Модуль 1

Введение в функциональное программирование

Курс «Функциональное и логическое программирование на Haskell»

Парадигмы программирования

Как мы думаем о вычислениях?

Императивная

Последовательность команд, изменение состояния, циклы

Примеры:

C, Java, Python

Функциональная

Функции как значения, отсутствие побочных эффектов, рекурсия

Примеры:

Haskell, Erlang, Lisp

Логическая

Факты и правила, унификация, поиск с возвратом

Примеры:

Prolog, Datalog

Императивная vs Декларативная

Императивный подход: КАК

```
-- Сумма списка (Python-стиль)
total = 0
for x in [1,2,3,4,5]:
    total = total + x
print(total)
```

Декларативный подход: ЧТО

```
-- Сумма списка (Haskell)
sum [1,2,3,4,5]

-- Или через fold:
foldl (+) 0 [1,2,3,4,5]
```

Императивный стиль описывает шаги, декларативный — результат

Ключевые различия

Характеристика	Императивная	Функциональная
Состояние	Изменяемое (mutable)	Неизменяемое (immutable)
Основной элемент	Инструкция / оператор	Выражение / функция
Управление потоком	Циклы, условия, goto	Рекурсия, pattern matching
Побочные эффекты	Повсеместны	Контролируемы (IO)
Ссылочная прозрачность	Нет	Да

Тема 2

История ФП и λ-исчисление

От Чёрча до Haskell: как математика стала программированием

Хронология функционального программирования

1936
λ-исчисление
А. Чёрч

1958
Lisp
Дж. Маккарти

1973
ML
Р. Милнер

1990
Haskell 1.0
Комитет

2010
Haskell
Платформа

λ-исчисление Чёрча

Формальная система для определения вычислимых функций. Три ключевых элемента: переменные (x), абстракция (λx.M) и применение (M N). Любое вычисление можно выразить через эти три конструкции. λ-исчисление является теоретическим фундаментом всех функциональных языков.

```
-- Примеры λ-выражений
λx. x          -- тождество
λf. λx. f x    -- применение
(λx. x + 1) 5 -- = 6
```

Почему Haskell?

Чистая функциональность

Побочные эффекты строго изолированы в системе типов (IO-монада)

Мощная система типов

Статическая типизация, вывод типов, алгебраические типы данных

Ленивые вычисления

По умолчанию: значения вычисляются только когда нужны

Лаконичный синтаксис

Выразительный код, pattern matching, list comprehensions

Академическая ценность

Идеальная платформа для изучения фундаментальных концепций ФП

Тема 3

Установка GHC, GHCi
Основы синтаксиса

Настройка рабочего окружения и первые шаги в Haskell

Установка и настройка среды

GHCup — менеджер установки

Что устанавливается:

- GHC — компилятор Haskell
- GHCi — интерактивный интерпретатор (REPL)
- Cabal / Stack — системы сборки
- HLS — Language Server для IDE

```
curl --proto '=https' --tlsv1.2
  -sSf https://get-ghcup.haskell.org
  | sh
```

Рекомендуемые редакторы

VS Code + Haskell Extension
Лучший выбор для начинающих. Подсветка, автодополнение, подсказки типов.

Другие варианты:

- IntelliJ IDEA + HaskForce
- Vim / Neovim + HLS
- Emacs + haskell-mode

Первые шаги в GHCi

```
$ ghci
GHCi, version 9.8.1: https://www.haskell.org/ghc/
Prelude> 2 + 3
5
Prelude> "Hello" ++ " World"
"Hello World"
Prelude> let double x = x * 2
Prelude> double 21
42
Prelude> :type double
double :: Num a => a -> a
Prelude> :quit
```

Основы синтаксиса Haskell

Определение функций

```haskell
-- Простая функция
square x = x * x

-- Функция с условием (guards)
abs' n
  | n >= 0    = n
  | otherwise = -n

-- where-блок
bmi h w = w / h ^ 2
  where h = height / 100
```

Операторы и выражения

```haskell
-- Арифметика
3 + 4, 10 `div` 3, 2 ^ 10

-- Логика
True && False  -- False
not True       -- False

-- Строки
"Hello" ++ " " ++ "World"

-- if-then-else (выражение!)
if x > 0 then "pos" else "neg"
```

Тема 4

Базовые типы данных, функции и pattern matching

Строгая типизация — основа надёжности Haskell

Базовые типы данных

Тип	Описание	Примеры
`Int`	Целое число (фиксированная точность)	`42, -7, 0`
`Integer`	Целое число (произвольная точность)	`999999999999`
`Double`	Число с плавающей точкой	`3.14, 2.718`
`Bool`	Логический тип	`True, False`
`Char`	Символ	`'a', 'Я', '\n'`
`String`	Строка (= [Char])	`"Привет"`
`(a, b)`	Кортеж (фиксированный размер)	`(1, "ok", True)`
`[a]`	Список (однородный)	`[1,2,3], []`

Pattern Matching

Деконструкция значений по образцу — одна из самых мощных возможностей Haskell

Факториал

```haskell
-- Pattern matching на числах
factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n-1)

-- Pattern matching на списках
head' :: [a] -> a
head' []    = error "empty"
head' (x:_) = x
```

Кортежи и case

```haskell
-- Кортежи
fst' (a, _) = a
snd' (_, b) = b

-- case-выражение
descr xs = case xs of
  []  -> "пусто"
  [x] -> "один"
  _   -> "много"
```

Итоги модуля 1

- Функциональное программирование описывает ЧТО вычислять, а не КАК
- λ-исчисление — математический фундамент всех ФП-языков
- Haskell — чистый, ленивый, со строгой типизацией
- GHCi — основной инструмент для экспериментов
- Pattern matching — ключевой механизм деконструкции данных

Следующий модуль:

Основы Haskell — списки, рекурсия, функции высшего порядка

Рекомендуемые ресурсы

«Изучай Haskell во имя добра!»

М. Липовача — основной учебник курса (доступен на русском)

Haskell Wiki — haskell.org/haskellwiki

Официальная документация и руководства

Видеолекции CS центра

Курс «Функциональное программирование на Haskell» (YouTube)

Exercism — exercism.org/tracks/haskell

Практические задачи с менторством

Вопросы?

Модуль 1: Введение в функциональное программирование