

# Функционално програмиране

---

спец. “Информационни системи”

# гл. ас. д-р Калоян Йовчев

- email: [k.yovchev@fmi.uni-sofia.bg](mailto:k.yovchev@fmi.uni-sofia.bg)

# Примерна задача 1

Да се напише на Haskell функция `sumUnique`, която по списък от списъци от цели числа намира сумата на тези от числата, които са уникални в рамките на списъка, в който се срещат.

*Примери:*

- `sumUnique [[1,2,3,2],[-4,-4],[5]]`  $\rightarrow 9$  ( $= 1+3+5$ )
- `sumUnique [[2,2,2],[3,3,3],[4,4,4]]`  $\rightarrow 0$
- `sumUnique [[1,2,3],[4,5,6],[7,8,9]]`  $\rightarrow 45$

# Примерна задача 1 - Решение

Да се напише на Haskell функция `sumUnique`, която по списък от списъци от цели числа намира сумата на тези от числата, които са уникални в рамките на списъка, в който се срещат.

*Решение:*

- `sumUnique =`  
    `sum . map (\l -> sum (filter (\x -> notElem x (delete x l)) l))`

## Примерна задача 2

- Да се дефинира функция `maximize`, за която оценката на обръщението `maximize l`, където `l` е непразен списък от едноместни числови функции, да е едноместна числова функция на аргумент `x`, която дава стойността  $f(x)$  на тази функция  $f$  от списъка `l`, за която числото  $f(x)$  е най-голямо по абсолютна стойност.

*Пример:*

- Ако `fn = maximize [(\x -> x*x*x),(\x -> x+1)]`,  
то `fn 0.5`  $\rightarrow$  1.5, а `fn (-2)`  $\rightarrow$  -8

## Примерна задача 2 – Решение

- Да се дефинира функция `maximize`, за която оценката на обръщението `maximize l`, където `l` е непразен списък от едноместни числови функции, да е едноместна числова функция на аргумент `x`, която дава стойността  $f(x)$  на тази функция  $f$  от списъка `l`, за която числото  $f(x)$  е най-голямо по абсолютна стойност.

*Решение:*

- `maximize fs = maximum . (\x -> [(f x) | f <- fs])`

# Ниво на абстракция

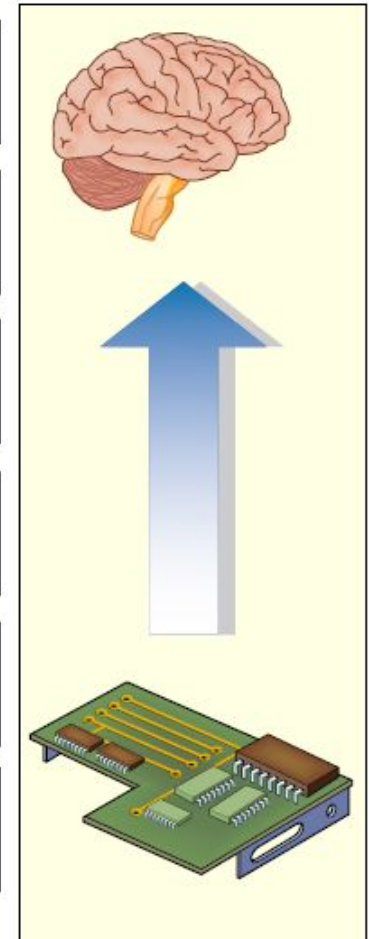
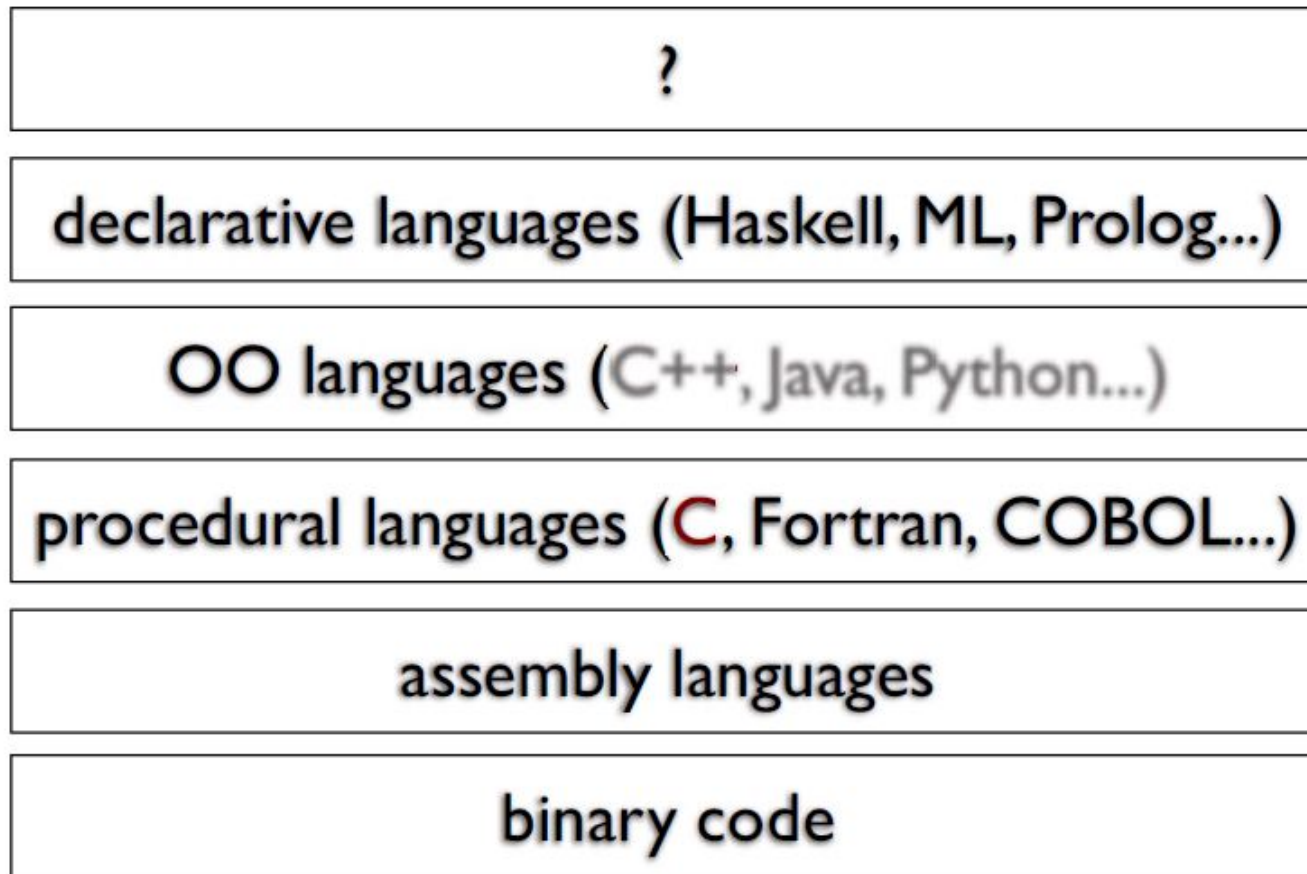


Figure by MIT OpenCourseWare.

# Процедурна парадигма

- позната от часовете по УП
- програма = множество от процедури, една начална (входна) и множество извиквания помежду им, евентуално разделени в отделни файлове
- може да има процедури със странични ефекти
- императивен стил на програмиране
- примери за езици, в които е застъпена - C



# Обектно-ориентирана парадигма

- позната от часовете по ООП
- основната програмна единица е класът. Класовете моделират някакви обекти от реалния свят или домейна на проблема, който решаваме.
- всеки обект има поведение (множество от методи, които притежава) и състояние (множество от неговите полета)
- може да имаме методи със странични ефекти
- императивен стил на програмиране
- примери за езици, в които е застъпена - Java, C++, Python, Ruby, други

# Функционална парадигма

- програмите са минималистични
- декларативен стил на програмиране
- програма = съвкупност от функции, в математическия смисъл на понятието
- всяка функция връща резултат и няма странични ефекти (чисто функционални езици)
- примери за езици, в които е застъпена - диалекти на Lisp - Scheme, Common Lisp, Haskell. Елементи в Javascript, Ruby, Python, други езици (лямбда изрази в Java 8, C#)

# Функционална парадигма - специфики

- липса на странични ефекти
- от една страна неоптималност на изчисленията - многократни пресмятания на едни и същи функции, поради невъзможност за запазване на резултата
- голяма полза при разработване на многонишкови приложения - решава основния проблем при разработка на такива приложения - необходимост от синхронизация при достъп до споделено състояние. Имаме липса на състояние (странични ефекти) липса на необходимост от синхронизация
- обикновено, в “чисто” функционалните езици няма оператори за цикъл - симулира се чрез рекурсия
- липсата на странични ефекти и декларативния стил ни карат да разбиваме решаваната задача на подзадачи, за които да пишем отделни функции - добър стил

# Задачата „Факториел“

- $n! = 1.2.3....n$ , където  $n$  е естествено число

- Решение в процедурен стил:

```
int fact(int n) {  
    int res = 1;  
    for (int i = 2; i <= n; ++i)  
        res *= i;  
    return res;  
}
```

- Решение във функционален стил:

```
fact n = if (n == 0) then 1 else (n * (fact (n - 1)))
```

# Haskell

- Препоръчвам за целта на упражненията да използвате за редактор Visual Studio Code с разширения Code Runner и Haskell Syntax Highlighting

- Главна функция:

```
main = do  
    print "Hello world!"  
    print (3 * 3)
```

*Заб.: Идентацията, както и скобите, са важни!!!*

# Основен синтаксис на Haskell - 1

- Основни типове:
  - Цели числа: 1, 2, 3, ...
  - Реални числа: 1.2, 1.22, ...
  - Символи (с единични кавички): 'a', 'b', ...
  - Низове (с двойни кавички): "aaaa", "bbbb", ...
  - Булеви стойности: True, False

# Основен синтаксис на Haskell - 2

- Основни оператори:
  - аритметични операции: `+`, `-`, `*`, `^` (повдигане в степен естествено число), `**` (повдигане в реална степен), `div` (частно при целочислено деление), `mod` (остатък при целочислено деление), `abs` (абсолютна стойност)
  - оператори за сравнение: `>`, `>=`, `<`, `<=`, `==`, `/=` (различно)
  - булеви оператори: `&&` (конюнкция), `||` (дизюнкция), `not` (отрицание)

# Основен синтаксис на Haskell - 3

- Дефиниране на константи:

`<име> = <стойност>`

*Примери:*

- `n = 1`
- `name = 2`

- Дефиниране на функция:

`<име> <арг-1> <арг-2> ... <арг-n> = <тяло>`

*Примери:*

- `f1 a b c = a * b * c`
- `f2 a b = a * (b + b)`

*Заб.: = не е оператор за присвояване!!!*



# Основен синтаксис на Haskell - 4

- Условни изрази:

if <условие> then <израз-за-да> else <израз-за-не>

*Примери:*

- if (1 == 2) then "yes" else "no"
- if (n == 0) then 1 else (n \* (fact (n - 1)))

# Факториел на Haskell

```
main = do  
    print (fact 5)
```

```
fact n = if (n == 0) then 1 else (n * (fact (n - 1)))
```

# Рекурсия - видове

- линейно рекурсивна функция: такава, в която има единствено рекурсивно обръщение. Пример: рекурсивно пресмятане на факториел
- дървовидно рекурсивна функция: такава, в която рекурсивните обръщения са повече от едно. Пример: пресмятане на  $n$ -тото число на Фибоначи, чрез рекурсия
- опашково рекурсивна функция: такава, в която рекурсивното обръщение е последното нещо в тялото. В повечето случаи се връща стойността на това рекурсивно извикване. Пример: пресмятане на НОД