Тема 2

5 case, cond и if

В тази тема ще научим структурите case, cond и if.

5.1 case

`case` ни позволява да сравним стойност с много модели, докато не намерим съвпадащ:

```
iex> case {1,2,3}   do
...>{4,5,6}->
...>"This clause won't match"
...>{1, x,3}->
...>"This clause will match and bind x to 2 in this clause"
...>_->
...>"This clause would match any value"
...> end
```

Ако искате да съвпадате с модел срещу съществуваща променлива, трябва да използвате оператора ^:

```
iex>x=1
1
iex> case 10 do
...>^x->"Won't match"
...>_->"Will match"
...> end
```

Клаузите също така позволяват допълнителни условия да бъдат посочени чрез защити (предпазители):

```
iex> case {1,2,3}   do
...>{1, x,3}     when x>0->
...>"Will match"
...>_->
...>"Won't match"
...> end
```

Първата клауза по-горе ще съвпада само когато x е положително.

5.2 Изрази в защитни клаузи (Expressions in guard clauses).

Виртуалната машина на Erlang (VM) позволява само ограничен набор от изрази в предпазни клаузи:

• оператори за сравнение (==, !=, ===, !==, >, <, <=, >=)

• булеви оператори (`and`, `or`) и оператори за отрицание (`not`, `!`)

• аритметични оператори (+, -, *, /)

• <> и ++ стига лявата страна да е литерал

• операторът in

• всички по-долу изброени функции за проверка на типа:

- `is_atom/1`
- `is_binary/1`
- `is_bitstring/1`

- is_boolean/1
- is_float/1
- is_function/1
- is_function/2
- is_integer/1
- is_list/1
- is_map/1
- is_number/1
- is_pid/1
- is_port/1
- is_reference/1
- is_tuple/1

•плюс тези функции:

- abs(number)
- bit_size(bitstring)
- byte_size(bitstring)
- div(integer, integer)
- elem(tuple, n)
- hd(list)
- length(list)
- map_size(map)
- node()
- node(pid | ref | port)
- rem(integer, integer)
- round(number)
- self()
- tl(list)
- trunc(number)
- tuple_size(tuple)

Имайте предвид, че грешките в предпазните клаузи не изтичат, а просто карат защитата да се повреди:

```
iex>hd(1)
** (ArgumentError) argument error
    :erlang.hd(1)
iex> case 1 do
...>x    when hd(x)->"Won't match"
...>x->"Got:      #{x}"
...> end
"Got 1"
```

Ако нито една от клаузите не съвпада, възниква грешка:

```
iex> case :ok do
...>:error->"Won't match"
...> end
** (CaseClauseError) no case clausematching::ok
```

Забележете, че анонимните функции могат също да имат множество клаузи и защити (предпазители):

```
iex>f=   fn
...>x, y    when x>0->x+y
...>x, y->x      * y
...> end
#Function<12.71889879/2 in :erl_eval.expr/5>
iex>f.(1,3)
4
iex>f.(-1,3)
-3
```

Броят на аргументите във всяка клауза за анонимна функция трябва да бъде един и същ, в противен случай възниква грешка.

5.3 cond

case е полезен, когато трябва да съпоставите различни стойности. Въпреки това, при много обстоятелства искаме да проверим различни условия и да намерим първото, което се оценява като вярно. В такива случаи може да се използва cond:

```
iex> cond do
...>2+2==5->
...>"This will not be true"
...>2    * 2==3->
...>"Nor this"
...>1+1==2->
...>"But this will"
...> end
"But this will"
```

Това е еквивалентно на клаузи else if в много императивни езици (въпреки че тук се използва много по-рядко).

Ако нито едно от условията не върне true, възниква грешка. Поради тази причина може да се наложи да добавите крайно условие, равно на true, което винаги ще съвпада:

```
iex> cond do
...>2+2==5->
...>"This is never true"
...>2    * 2==3->
...>"Nor this"
...>true->
...>"This is always true (equivalent to else)"
...> end
```

И накрая, като забележка cond счита всяка стойност освен nil и false за истина:

```
iex> cond do
...>hd([1,2,3])->
...>"1 is considered as true"
...> end
"1 is considered as true"
```

5.4 Ако и освен ако (if and unless)

Освен case и cond, Elixir предоставя и макросите if/2 и unless/2, които са полезни, когато трябва да проверите само за едно условие:

```
iex> if true do
...>"This works!"
...> end
"This works!"
iex> unless true do
...>"This will never be seen"
...> end
nil
```

Ако условието, дадено на if/2, връща false или `nil`, тялото, дадено между do/end, не се изпълнява и просто връща нула. Обратното се случва с unless/2.

Те също така поддържат else блокове:

```
iex> if nil do
...>   "This won't be seen"
...> else
...>   "This will"
...> end
"This will"

Note: An interesting note regarding ``if/2`` and ``unless/2`` is
that they are implemented as macros in the language; they aren't
special language constructs as they would be in many languages. You
can check the documentation and the source of ``if/2`` in `the
``Kernel`` module docs </docs/stable/elixir/Kernel.html>`_. The
``Kernel`` module is also where operators like ``+/2`` and functions
like ``is_function/2`` are defined, all automatically imported and
available in your code by default.
```

5.5 `do/end` блокове (`do/end` blocks)

До този момент научихме четири контролни структури: case, cond, if и unless и всички те бяха обвити в do/end блокове. Също така можем да напишем и if, както следва:

```
iex> if true,do:1+2
3
```

В Elixir блоковете do/end са удобство за предаване на група изрази към do:. Те са еквивалентни:

```
iex> if true do
...>a=1+2
...>a+10
...> end
13
iex> if true,do: (
...>a=1+2
...>a+10
...>)
13
```

Казваме, че вторият синтаксис използва списъци с ключови думи (**keyword lists**). Можем да запишем else, използвайки този синтаксис:

```
iex> if false,do::this,else::that
:that
```

Едно нещо, което трябва да имате предвид, когато използвате do/end блокове, е, че те винаги са обвързани с най-външното извикване на функция. Например следният израз:

```
iex>is_number  if true do
...>1+2
...> end
```

Ще бъде анализиран като:

```
iex>is_number( if true) do
...>1+2
...> end
```

Което води до грешка при недефинирана функция, тъй като Elixir се опитва да извика is_number/2. Добавянето на изрични скоби е достатъчно, за да разреши неяснотата:

```
iex>is_number( if true do
...>1+2
...> end)
true
```

Списъците с ключови думи играят важна роля в езика и са доста често срещани в много функции и макроси.