стратегия за комбиниране на изчисления/операции в по-сложни такива. Всеки Всеки Monad представя:

return функция Комбинаторна функция bind Типизиран конструктор

Източници:

1.Парадигми за програмиране –

<https://github.com/BG-IT-Edu/School-Programming/blob/main/Courses/Applied-Programmer/Functional-Programming-Haskell/01-introduction.pdf>

2.Функционални езици –

<https://github.com/BG-IT-Edu/School-Programming/blob/main/Courses/Applied-Programmer/Functional-Programming-Haskell/01-introduction.pdf>

3.Входно/изходни операции –

<https://github.com/BG-IT-Edu/School-Programming/blob/main/Courses/Applied-Programmer/Functional-Programming-Haskell/01-introduction.pdf>

4.Състояние на програма –

<https://github.com/BG-IT-Edu/School-Programming/blob/main/Courses/Applied-Programmer/Functional-Programming-Haskell/01-introduction.pdf>

Изготвил: Валентин Кьосев 12 ,,А‘‘ клас