

Какво е функционално програмиране?

Трифон Трифонов

Функционално програмиране, 2023/24 г.

4 октомври 2023 г.

Тази презентация е достъпна под лиценза Creative Commons Признание-Некомерсиално-Споделяне на споделеното 4.0 Международен © ⓘ ⓘ ⓘ

Императивен стил

Описваме последователно изчислителните стъпки.

Неструктурирано програмиране

- 1 Въведи a , b
- 2 Ако $a = b$, към 6.
- 3 Ако $a > b$, към 5.
- 4 $b \leftarrow b - a$; към 2.
- 5 $a \leftarrow a - b$; към 2.
- 6 Изведи a
- 7 Край

Структурирано програмиране

- Въведи a , b
- Докато $a \neq b$
 - Ако $a > b$
 - $a \leftarrow a - b$
 - В противен случай
 - $b \leftarrow b - a$
- Изведи a

Декларативен стил

Описваме свойствата на желания резултат.

Програмиране с ограничения

- Дадени са a и b .
- Търсим d , такова че:
 - $1 \leq d \leq a, b$
 - “ d е делител на a ”
 - “ d е делител на b ”
 - d е възможно най-голямо,
 - където за дадени x и y :
 - “ x е делител на y ”, ако
 - намерим такова естествено число k , че
 - $1 \leq k \leq y$
 - $k * x = y$

Декларативен стил (2)

Описваме свойствата на желания резултат.

Логическо програмиране

- Описваме релацията над естествени числа $\text{gcd}(a, b, c)$
- $\forall a \text{ gcd}(a, a, a)$ [факт]
- $\forall a \forall b (a > b \wedge \forall c (\text{gcd}(a - b, b, c) \rightarrow \text{gcd}(a, b, c)))$ [правило 1]
- $\forall a \forall b (a < b \wedge \forall c (\text{gcd}(a, b - a, c) \rightarrow \text{gcd}(a, b, c)))$ [правило 2]
- Дадени са a, b
- Намери такова c , за което $\text{gcd}(a, b, c)$

Декларативен стил (2)

Описваме свойствата на желания резултат.

Логическо програмиране

- Описваме релацията над естествени числа $\text{gcd}(a, b, c)$
- $\forall a \text{ gcd}(a, a, a)$ [факт]
- $\forall a \forall b (a > b \wedge \forall c (\text{gcd}(a - b, b, c) \rightarrow \text{gcd}(a, b, c)))$ [правило 1]
- $\forall a \forall b (a < b \wedge \forall c (\text{gcd}(a, b - a, c) \rightarrow \text{gcd}(a, b, c)))$ [правило 2]
- Дадени са a, b
- Намери такова c , за което $\text{gcd}(a, b, c)$

Пример: Нека $a = 8, b = 12$. Тогава:

$$\xrightarrow{\text{факт}} \text{gcd}(4, 4, 4) \xrightarrow{\text{правило 1}} \text{gcd}(8, 4, 4) \xrightarrow{\text{правило 2}} \text{gcd}(8, 12, 4)$$

Декларативен стил (3)

Описваме свойствата на желания резултат.

Функционално програмиране

- Функцията над естествени числа $\text{gcd}(a, b)$ притежава следните свойства:
- $\text{gcd}(a, a) = a$ (свойство 1)
- $\text{gcd}(a - b, b) = \text{gcd}(a, b)$, ако $a > b$ (свойство 2)
- $\text{gcd}(a, b - a) = \text{gcd}(a, b)$, ако $b > a$ (свойство 3)
- Дадени са a, b
- Да се пресметне $\text{gcd}(a, b)$.

Декларативен стил (3)

Описваме свойствата на желания резултат.

Функционално програмиране

- Функцията над естествени числа $\text{gcd}(a, b)$ притежава следните свойства:
- $\text{gcd}(a, a) = a$ (свойство 1)
- $\text{gcd}(a - b, b) = \text{gcd}(a, b)$, ако $a > b$ (свойство 2)
- $\text{gcd}(a, b - a) = \text{gcd}(a, b)$, ако $b > a$ (свойство 3)
- Дадени са a, b
- Да се пресметне $\text{gcd}(a, b)$.

Пример: Нека $a = 8, b = 12$.

$$\text{gcd}(8, 12) \stackrel{\text{свойство 3}}{=} \text{gcd}(8, 4) \stackrel{\text{свойство 2}}{=} \text{gcd}(4, 4) \stackrel{\text{свойство 1}}{=} 4.$$

Още един пример

Да се намери сумата на квадратите на нечетните числа в списъка `l`.

Императивен стил

- Нека $s = 0$.
- За i от 1 до `length(l)`:
 - Ако `l[i]` е нечетно, то
 - $s = s + l[i]^2$.
- Изведи s .

Функционален стил

- От елементите на `l`...
- ... избираме нечетните, ...
- ... прилагаме над тях функцията x^2 ...
- ... и ги групираме с операцията $+$.

Още един пример (2)

C++:

```
int s = 0;
for(int i = 0; i < sizeof(l); i++)
    if (l[i] % 2 != 0)
        s += l[i] * l[i];
cout << s;
```

Още един пример (2)

C++:

```
int s = 0;
for(int i = 0; i < sizeof(l); i++)
    if (l[i] % 2 != 0)
        s += l[i] * l[i];
cout << s;
```

Scheme: (apply + (map square (filter odd? l)))

Още един пример (2)

C++:

```
int s = 0;
for(int i = 0; i < sizeof(l); i++)
    if (l[i] % 2 != 0)
        s += l[i] * l[i];
cout << s;
```

Scheme: (apply + (map square (filter odd? l)))

Haskell: foldr1 (+) [x² | x <- l, odd x]

Още един пример (2)

C++:

```
int s = 0;
for(int i = 0; i < sizeof(l); i++)
    if (l[i] % 2 != 0)
        s += l[i] * l[i];
cout << s;
```

Scheme: (apply + (map square (filter odd? l)))

Haskell: foldr1 (+) [x² | x <- l, odd x]

Haskell: sum . map (^2) . filter odd

Какво може да се сметне с компютър?

Нека $f : \mathbb{N} \rightarrow \mathbb{N}$ е функция над естествени числа.

Примери: $f(x) = x^2$, $f(x)$ = x -тото число на Фибоначи.

Какво може да се сметне с компютър?

Нека $f : \mathbb{N} \rightarrow \mathbb{N}$ е функция над естествени числа.

Примери: $f(x) = x^2$, $f(x) = x$ -тото число на Фибоначи.

Въпрос 1: Какво означава да изчислим f с компютър?

Какво може да се сметне с компютър?

Нека $f : \mathbb{N} \rightarrow \mathbb{N}$ е функция над естествени числа.

Примери: $f(x) = x^2$, $f(x) = x$ -тото число на Фибоначи.

Въпрос 1: Какво означава да изчислим f с компютър?

Въпрос 2: Какво означава “алгоритъм” или “програма”?

Какво може да се сметне с компютър?

Нека $f : \mathbb{N} \rightarrow \mathbb{N}$ е функция над естествени числа.

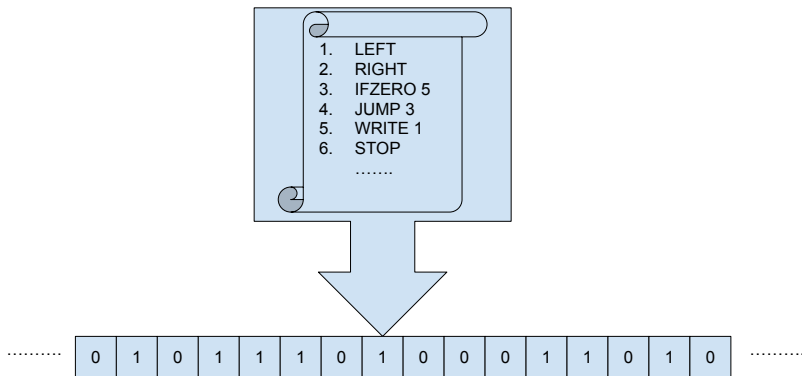
Примери: $f(x) = x^2$, $f(x) = x$ -тото число на Фибоначи.

Въпрос 1: Какво означава да изчислим f с компютър?

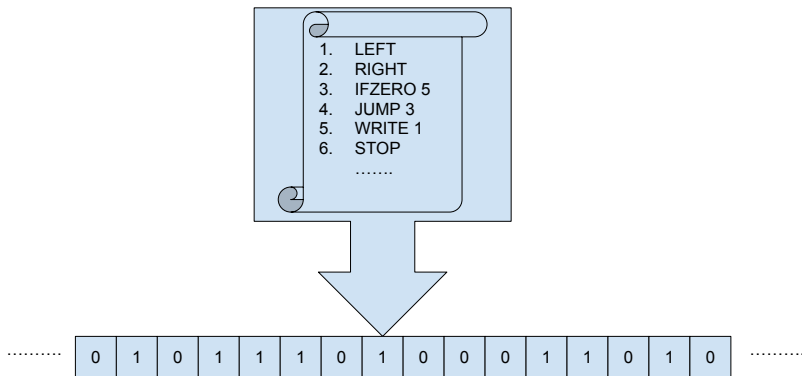
Въпрос 2: Какво означава “алгоритъм” или “програма”?

Въпрос 3: Има ли функции, които не могат да бъдат изчислени с компютър?

Машина на Turing

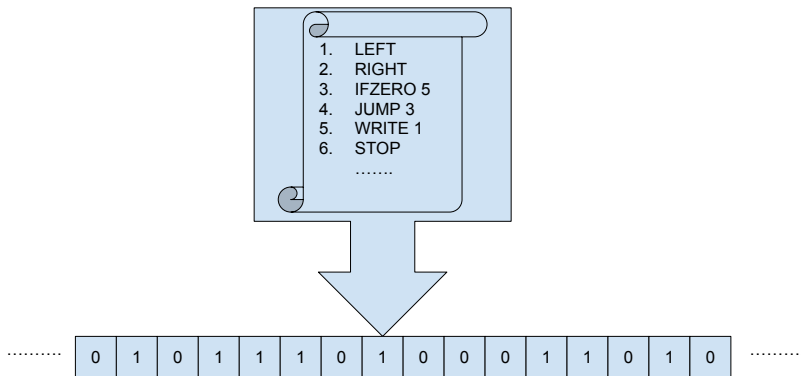


Машина на Turing



M изчислява функцията f_M , ако при лента с числото n машината M завършва и записва върху лентата числото $f_M(n)$.

Машина на Turing



M изчислява функцията f_M , ако при лента с числото n машината M завършва и записва върху лентата числото $f_M(n)$.

Ако M не завърши за някое n , казваме, че $f_M(n)$ не е дефинирана.

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).
- Но функциите от вида $\mathbb{N} \rightarrow \mathbb{N}$ са колкото редиците от естествени числа ...

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).
- Но функциите от вида $\mathbb{N} \rightarrow \mathbb{N}$ са колкото редиците от естествени числа ...
- ... които са неизброимо много! (защо?).

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).
- Но функциите от вида $\mathbb{N} \rightarrow \mathbb{N}$ са колкото редиците от естествени числа ...
- ... които са неизброимо много! (защо?).
- Следователно, има неизброимо много неизчислими функции. □

Има неизчислими функции!

- Всяка машина на Turing може да се кодира като дълго естествено число.
- Всяка изчислима функция се изчислява от (поне една) машина.
- Следователно, изчислимите функции са не повече от естествените числа (изброимо много).
- Но функциите от вида $\mathbb{N} \rightarrow \mathbb{N}$ са колкото редиците от естествени числа ...
- ... които са неизброимо много! (защо?).
- Следователно, има неизброимо много неизчислими функции. □
- Но кои са те?

Стоп проблем

Нека с $\{n\}$ означаваме машината на Turing с код n . Разглеждаме функцията:

$$halts(n) = \begin{cases} 1, & \text{ако } \{n\} \text{ завършва над лента с числото } n, \\ 0, & \text{иначе.} \end{cases}$$

Стоп проблем

Нека с $\{n\}$ означаваме машината на Turing с код n . Разглеждаме функцията:

$$halts(n) = \begin{cases} 1, & \text{ако } \{n\} \text{ завършва над лента с числото } n, \\ 0, & \text{иначе.} \end{cases}$$

halts не е изчислима!

Стоп проблем

Нека с $\{n\}$ означаваме машината на Turing с код n . Разглеждаме функцията:

$$halts(n) = \begin{cases} 1, & \text{ако } \{n\} \text{ завършва над лента с числото } n, \\ 0, & \text{иначе.} \end{cases}$$

halts не е изчислима!

Доказателство.

Да допуснем, че *halts* се изчислява от машина на Turing H . Дефинираме нова машина D :

1. (тук слагаме всички инструкции на H)

$k + 1$. IFZERO $k + 3$

$k + 2$. JUMP $k + 1$

$k + 3$. STOP

Нека $D = \{d\}$. Завършва ли D над d ?



Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$ -тото поред просто число

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$ -тото поред просто число
- $f_3(n) = n$ -тата цифра на числото π

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$ -тото поред просто число
- $f_3(n) = n$ -тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$ -тото поред просто число
- $f_3(n) = n$ -тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3 , които могат да се умножат в някакъв ред, така че да се получи O

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$ -тото поред просто число
- $f_3(n) = n$ -тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3 , които могат да се умножат в някакъв ред, така че да се получи O
- $f_6(n) = n$ е код на вярна съждителна формула

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$ -тото поред просто число
- $f_3(n) = n$ -тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3 , които могат да се умножат в някакъв ред, така че да се получи O
- $f_6(n) = n$ е код на вярна съждителна формула
- $f_7(n) = n$ е код на вярна предикатна формула

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$ -тото поред просто число
- $f_3(n) = n$ -тата цифра на числото π
- $f_4(n) = n$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3 , които могат да се умножат в някакъв ред, така че да се получи O
- $f_6(n) = n$ е код на вярна съждителна формула
- $f_7(n) = n$ е код на вярна предикатна формула
- $f_8(n) = m$, където $\{m\}$ пресмята f_8

Въпроси за изчислимост

Според вас изчислими ли са следните функции?

- $f_1(n) = n$ е просто число
- $f_2(n) = n$ -тото поред просто число
- $f_3(n) = n$ -тата цифра на числото π
- $f_4(n) =$ има n последователни седмици в числото π
- $f_5(n) = n$ е код на множество от матрици 3×3 , които могат да се умножат в някакъв ред, така че да се получи O
- $f_6(n) = n$ е код на вярна съждителна формула
- $f_7(n) = n$ е код на вярна предикатна формула
- $f_8(n) = m$, където $\{m\}$ пресмята f_8
- $f_9(n) =$ машините $\{n\}$ и $\{2n\}$ изчисляват еднакви функции

λ -смятане

Нека разполагаме с изброимо много променливи x, y, z, \dots

Три вида λ -изрази (E)

- x (променлива)
- $E_1(E_2)$ (апликация, прилагане на функция)
- $\lambda x E$ (абстракция, конструиране на функция)

Примери: $\lambda x x$, $(\lambda x x)(z)$, $\lambda f \lambda x f(f(f(x)))$

λ -смятане

Нека разполагаме с изброимо много променливи x, y, z, \dots

Три вида λ -изрази (E)

- x (променлива)
- $E_1(E_2)$ (апликация, прилагане на функция)
- $\lambda x E$ (абстракция, конструиране на функция)

Примери: $\lambda x x$, $(\lambda x x)(z)$, $\lambda f \lambda x f(f(f(x)))$

Едно изчислително правило:

$$(\lambda x E_1)(E_2) \mapsto E_1[x := E_2].$$

Машины на Turing = λ -смятане

Теорема (Alan Turing, 1937)

Функциите, които могат да се изчислят с машина на Turing са точно тези, които могат да се дефинират с λ -израз.

Машины на Turing = λ -смятане

Теорема (Alan Turing, 1937)

Функциите, които могат да се изчислят с машина на Turing са точно тези, които могат да се дефинират с λ -израз.

Машины на Turing	=	императивен стил за програмиране
λ -смятане	=	функционален стил за програмиране

Машины на Turing = λ -смятане

Теорема (Alan Turing, 1937)

Функциите, които могат да се изчислят с машина на Turing са точно тези, които могат да се дефинират с λ -израз.

Машины на Turing	=	императивен стил за програмиране
λ -смятане	=	функционален стил за програмиране

Факт: Практически всички съвременни езици за програмиране са със същата изчислителна сила като на машините на Turing.

Машины на Turing = λ -смятане

Теорема (Alan Turing, 1937)

Функциите, които могат да се изчислят с машина на Turing са точно тези, които могат да се дефинират с λ -израз.

Машины на Turing	=	императивен стил за програмиране
λ -смятане	=	функционален стил за програмиране

Факт: Практически всички съвременни езици за програмиране са със същата изчислителна сила като на машините на Turing.

Тезис на Church-Turing: Всяка функция, чието изчисление може да се автоматизира, може да бъде пресметната с машина на Turing.

Във функционалното програмиране...

... има:

- функции с параметри, (абстракция)

Във функционалното програмиране...

... има:

- функции с параметри, (абстракция)
- които могат да се прилагат над аргументи, (апликация)

Във функционалното програмиране...

... има:

- функции с параметри, (абстракция)
- които могат да се прилагат над аргументи, (апликация)
- които могат да са други функции (функции от висок ред)

Във функционалното програмиране...

... има:

- функции с параметри, (абстракция)
- които могат да се прилагат над аргументи, (апликация)
- които могат да са други функции (функции от висок ред)
- и могат да се дефинират чрез себе си, (рекурсия)

Във функционалното програмиране...

... има:

- функции с параметри, (абстракция)
- които могат да се прилагат над аргументи, (апликация)
- които могат да са други функции (функции от висок ред)
- и могат да се дефинират чрез себе си, (рекурсия)

... но няма:

Във функционалното програмиране...

... има:

- функции с параметри, (абстракция)
- които могат да се прилагат над аргументи, (апликация)
- които могат да са други функции (функции от висок ред)
- и могат да се дефинират чрез себе си, (рекурсия)

... но няма:

- памет

Във функционалното програмиране...

... има:

- функции с параметри, (абстракция)
- които могат да се прилагат над аргументи, (апликация)
- които могат да са други функции (функции от висок ред)
- и могат да се дефинират чрез себе си, (рекурсия)

... но няма:

- памет
- присвояване

Във функционалното програмиране...

... има:

- функции с параметри, (абстракция)
- които могат да се прилагат над аргументи, (апликация)
- които могат да са други функции (функции от висок ред)
- и могат да се дефинират чрез себе си, (рекурсия)

... но няма:

- памет
- присвояване
- цикли

Във функционалното програмиране...

... има:

- функции с параметри, (абстракция)
- които могат да се прилагат над аргументи, (апликация)
- които могат да са други функции (функции от висок ред)
- и могат да се дефинират чрез себе си, (рекурсия)

... но няма:

- памет
- присвояване
- цикли
- прескачане (goto, break, return)

Защо функционално програмиране?

- Кратки и ясни програми (изразителност)

Защо функционално програмиране?

- Кратки и ясни програми (изразителност)
- Лесна проверка за коректност

Защо функционално програмиране?

- Кратки и ясни програми (изразителност)
- Лесна проверка за коректност
- При еднакви входни данни връщат един и същ резултат (референциална прозрачност),

Защо функционално програмиране?

- Кратки и ясни програми (изразителност)
- Лесна проверка за коректност
- При еднакви входни данни връщат един и същ резултат (референциална прозрачност), което позволява...

Защо функционално програмиране?

- Кратки и ясни програми (изразителност)
- Лесна проверка за коректност
- При еднакви входни данни връщат един и същ резултат (референциална прозрачност), което позволява...
- Избягване на повторно пресмятане на резултати чрез запомняне (мемоизация)

Защо функционално програмиране?

- Кратки и ясни програми (изразителност)
- Лесна проверка за коректност
- При еднакви входни данни връщат един и същ резултат (референциална прозрачност), което позволява...
- Избягване на повторно пресмятане на резултати чрез запомняне (мемоизация)
- Премахване на части от програмата, които не участват в крайния резултат (мъртъв код)

Защо функционално програмиране?

- Кратки и ясни програми (изразителност)
- Лесна проверка за коректност
- При еднакви входни данни връщат един и същ резултат (референциална прозрачност), което позволява...
- Избягване на повторно пресмятане на резултати чрез запомняне (мемоизация)
- Премахване на части от програмата, които не участват в крайния резултат (мъртъв код)
- Пренареждане на програмата за по-ефективно изпълнение (стратегия за оценяване)

Защо функционално програмиране?

- Кратки и ясни програми (изразителност)
- Лесна проверка за коректност
- При еднакви входни данни връщат един и същ резултат (референциална прозрачност), което позволява...
- Избягване на повторно пресмятане на резултати чрез запомняне (мемоизация)
- Премахване на части от програмата, които не участват в крайния резултат (мъртъв код)
- Пренареждане на програмата за по-ефективно изпълнение (стратегия за оценяване)
- Паралелно изпълнение на независими части от програмата (паралелизация)

Видове функционални езици

- според типовата система
 - динамично типизирани (стойностите имат тип)
 - статично типизирани (променливите имат тип)
- според страничните ефекти
 - нечисти (със странични ефекти)
 - чисти (без странични ефекти)
- според стратегията за оценяване
 - стриктно (първо сметни, после предай)
 - лениво (първо предай, после смятай)

Видове функционални езици

- според типовата система
 - динамично типизирани (стойностите имат тип) [Scheme]
 - статично типизирани (променливите имат тип) [Haskell]
- според страничните ефекти
 - нечисти (със странични ефекти) [Scheme]
 - чисти (без странични ефекти) [Haskell]
- според стратегията за оценяване
 - стриктно (първо сметни, после предай) [Scheme]
 - лениво (първо предай, после смятай) [Haskell]

История на функционалното програмиране

(1936) Church и Rosser дефинират λ -смятането

История на функционалното програмиране

(1936) Church и Rosser дефинират λ -смятането

(1960) McCarthy създава първия функционален език LISP

История на функционалното програмиране

- (1936) Church и Rosser дефинират λ -смятането
- (1960) McCarthy създава първия функционален език LISP
- (1975) Steele и Sussman създават **Scheme**, диалект на LISP

История на функционалното програмиране

- (1936) Church и Rosser дефинират λ -смятането
- (1960) McCarthy създава първия функционален език LISP
- (1975) Steele и Sussman създават **Scheme**, диалект на LISP
- (1977) Backus (авторът на FORTRAN) популяризира функционалния стил

История на функционалното програмиране

- (1936) Church и Rosser дефинират λ -смятането
- (1960) McCarthy създава първия функционален език LISP
- (1975) Steele и Sussman създават **Scheme**, диалект на LISP
- (1977) Backus (авторът на FORTRAN) популяризира функционалния стил
- (1985) Turner създава Miranda, първият комерсиален чист функционален език

История на функционалното програмиране

- (1936) Church и Rosser дефинират λ -смятането
- (1960) McCarthy създава първия функционален език LISP
- (1975) Steele и Sussman създават **Scheme**, диалект на LISP
- (1977) Backus (авторът на FORTRAN) популяризира функционалния стил
- (1985) Turner създава Miranda, първият комерсиален чист функционален език
- (1990) Публикувана е първата версия на **Haskell**

История на функционалното програмиране

- (1936) Church и Rosser дефинират λ -смятането
- (1960) McCarthy създава първия функционален език LISP
- (1975) Steele и Sussman създават **Scheme**, диалект на LISP
- (1977) Backus (авторът на FORTRAN) популяризира функционалния стил
- (1985) Turner създава Miranda, първият комерсиален чист функционален език
- (1990) Публикувана е първата версия на **Haskell**
- (1998) Отваряне на кода на реализацията на Erlang

История на функционалното програмиране

- (1936) Church и Rosser дефинират λ -смятането
- (1960) McCarthy създава първия функционален език LISP
- (1975) Steele и Sussman създават **Scheme**, диалект на LISP
- (1977) Backus (авторът на FORTRAN) популяризира функционалния стил
- (1985) Turner създава Miranda, първият комерсиален чист функционален език
- (1990) Публикувана е първата версия на **Haskell**
- (1998) Отваряне на кода на реализацията на Erlang
- (1990–2000) Функционални елементи в императивни езици: Python (1991), JavaScript (1995), Ruby (1995), ActionScript (1998)

История на функционалното програмиране

- (1936) Church и Rosser дефинират λ -смятането
- (1960) McCarthy създава първия функционален език LISP
- (1975) Steele и Sussman създават **Scheme**, диалект на LISP
- (1977) Backus (авторът на FORTRAN) популяризира функционалния стил
- (1985) Turner създава Miranda, първият комерсиален чист функционален език
- (1990) Публикувана е първата версия на **Haskell**
- (1998) Отваряне на кода на реализацията на Erlang
- (1990–2000) Функционални елементи в императивни езици: Python (1991), JavaScript (1995), Ruby (1995), ActionScript (1998)
- (2000–) Функционалният стил на програмиране превзема света: Scala (2003), F# (2005), C# (2007), Clojure (2007), C++11 (2011), Elixir (2011), Java 8 (2014)