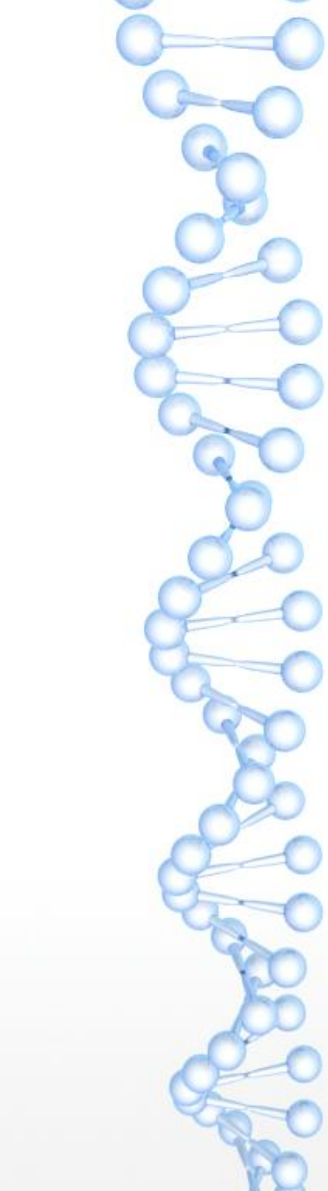
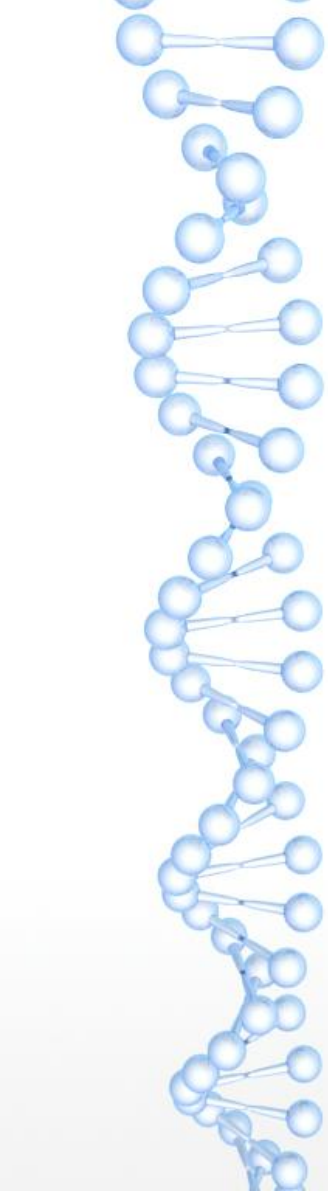


Функциональное и логическое программирование

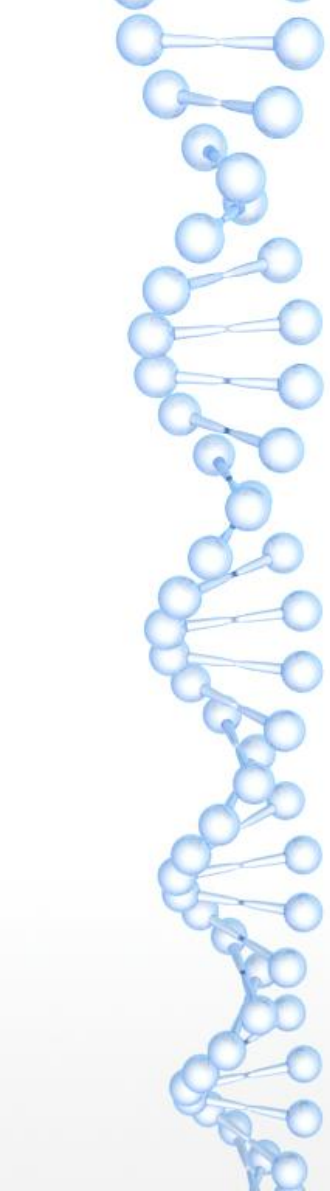
Лекция 10. Прикладное функциональное программирование



Активные шаблоны

Сопоставление с образцом расширяет возможности программиста, предоставляя более выразительный способ организации ветвлений в коде по сравнению с условными выражениями. Они позволяют выполнять сопоставление с константами, захватывать значения и даже выполнять сопоставление со структурами данных. Однако сопоставление с образцом имеет один существенный недостаток – оно позволяет выполнять сопоставление только с литералами, такими как строки, и со значениями некоторых типов, таких как массивы и списки.

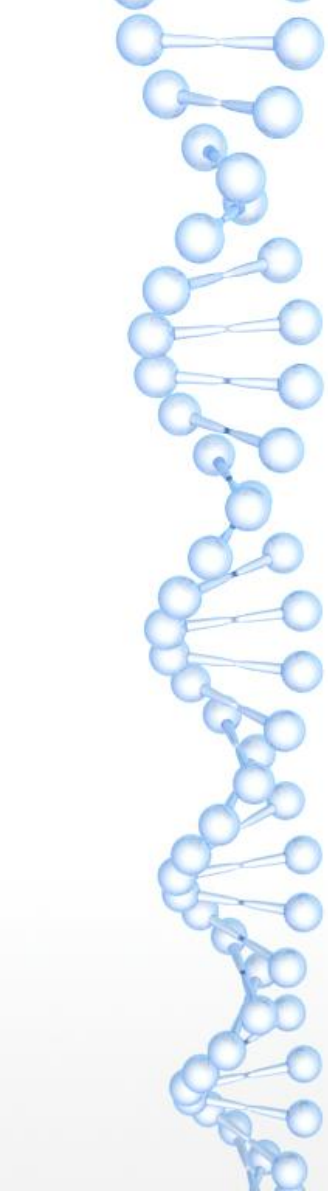
В идеале было бы здорово, если бы имелась возможность выполнять сопоставление с другими высокоуровневыми компонентами, такими как элементы последовательности `seq<_>`. Ниже приводится пример того, что можно было бы написать, но, к сожалению, этот код не компилируется:



Активные шаблоны

```
// Не компилируется
let containsVowel (word : string) =
    let letters = word.Chars
    match letters with
    | ContainsAny [ 'a'; 'e'; 'i'; 'o'; 'u' ]
        -> true
    | SometimesContains [ 'y' ]
        -> true
    | _ -> false
```

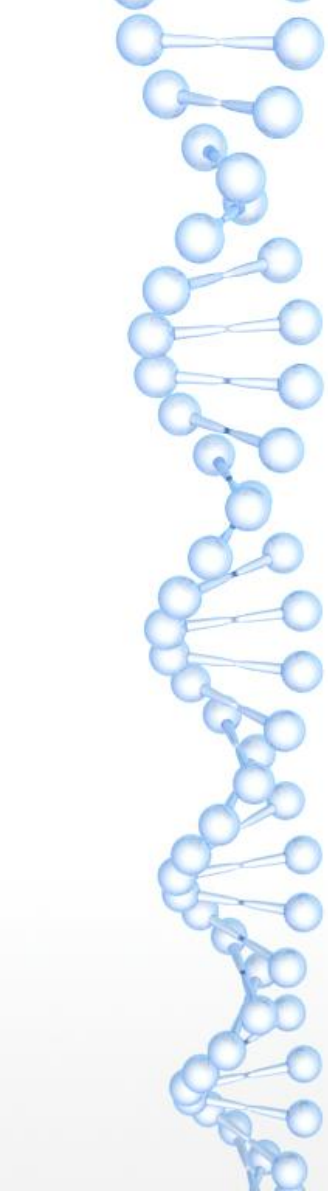
```
let containsVowel (word : string) =
    let letters = word |> Set.ofSeq
    match letters with
    | _ when letters.Contains('a') || letters.Contains('e') ||
             letters.Contains('i') || letters.Contains('o') ||
             letters.Contains('u') || letters.Contains('y')
        -> true
    | _ -> false
```



Активные шаблоны

К счастью, в языке F# есть возможность, позволяющая добиться желаемой выразительности и элегантности выражений сопоставления, под названием *активные шаблоны (active patterns)*.

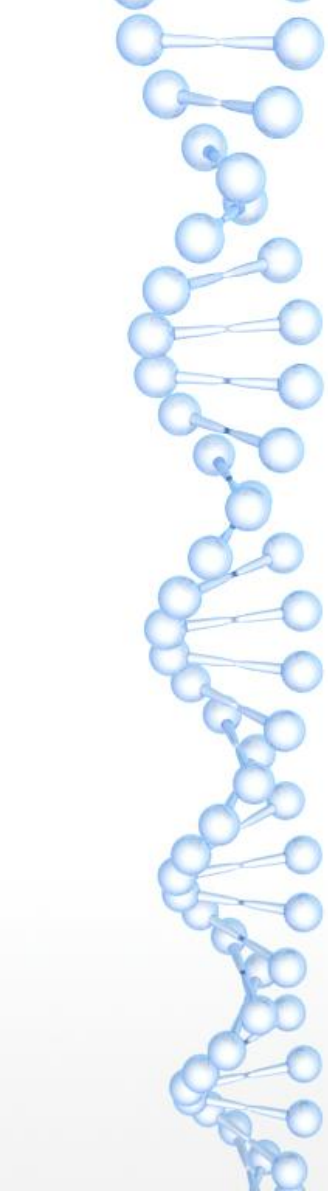
Активные шаблоны – это всего лишь особые функции, которые могут использоваться внутри правил сопоставления с образцом. Применение этих функций ликвидирует необходимость использования предложений `when`, а кроме того, делает реализацию выражений сопоставления более простой и понятной, что позволяет более наглядно отразить решаемую проблему.



Активные шаблоны

Существует несколько разновидностей активных шаблонов, каждая из которых принимает входные данные и преобразует их в одно или более новых значений:

- Одновариантные активные шаблоны (single-case active patterns) преобразуют входные данные в новое значение.
- Частичные активные шаблоны (partial-case active pattern) выделяют только часть пространства входных данных. Примером может служить определение строк, содержащих букву «е».
- Многовариантные активные шаблоны (multi-case active pattern) делят пространство входных данных на несколько значений. Примером может служить деление всех целых чисел на четные, нечетные и ноль.



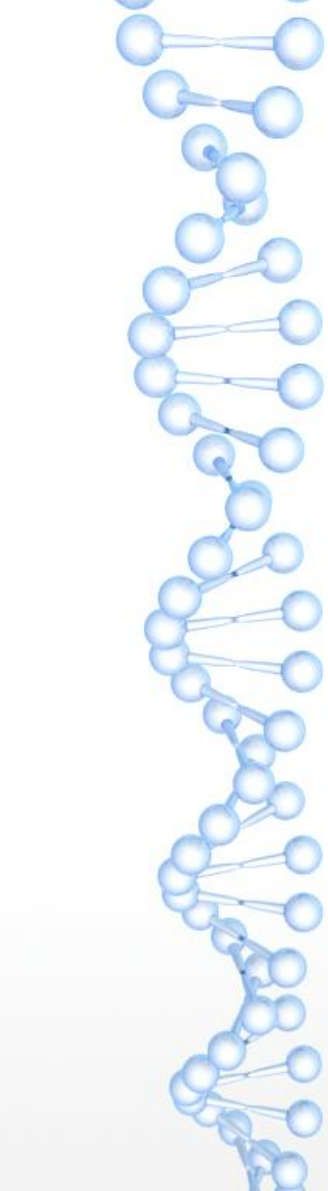
Одновариантные Активные шаблоны

Самой простой разновидностью активных шаблонов являются одновариантные активные шаблоны, которые преобразуют данные из одного типа в другой. Это позволяет использовать в выражениях сопоставления экземпляры классов и другие значения, которые иначе нельзя использовать в правилах сопоставления. Вскоре мы увидим пример использования этой возможности.

Чтобы определить активный шаблон, необходимо объявить функцию, заключенную в пары символов (| и |).

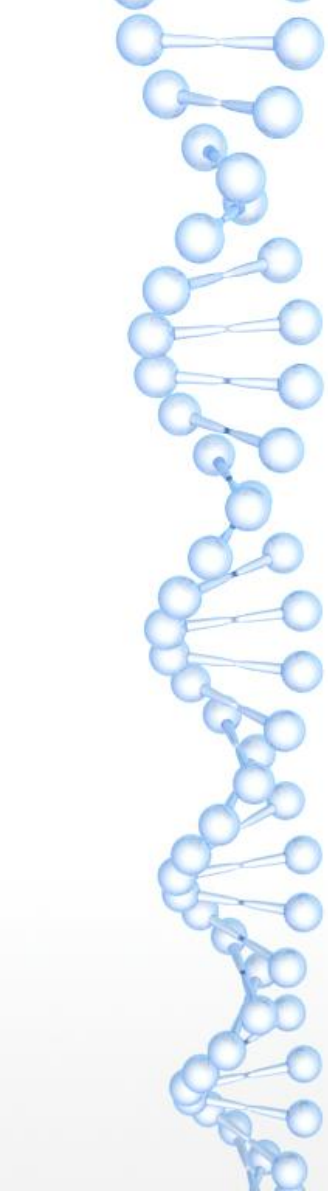
В примере 7.5 определяется активный шаблон, преобразующий имя файла в его расширение. Он позволяет выполнять сопоставление с расширениями файлов без использования предложения `when`.

Для применения активного шаблона `FileExtension` достаточно просто использовать его вместо правила сопоставления. Образец, с которым сопоставляется результат вызова активного шаблона, указывается сразу же за именем активного шаблона. Во всем остальном правило сопоставления с активным шаблоном ничем не отличается от любых других правил. Так, в примере ниже результат вызова активного шаблона сопоставляется с константой `".jpg"`. Другое правило сопоставления связывает новое значение `ext` с результатом вызова активного шаблона.



Одновариантные Активные шаблоны

```
open System.IO

// Преобразование имени файла в его расширение
let (|FileExtension|) filePath = Path.GetExtension(filePath)

let determineFileType (filePath : string) =
    match filePath with

    // Без применения активных шаблонов
    | filePath when Path.GetExtension(filePath) = ".txt"
        -> printfn "It is a text file."

    // Преобразование данных с помощью активного шаблона
    | FileExtension ".jpg"
    | FileExtension ".png"
    | FileExtension ".gif"
        -> printfn "It is an image file."

    // Связывает результат сопоставления с новым значением
    | FileExtension ext
        -> printfn "Unknown file extension [%s]" ext
```



Частичные активные шаблоны

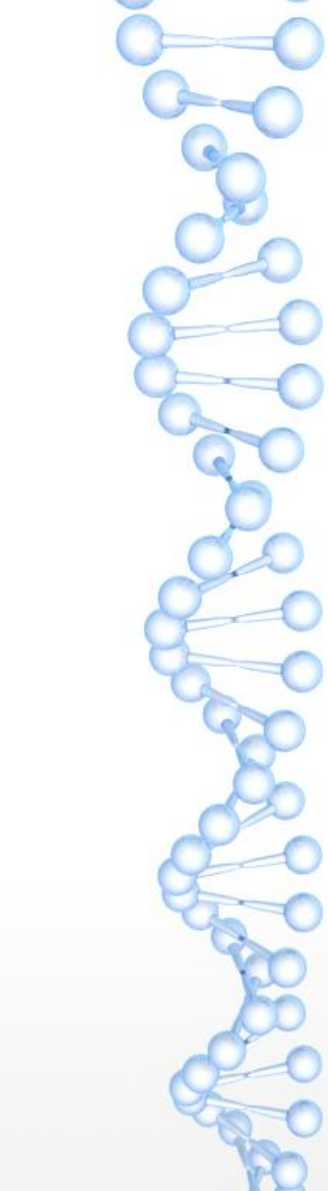
Одновариантные активные шаблоны удобно использовать для повышения наглядности выражений сопоставления. Однако существует множество ситуаций, когда преобразование данных возможно не всегда. Например, что может произойти, если написать активный шаблон, который преобразует строки в целые числа?

```
> // Активный шаблон, преобразующий строки в целые числа
open System
let (|ToInt|) x = Int32.Parse(x);;

val ( |ToInt| ) : string -> int

> // Проверить, является ли входная строка изображением числа 4
let isFour str =
    match str with
    | ToInt 4 -> true
    | _ -> false;;
```

Частичные активные шаблоны

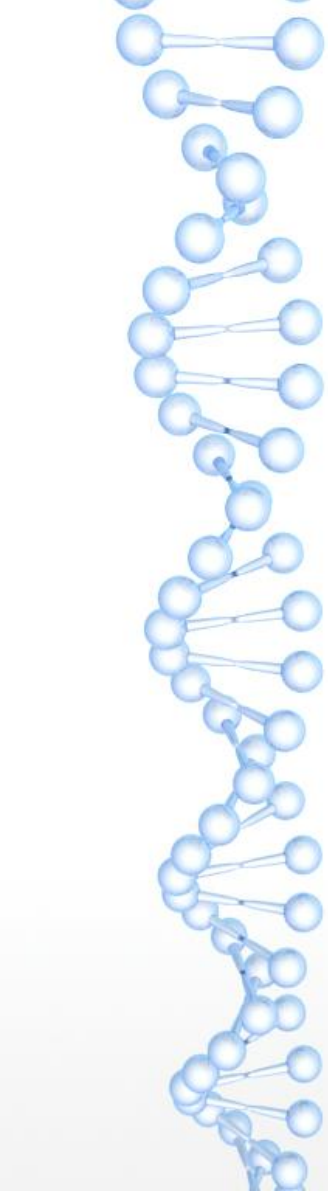
```
> isFour " 4 ";;
val it : bool = true
> isFour "Not a valid Integer";;
System.FormatException: Input string was not in a correct format.
    at System.Number.StringToNumber(String str, NumberStyles options,
    NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
    at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
    at FSI_0007.IsFour(String str)
    at <StartupCode$FSI_0009>.$FSI_0009._main()
stopped due to error
```

В языке F# можно определить *частичные активные шаблоны*, которые не всегда выполняют преобразование входных данных. Для этого частичный активный шаблон должен возвращать экземпляр типа `option`. (Который, как вы помните, может иметь только два значения, `Some('a)` и `None`.) Если сопоставление завершилось успешно, возвращается значение `Some`, в противном случае – `None`. При связывании результата активного шаблона с новым значением производится обратное преобразование типа `option` в фактический тип результата.



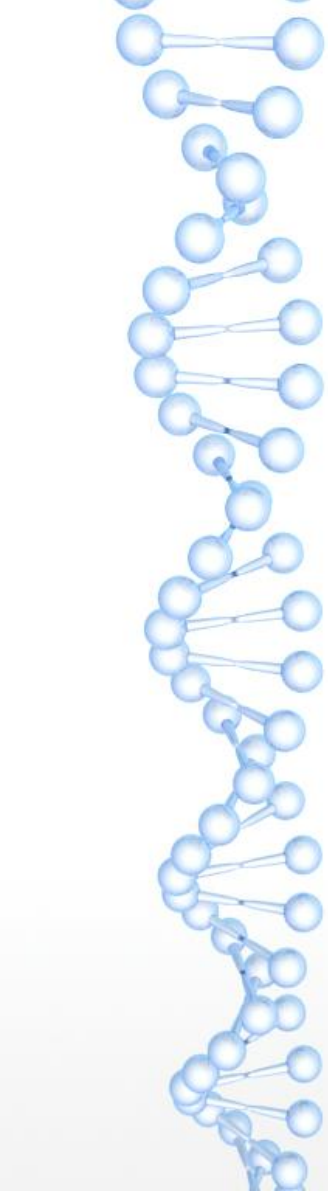
Частичные активные шаблоны

```
open System
let (|ToBool|_|) x =
    let success, result = Boolean.TryParse(x)
    if success then Some(result)
    else            None

let (|ToInt|_|) x =
    let success, result = Int32.TryParse(x)
    if success then Some(result)
    else            None

let (|ToFloat|_|) x =
    let success, result = Double.TryParse(x)
    if success then Some(result)
    else            None

let describeString str =
    match str with
    | ToBool  b -> printfn "%s is a bool with value %b" str b
    | ToInt   i -> printfn "%s is an integer with value %d" str i
    | ToFloat f -> printfn "%s is a float with value %f" str f
    | _         -> printfn "%s is not a bool, int, or float" str
```



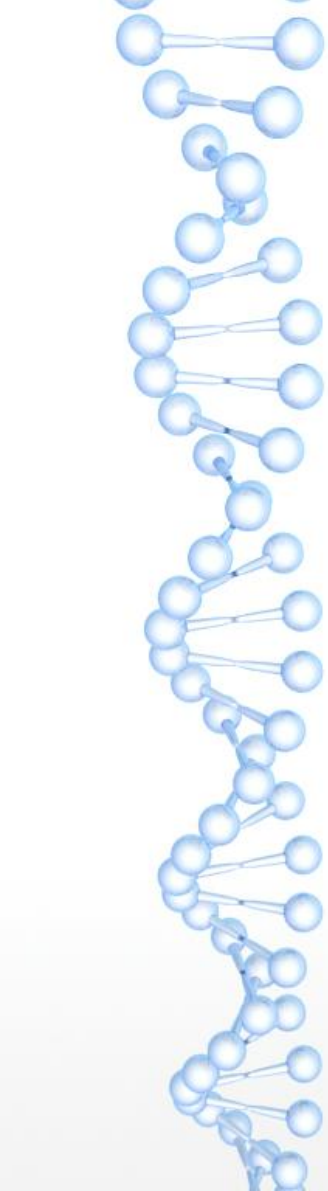
Частичные активные шаблоны

```
> describeString “ 3.141 “;;
3.141 is a float with value 3.141000
val it : unit = ()
> describeString “Not a valid integer”;;
Not a valid integer is not a bool, int, or float
val it : unit = ()
```



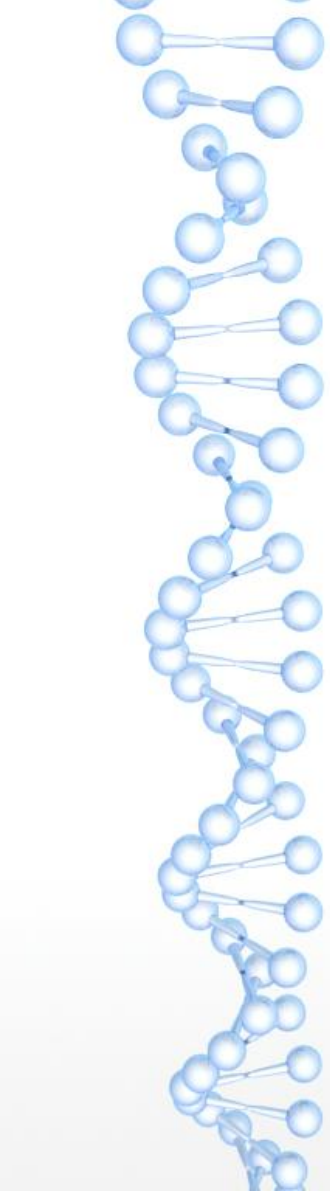
Параметризованные активные шаблоны

```
open System.Text.RegularExpressions

// Использование регулярного выражения
// для получения совпадений с тремя группами
let (|RegexMatch3|_|) (pattern : string) (input : string) =
    let result = Regex.Match(input, pattern)

    if result.Success then
        match (List.tail [ for g in result.Groups -> g.Value ]) with
        | fst :: snd :: trd :: []
            -> Some (fst, snd, trd)
        | [] -> failwith <| "Match succeeded, but no groups found.\n" +
                            "Use '(.*)' to capture groups"
        | _ -> failwith "Match succeeded, but did not find exactly three
groups."
    else
        None

let parseTime input =
    match input with
    // Проверить, имеет ли строка input вид "6/20/2008"
    | RegexMatch3 "(\d+)/(\d+)/(\d\d\d\d)" (month, day, year)
    // Проверить, имеет ли строка input вид "2004-12-8"
    | RegexMatch3 "(\d\d\d\d)-(\d+)-(\d+)" (year, month, day)
        -> Some( new DateTime(int year, int month, int day) )
    | _ -> None
```



Параметризованные активные шаблоны

```
> parseTime "1996-3-15";;
val it : DateTime option
= Some 3/15/1996 12:00:00 AM {Date = ...;
                              Day = 15;
                              DayOfWeek = Friday;
                              DayOfYear = 75;
                              Hour = 0;
                              Kind = Unspecified;
                              Millisecond = 0;
                              Minute = 0;
                              Month = 3;
                              Second = 0;
                              Ticks = 629624448000000000L;
                              TimeOfDay = 00:00:00;
                              Year = 1996;}
> parseTime "invalid";;
val it : DateTime option = None
```



Многовариантные активные шаблоны

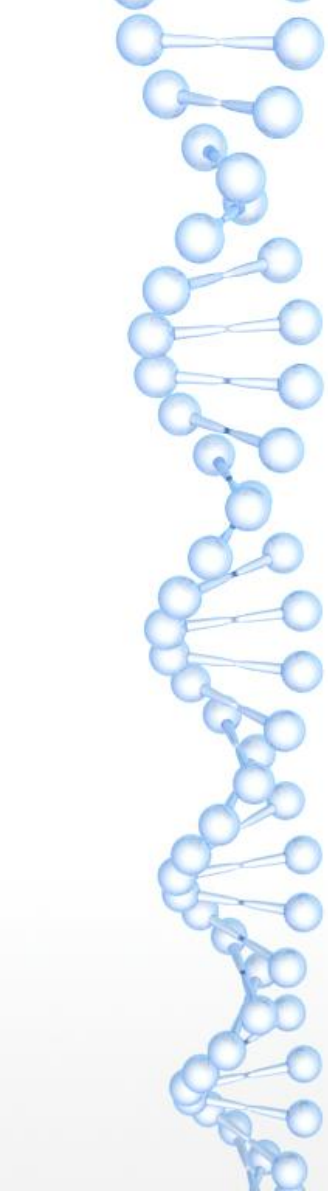
Одновариантные активные шаблоны удобно использовать для преобразования данных, но они имеют один недостаток – одновариантные шаблоны могут преобразовать входные данные только в какой-то один новый формат. Но иногда бывает необходимо преобразовать данные сразу в несколько значений различных типов, например, когда желательно разбить пространство входных данных на несколько категорий.

С помощью *многовариантных активных шаблонов* можно разделить пространство входных данных на множество возможных значений. Чтобы определить многовариантный активный шаблон, достаточно использовать ту же конструкцию со скобками (| |), но при этом нужно указать несколько идентификаторов, обозначающих категории на выходе.

Например, рассмотрим реализацию сопоставления записи из таблицы `Customers`, находящейся в базе данных. Можно было бы написать несколько частичных активных шаблонов и с их помощью отнести клиента к одной из категорий: «опытный пользователь», «непользователь», «ценный клиент» и так далее. Однако вместо того чтобы создавать множество частичных активных шаблонов для определения категории входных данных, можно создать один многовариантный активный шаблон, который поделит пространство входных данных в точности на указанное количество групп возможных результатов.

Многовариантные активные шаблоны

```
open System

// Этот активный шаблон делит все строки на четыре категории.
let (|Paragraph|Sentence|Word|WhiteSpace|) (input : string) =
    let input = input.Trim()

    if input = "" then
        WhiteSpace
    elif input.IndexOf(".") <> -1 then
        // Для категории Paragraph должен возвращаться кортеж
        // со списком предложений и его размером.
        let sentences = input.Split([|"."|], StringSplitOptions.None)
        Paragraph (sentences.Length, sentences)
    elif input.IndexOf(" ") <> -1 then
        // Для категории Sentence должен возвращаться массив слов
        Sentence (input.Split([|" "|], StringSplitOptions.None))
    else
        // Для категории Word должна возвращаться строка
        Word (input)

// Подсчет количества букв в строке
let rec countLetters str =
    match str with
    | WhiteSpace -> 0
    | Word x -> x.Length
    | Sentence words
        -> words
            |> Array.map countLetters
            |> Array.sum
    | Paragraph (_, sentences)
        -> sentences
            |> Array.map countLetters
            |> Array.sum
```

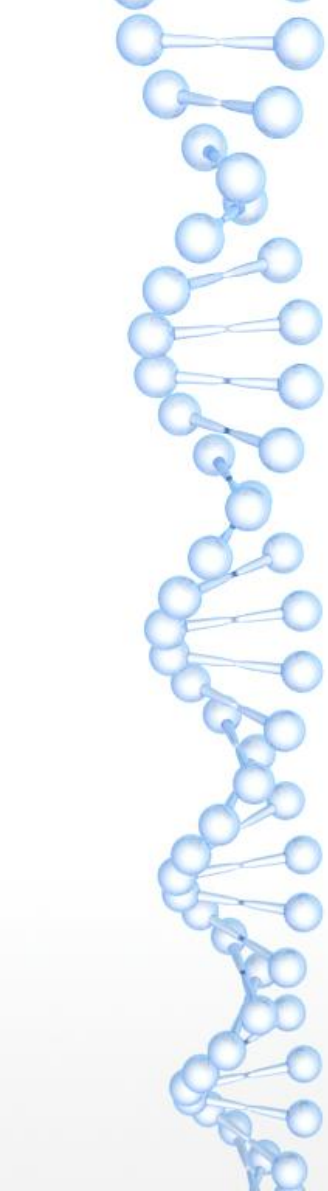


Использование активных шаблонов

Активные шаблоны могут использоваться не только в выражениях сопоставления, как мы видели это до сих пор. Активные шаблоны могут объединяться, вкладываться друг в друга и использоваться за пределами выражений сопоставления, как любые другие правила сопоставления с образцом.

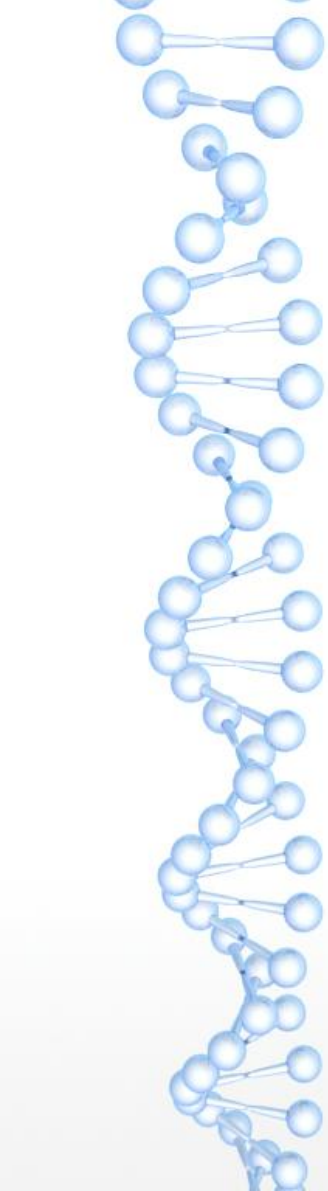
Активные шаблоны могут использоваться вместо любого элемента, выполняющего сопоставление, которые, как вы наверняка помните, могут использоваться в операторах связывания `let` или в качестве параметров лямбда-выражений.

В следующем фрагменте демонстрируется применение активного шаблона `ToUpper` к первому параметру функции `f`:



Использование активных шаблонов

```
> let (|ToUpper|) (input : string) = input.ToUpper();;

val ( |ToUpper| ) : string -> string

> let f ( ToUpper x ) = printfn "x = %s" x;;

val f : string -> unit

> f "this is lower case";;
x = THIS IS LOWER CASE
val it : unit = ()
```

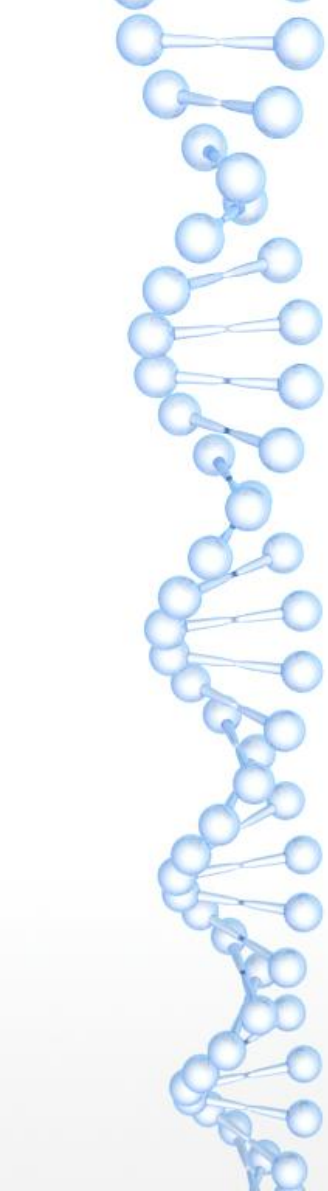


Использование активных шаблонов

```
// Классификация фильмов
let (|Action|Drama|Comedy|Documentary|Horror|Romance|) movie =
    // ...

// Название награды, которую получил фильм
let (|WonAward|_|) awardTitle movie =
    // ...

// Оценка фильма
let goodDateMovie movie =
    match movie with
    // Варианты сопоставления многовариантного активного шаблона
    | Romance
    | Comedy

    // Параметризованный активный шаблон
    | WonAward “Best Picture”
    | WonAward “Best Adaptation”
        -> true

    | _ -> false
```

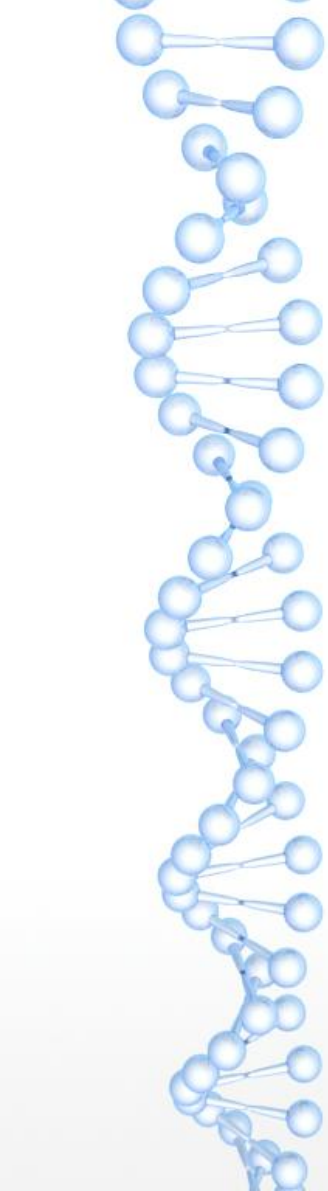
Использование активных шаблонов

```
open System.IO

let (|KBInSize|MBInSize|GBInSize|) filePath =
    let file = File.Open(filePath, FileMode.Open)
    if file.Length < 1024L * 1024L then
        KBInSize
    elif file.Length < 1024L * 1024L * 1024L then
        MBInSize
    else
        GBInSize

let (|IsImageFile|_|) filePath =
    match filePath with
    | EndsWithExtension ".jpg"
    | EndsWithExtension ".bmp"
    | EndsWithExtension ".gif"
        -> Some()
    | _ -> None

let ImageTooBigForEmail filePath =
    match filePath with
    | IsImageFile & (MBInSize | GBInSize)
        -> true
    | _ -> false
```



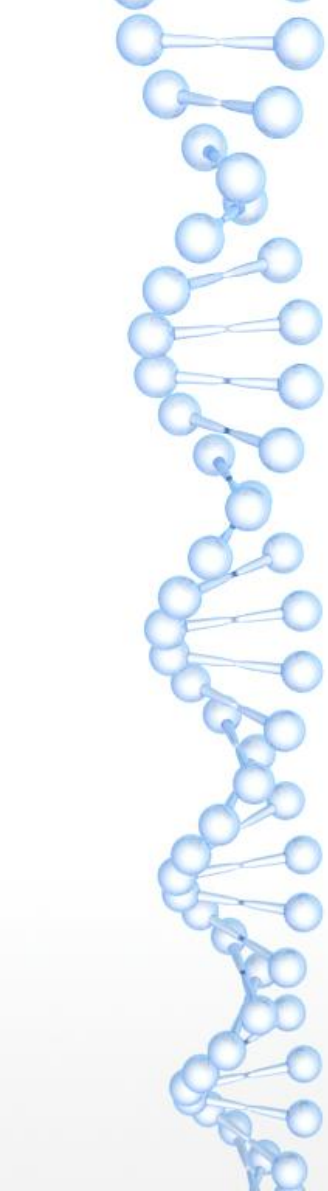
Вложенные активные шаблоны

```
// Для этого примера необходима ссылка на System.Xml.dll
#r "System.Xml.dll"
open System.Xml

// Сопоставление с XML элементом
let (|Elem|_|) name (inp : XmlNode) =
    if inp.Name = name then Some(inp)
    else                    None

// Извлечение атрибутов элемента
let (|Attributes|) (inp : XmlNode) = inp.Attributes

// Сопоставление с именем атрибута
let (|Attr|) attrName (inp : XmlAttributeCollection) =
    match inp.GetNamedItem(attrName) with
    | null -> failwithf "Attribute %s not found" attrName
    | attr -> attr.Value

// Анализируемые элементы
type Part =
    | Widget   of float
    | Sprocket of string * int

let ParseXmlNode element =
    match element with
    // Анализ элемента Widget без использования вложенных активных шаблонов
    | Elem "Widget" xmlElement
        -> match xmlElement with
           | Attributes xmlElementsAttributes
               -> match xmlElementsAttributes with
                  | Attr "Diameter" diameter
                      -> Widget(float diameter)

// Анализ элемента Sprocket с использованием вложенных активных шаблонов
| Elem "Sprocket" (Attributes (Attr "Model" model & Attr "SerialNumber" sn))
    -> Sprocket(model, int sn)

|_ -> failwith "Unknown element"
```



Вложенные активные шаблоны

```
> // Исходный документ XML
let xmlDoc =
    let doc = new System.Xml.XmlDocument()
    let xmlText =
        "<?xml version=\"1.0\" encoding=\"utf-8\"?>
        <Parts>
            <Widget Diameter='5.0' />
            <Sprocket Model='A' SerialNumber='147' />
            <Sprocket Model='B' SerialNumber='302' />

         </Parts>
         "
     doc.LoadXml(xmlText)
     doc;;

 val xmlDoc : XmlDocument

 > // Анализ узлов документа
 xmlDoc.DocumentElement.ChildNodes
 |> Seq.cast<XmlElement>
 |> Seq.map ParseXmlNode;;
 val it : seq<Part> = seq [Widget 5.0; Sprocket ("A",1024); Sprocket ("B",306)]
```



Вложенные активные шаблоны