Peter the Great
Saint-Petersburg Polytechnic University

Ocaml: синтаксис и базовые типы

30/10/2025

Презентацию подготовили:
- Дрекалов Никита
- Соколов Артём

POLYTECH

План презентации

- Переменные и выражения
  - Объявление переменных, изменяемые значения
- Типы и вывод типов
  - Примеры, преобразования и операция +
- Функции
  - Объявление функции, функции с несколькими аргументами, анонимные функции
- Кортежи
- Списки
  - Добавление элемента, конкатенация, обход
- Pattern Matching
  - Пример

Переменные и выражения

Объявление переменных

- `OCaml` – функциональный язык, в котором всё – выражение, возвращающее значение
- `let` связывает имя с выражением
- `OCaml` неизменяемый по умолчанию – `x` нельзя переназначить

```
let x = 5;;
let name = "OCaml";;
x := !x + 1;;  (* Ошибка *)
```

Изменяемые значения

- `ref` создаёт ссылку на значение переменной
- `!` – разыменование (получение значения)
- `:=` – присвоение (обновление значения)

```
let mutable_x = ref 10;;
mutable_x := !mutable_x + 1;;
!mutable_x;;   (* int = 11 *)
```

Типы и вывод типов

Примеры

- `OCaml` – язык со строгой статической типизацией, но типы *почти всегда* выводятся автоматически

```
let a = 42;;              (* val a : int = 42 *)
let b = 3.14;;            (* val b : float = 3.14 *)
let c = "Привет";;        (* val c : string = "Привет" *)
let d = true;;            (* val d : bool = true *)
```

Преобразования и оператор сложения

- В `OCaml` нет неявных преобразований типов
- `OCaml` предоставляет готовые функции для перевода между типами
  - Общая структура функций: `<src-type>_of_<dest-type>`

```
int_of_float 3.14;; (* - : int = 3 *)
string_of_int 42;;  (* - : string = "42" *)

let a = 2 + 3;;     (* int *)
let b = 2.5 +. 3.1;; (* float *)
```

Функции

Объявление функции

```
let square x = x * x;;
```

- `square` : `int->int` – принимает `int`, возвращает `int`

- Функции с несколькими аргументами

```
let add x y = x + y;;
add 2 3;; (* - : int = 5 *)
```

- Все функции каррированы – `add 2 3 = (add 2) 3`

Анонимные функции

- Для объявления анонимных функций (лямбд) используется ключевое слово `fun`

```
let square = fun x -> x * x;;
square 2;;  (* - : int = 4 *)
```

Кортежи и списки

Кортежи

- `Кортеж` – упорядоченный набор фиксированной длины

```
let person = ("Forthey", 21, true);;
person;; (* Тип (string * int * bool) *)
```

- Извлечение элементов

```
let (x, y) = (10, 20);;
print_int x;;  (* 10 *)
print_int y;;  (* 20 *)
```

Списки

- Тот же смысл, что и в других языках

```
let nums = [1; 2; 3];;
let words = ["OCaml"; "is"];;
```

- Добавление элемента:

```
0 :: !nums;; (* int list = [0; 1; 2; 3] *)
```

- Конкатенация:

```
let words_end = ["fun"];;
words @ words_end;; (* string list = ["OCaml"; "is"; "fun"] *)
```

- Обход списка

```
List.map (fun x -> x * 2) nums;; (* int list = [2; 4; 6] *)
```

Pattern Matching

Концепция

- `Pattern Matching` – это механизм, который:
  - Разбирает структуру данных (списки, кортежи, варианты и т. д.)
  - Проверяет, подходит ли она под определённый "образец" (pattern)
  - Выполняет соответствующий код
- В каком-то смысле это «улучшенный» `switch-case`

```
match выражение with
| шаблон1 -> результат1
| шаблон2 -> результат2
| _ -> результат_по_умолчанию
```

Простое сопоставление

```
let x = 3;;

match x with
| 0 -> "ноль"
| 1 -> "один"
| _ -> "другое";;

(* Результат: string = "другое" *)
```

Сумма элементов списка

```
let rec sum lst =
  match lst with
  | [] -> 0
  | x :: xs -> x + sum xs;;

sum [1; 2; 3];; (* int = 6 *)
```

- `match lst with` – проверяем , какая форма у списка.
- `| [] -> 0` – если список пуст, сумма равна 0.
- `| x :: xs -> x + sum xs` – если есть элементы:
  - `x` – первый элемент, `xs` – остальные
  - рекурсивно вызываем `sum` для хвоста и прибавляем первый элемент

Спасибо за внимание!