

Язык программирования Elixir



- **Функции и управление сложностью кода**

Выполнили Фролов Иван и Ткачев Михаил, гр. 5030102/20202

Входные данные

Входные данные передаются по ссылке

- `defmodule MyList do`
- `def add_item(list, item) do`
- `[item | list]` *# Возвращает НОВЫЙ список, старый не тронут*
- `end`
- `end`
-
- `original = [1, 2, 3]`
- `new_list = MyList.add_item(original, 0)`
-
- `IO.inspect(original) #=> [1, 2, 3]`
- `IO.inspect(new_list) #=> [0, 1, 2, 3]`



elixir

ВЫХОДНЫЕ ДАННЫЕ

- Любая функция в Elixir возвращает значение результата своего последнего выражения. Ключевое слово `return` отсутствует.

- `def square(x) do`
- `x * x`
- `end`
-
- `square(5) # => 25`



Рекурсия

- `defmodule Math do`
- `def factorial(0), do: 1` *# Базовый случай*
- `def factorial(n) when n > 0 do`
- `n * factorial(n - 1)` *# Рекурсивный случай*
- `end`
- `end`
-
- `Math.factorial(5) # => 120`

Elixir оптимизирует **хвостовую рекурсию**

- `defmodule TailRecursive do`
- `def sum_list(list, accumulator \\ 0)`
- `def sum_list([], accumulator), do:`
 `accumulator` *# Базовый случай*
- `def sum_list([head | tail], accumulator) do`
- `sum_list(tail, head + accumulator)` *#*
 Хвостовой вызов
- `end`
- `end`
-
- `TailRecursive.sum_list([1, 2, 3, 4]) # => 10`



Замыкания

- `defmodule ClosureExample do`
- `def create_multiplier(factor) do`
- `# Возвращаем анонимную функцию, которая`
`"запомнила" значение factor`
- `fn x -> x * factor end`
- `end`
- `end`
-
- `double = ClosureExample.create_multiplier(2)`
- `triple = ClosureExample.create_multiplier(3)`
-
- `double.(4) # => 8`
- `triple.(4) # => 12`



Пространства имён и Модули

В Elixir **модули** являются основным способом организации кода. Они действуют как пространства имён, группируя связанные функции вместе.

```
defmodule Geometry.Rectangle do
```

```
  def area(length, width) do
```

```
    length * width
```

```
  end
```

```
end
```

```
defmodule Geometry.Circle do
```

```
  @pi 3.14159
```

```
  def area(radius) do
```

```
    @pi * radius * radius
```

```
  end
```

```
end
```

```
Geometry.Circle.area(10) # => 314.159
```

```
Geometry.Rectangle.area(5, 3) # => 15
```



elixir

Пакеты

В контексте Elixir и его менеджера зависимостей Mix, понятие «пакета» относится скорее к проекту или библиотеке. Пакет — это набор модулей, поставляемых вместе. Вы управляете ими через файл `mix.exs`. Например, при добавлении зависимости `{:phoenix, "~> 1.7"}` вы добавляете целый пакет модулей в свой проект



Экспорт и Импорт

- `defmodule SecretKeeper do`
- *# Эта функция экспортируется и может быть вызвана извне*
- `def public_api do`
- `secret_calculation() # Вызов приватной функции изнутри`
- `end`
-
- *# Эта функция приватная (defp) и доступна только внутри своего модуля*
- `defp secret_calculation do`
- `42`
- `end`
- `end`

- *# Без импорта*
- `List.flatten([1, [2], 3])`
-
- *# С импортом модуля List*
- `import List`
- `flatten([1, [2], 3])`
- **Alias создаёт псевдоним для модуля.**
- `alias Geometry.Circle, as: Circle`
- *# Теперь вместо `Geometry.Circle.area(10)` можно писать:*



elixir

ВЫВОДЫ

- **Чистые функции**, работающие с **неизменяемыми данными**, делают код предсказуемым и легким для тестирования.
- **Рекурсия** и **замыкания** предоставляют гибкие инструменты для создания логики.
- **Модули**, **импорт/экспорт** и **пакеты** позволяют организовывать этот код, скрывать сложность и строить масштабируемые и надежные приложения

Источники

- elixir-lang.org — официальный сайт языка
- Wikipedia — статья Elixir (programming language)

