

# Какво е функционално програмиране?

Трифон Трифонов

Функционално програмиране, 2025/26 г.

2 октомври 2025 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

- ➊ Въведи  $a, b$
- ➋ Ако  $a = b$ , към 6.
- ➌ Ако  $a > b$ , към 5.
- ➍  $b \leftarrow b - a$ ; към 2.
- ➎  $a \leftarrow a - b$ ; към 2.
- ➏ Изведи  $a$
- ➐ Край

## Структурирано програмиране

- ➊ Въведи  $a, b$
- ➋ Докато  $a \neq b$ 
  - ➌ Ако  $a > b$ 
    - ➍  $a \leftarrow a - b$
  - ➎ В противен случай
    - ➏  $b \leftarrow b - a$
- ➏ 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)

$$\forall x \exists! y (x, y) \in R$$

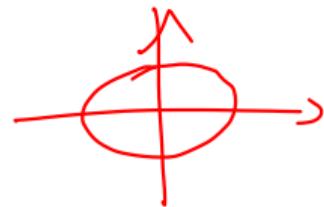
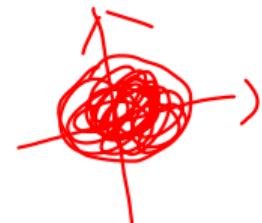
-1

Описваме свойствата на желания резултат.  
 Тоталнс објес.  $R \subseteq N^2$

$$\{ (x, y) \mid x^2 + y^2 \leq 1 \}$$

## Логическо програмиране

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

!  $\geq 1$  $\leq 1$ 

## Декларативен стил (2)

Описваме свойствата на желания резултат.

$$\{ (0,1), (1,2) \} \subseteq \mathbb{N}^2$$

### Логическо програмиране

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

$$\{ (x, y) \mid x \in \mathbb{N} \}$$

$$\forall x \forall y_1 y_2 (x, y_1) \in R \wedge (x, y_2) \in R \rightarrow y_1 = y_2$$

## Декларативен стил (2)

Описваме свойствата на желания резултат.

### Логическо програмиране

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

## Декларативен стил (2)

Описваме свойствата на желания резултат.

### Логическо програмиране

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

**Пример:** Нека  $a = 8, b = 12$ . Тогава:

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

$$\text{Числата } (4, 4, 4) \xrightarrow{1} \text{числата } (8, 4, 4) \xrightarrow{2} \text{числата } (8, 12, 4)$$

## Декларативен стил (3)

Описваме свойствата на желания резултат.

### Функционално програмиране

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

## Декларативен стил (3)

Описваме свойствата на желания резултат.

### Функционално програмиране

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

Пример: Нека  $a = 8, b = 12$ .

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

## 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  $\text{length}(l)$ :
  - 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 групирате 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 (+) [ x^2 | x <- l, odd x ]

{  $x^2$  |  $x \in l$ ,  $x$  - нечетн }

## 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;
```

$$\underbrace{f \circ g}_{\text{def}}(x) := f(g(x))$$

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

Haskell: foldr1 (+) [ x^2 | 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 Фиbonачи.

# 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 Фиbonачи.

**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 Фиbonачи.

**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.

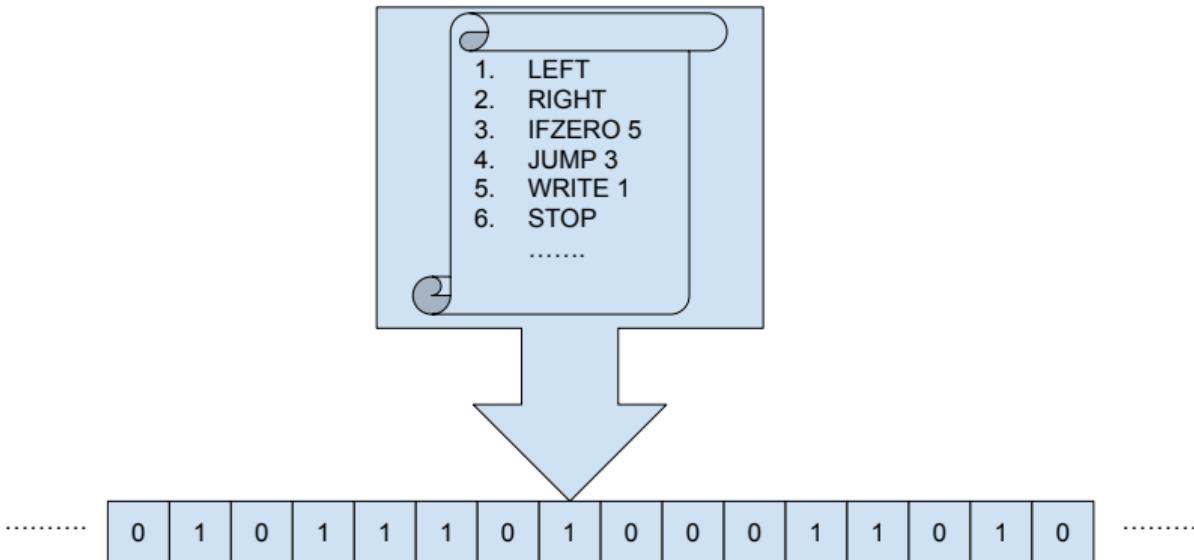
Примери:  $f(x) = x^2$ ,  $f(x)$  – x-тото число на Фиbonачи.

Въпрос 1: Какво означава да изчислим  $f$  с компютър?

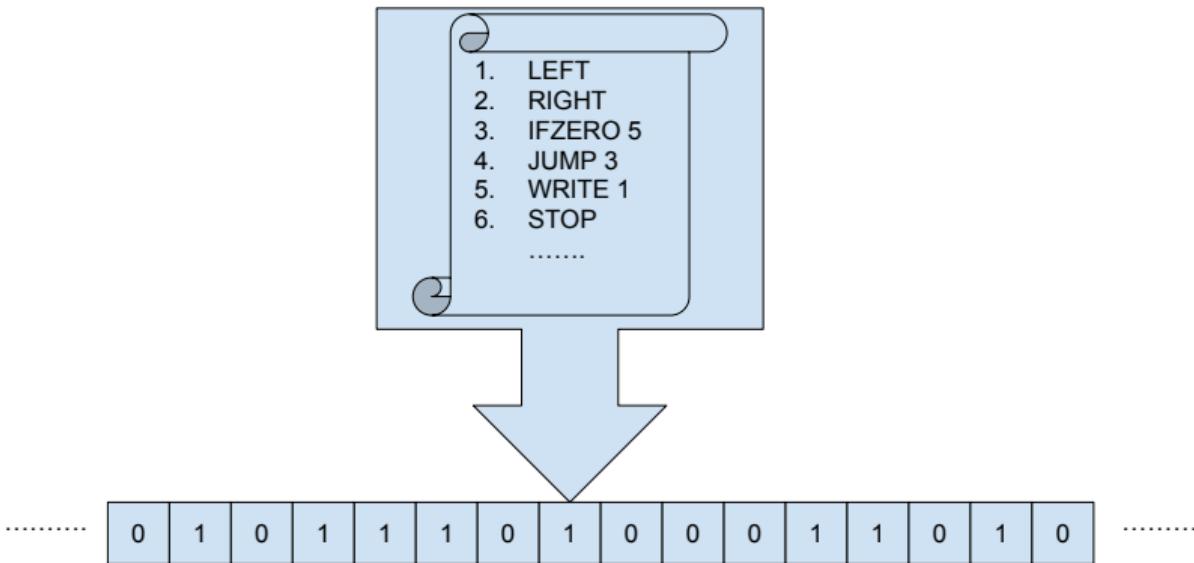
Въпрос 2: Какво означава „алгоритъм“ или „програма“?

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

# Машина на Turing

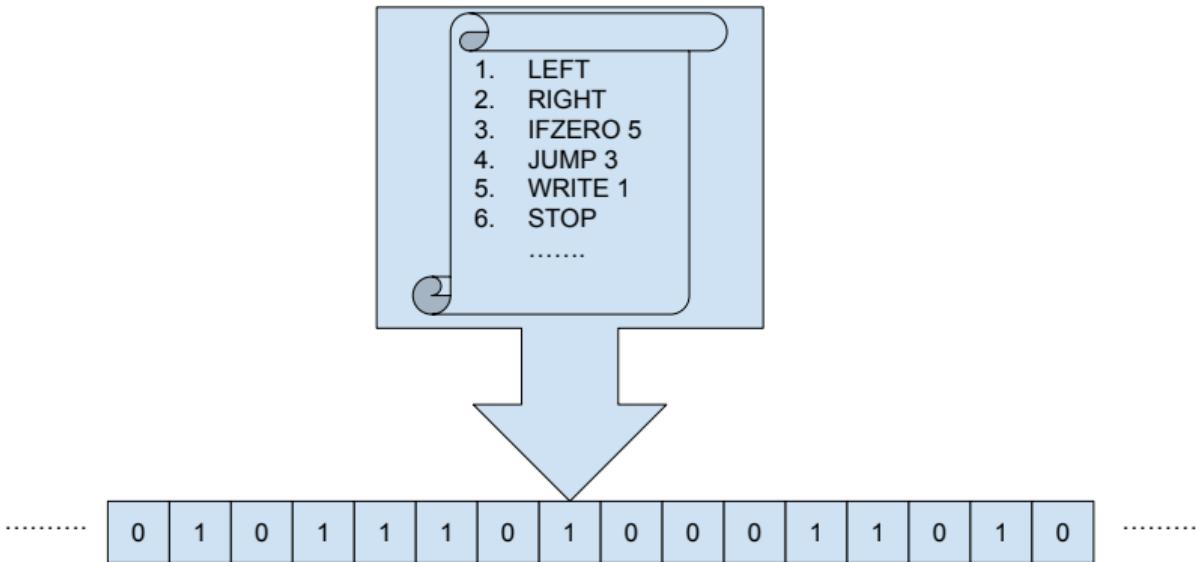


# Машина на Turing



$M$  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.

# $\lambda$ -смятане

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

Три вида  $\lambda$ -изрази ( $E$ )

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

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

# $\lambda$ -смятане

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

Три вида  $\lambda$ -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:

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

# Машини на Turing = $\lambda$ -смятане

Теорема (Alan Turing, 1937)

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

# Машини на Turing = $\lambda$ -смятане

Теорема (Alan Turing, 1937)

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

Машини на Turing	=	императивен стил за програмиране
$\lambda$ -смятане	=	функционален стил за програмиране

# Машини на Turing = $\lambda$ -смятане

Теорема (Alan Turing, 1937)

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

Машини на Turing	=	императивен стил за програмиране
$\lambda$ -смятане	=	функционален стил за програмиране

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

# Машини на Turing = $\lambda$ -смятане

Теорема (Alan Turing, 1937)

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

Машини на Turing	=	императивен стил за програмиране
$\lambda$ -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)

# Зашо функционално програмиране?

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

# Зашо функционално програмиране?

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

# Зашо функционално програмиране?

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

# Зашо 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)

# Зашо 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 ( memoизация )
- Премахване на части от програмата, които не участват в крайния резултат ( мъртъв код )

# Зашо 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 ( memoизация )
- Премахване на части от програмата, които не участват в крайния резултат ( мъртъв код )
- Пренареждане на програмата за по-ефективно изпълнение ( стратегия за оценяване )

# Зашо 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 ( memoизация )
- 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
  - нечиести (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
  - нечиести (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  $\lambda$ -смятането

# История на функционалното програмиране

- (1936) Church и Rosser дефинират  $\lambda$ -смятането
- (1960) McCarthy създава първия функционален език LISP

# История на функционалното програмиране

- (1936) Church и Rosser дефинират  $\lambda$ -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  $\lambda$ -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  $\lambda$ -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  $\lambda$ -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  $\lambda$ -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 реализациите на Erlang

# История на функционалното програмиране

- (1936) Church и Rosser дефинират  $\lambda$ -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 реализациите 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  $\lambda$ -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 реализациите 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), AWS Lambda (2014), Azure Functions (2016)