University of Kent

Functional patterns

University of Kent

Lists

Every list is either empty `[]` … or non-empty.

In the non-empty case it matches the pattern `[X|Xs]`,

… `X` matches the first element, and

… `Xs` matches the rest of the list.

University of Kent

Lists

Take the example of `[2,3,4,5]` ...

It matches the pattern `[X|Xs]` ...

... `X` matches the first element, `2`, and

... `Xs` matches the rest of the list, `[3,4,5]`.

University of Kent

Template for recursion over lists

To define a function **foo** by recursion over lists, try this template …

```
foo([])     -> … ;
foo([X|Xs]) -> … foo(Xs) ….
```

University of Kent

Sum a list of numbers

Give the sum of the elements in a list of numbers.

```
sum([])     -> 0 ;
sum([X|Xs]) -> X + sum(Xs).
```

University of Kent

Find the area of all shapes in a list

Find the area of each shape in a list of shapes.

```
all_areas([])     -> [] ;
all_areas([X|Xs]) -> [ area(X) | all_areas(Xs) ].
```

University of Kent

Find the circles in a list of shapes

Extract the circles from a list of shapes.

```
circles([])      -> [] ;

circles( [{circle,{X,Y},R} | Xs] ) ->
      [ {circle,{X,Y},R} | circles(Xs) ];

circles( [{rectangle,{_,_},_,_} | Xs] ) ->
      circles(Xs).
```

University of Kent

Patterns ...

Combine all the elements of a list into on …

… e.g. using `+` to give the sum of list.

Transform all the elements of a list …

… e.g. using `area` to give the area of each of the shapes.

Select the elements of a list with a particular property …
… e.g. select the circles from a list of the shapes.

University of Kent

Patterns …

Reduce (or fold) all the elements of a list into one …

… e.g. using `+` to give the sum of list.

Map all the elements of a list …

… e.g. using `area` to give the area of each of the shapes.

Filter all the elements of a list with a particular property …

… e.g. select the circles from a list of the shapes.


University of
Kent

University of Kent

From specific to generic

University of Kent

Patterns …

Combine all the elements of a list into one …

… e.g. using `+` to give the sum of list.

Transform all the elements of a list …

… e.g. using `area` to give the area of each of the shapes.

Select the elements of a list with a particular property …
… e.g. select the circles from a list of the shapes.

University of Kent

Patterns …

Reduce (or fold) all the elements of a list into one …

… e.g. using `+` to give the sum of list.

Map all the elements of a list …

… e.g. using `area` to give the area of each of the shapes.

Filter all the elements of a list with a particular property …

… e.g. select the circles from a list of the shapes.

University of Kent

Apply `F` to every element of a list

Find the area of all … the only specific thing is applying `area`

```
all_areas([])     -> [] ;
all_areas([X|Xs]) -> [ area(X) | all_areas(Xs) ].
```

University of Kent

Apply `F` to every element of a list

Find the area of all … the only specific thing is applying `area`

```
all_areas([])     -> [] ;
all_areas([X|Xs]) -> [ area(X) | all_areas(Xs) ].
```

Apply `F` instead of `area` … and add `F` as an argument:

```
map(F,[])     -> [] ;
map(F,[X|Xs]) -> [ F(X) | map(F, Xs) ].
```

Here's the final definition:

```
all_areas(Shapes) -> map(fun area/1, Shapes).
```

University of Kent

Patterns … why?

Map all the elements of a list …

… e.g. using `area` to give the area of each of the shapes.

We get reuse … define `map` once and reuse forever.

We get comprehensibility … when we see `map` we know what's going on – no need to understand a recursive definition.


University of
Kent

University of Kent

Find the circles in a list of shapes

Extract the circles from a list of shapes.

```
circles([])      -> [] ;

circles( [{circle,{X,Y},R} | Xs] ) ->
      [ {circle,{X,Y},R} | circles(Xs) ];

circles( [{rectangle,{_,_},_,_} | Xs] ) ->
      circles(Xs).
```

University of Kent

Find the circles in a list of shapes

What is specific here is the test of whether to add or not …
… which is done here by pattern matching.

```
circles([])      -> [] ;

circles( [{circle,{X,Y},R} | Xs] ) ->
      [ {circle,{X,Y},R} | circles(Xs) ];

circles( [{rectangle,{_,_},_,_} | Xs] ) ->
      circles(Xs).
```

University of Kent

Filter all the elements with property P

Include X in the list if it has the property P … i.e. P(X) is true.

```
filter(P,[]) ->      [];

filter(P,[X|Xs]) ->
    case P(X) of
      true  -> [X|filter(P,Xs)];
      false ->    filter(P,Xs)
    end.
```

University of Kent

Is this a circle?

We used pattern matching in our original definition …

```
circles([])      -> [] ;

circles( [{circle,{X,Y},R} | Xs] ) ->
      [ {circle,{X,Y},R} | circles(Xs) ];

circles( [{rectangle,{_,_},_,_} | Xs] ) ->
      circles(Xs).
```

… "factor this out" as a test like this:

```
is_circle( {circle,{_,_},_} ) ->
      true;

is_circle( {rectangle,{_,_},_,_} ) ->
      false.
```

Find the circles in a list of shapes

So finally we can define **`circles`** using **`filter`** …

```
circles(Shapes) ->
    filter(fun is_circle/1,Shapes).
```

… using this property (= function that returns a boolean):

```
is_circle( {circle,{_,_},R} ) ->
      true;

is_circle( {rectangle,{_,_},_,_} ) ->
      false.
```

University of Kent

The sum of a list of numbers

What's specific in the definition of **sum**?

```
sum([])     -> 0;
sum([X|Xs]) -> X + sum(Xs)
```

... the starting value of 0, and
... the combining function +.

```
reduce(Combine, Start, []) ->
  Start;
reduce(Combine, Start, [X|Xs]) ->
  Combine(X, reduce(Combine, Start, Xs)).


sum(Xs) -> reduce(fun plus/2, 0, Xs).

plus(X,Y) -> X+Y.
```

University of Kent

The sum of a list of numbers

What's specific in the definition of **sum**?

```
sum([])     -> 0;
sum([X|Xs]) -> X + sum(Xs)
```

... the starting value of 0, and
... the combining function +.

```
reduce(Combine, Start, []) ->
  Start;
reduce(Combine, Start, [X|Xs]) ->
  Combine(X, reduce(Combine, Start, Xs)).
```

```
sum(Xs) -> reduce(fun (X,Y) -> X+Y end,
                  0, Xs).
```

University of Kent

Using these higher-order functions

Process a collection of web pages … using `map`.

    For each word on the page generate value … using `map` again.

Select all values for a particular word … using `filter` (or `map`).

Combine that set of values to a single value … using `reduce`.

Available in a map-reduce framework like Hadoop.


University of
Kent

Terminal Shell Edit View Window Help

simonthompson — xterm — beam.smp

```
% erl
Erlang/OTP 17 [erts-6.3] [source-f9282c6] [64-bit] [smp:8:8] [a
sync-threads:10] [hipe] [kernel-poll:false]

Eshell V6.3  (abort with ^G)
1> c(z).
{ok,z}
2> zip_with(fun (X,Y) -> max(X,Y) end, [1,3,2,4], [0,5,4,3]).
** exception error: undefined shell command zip_with/3
3> z:zip_with(fun (X,Y) -> max(X,Y) end, [1,3,2,4], [0,5,4,3]).

[1,5,4,4]
4> z:zip_with(fun max/2, [1,3,2,4], [0,5,4,3]).
** exception error: undefined function erl_eval:max/2
5> z:zip_with(fun erlang:max/2, [1,3,2,4], [0,5,4,3]).
[1,5,4,4]
6> z:zip_with(fun (X,Y) -> {\X,Y} end, [1,3,2,4], [0,5,4,3]).
* 1: syntax error before: '\\'
6> z:zip_with(fun (X,Y) -> {X,Y} end, [1,3,2,4], [0,5,4,3]).
[{1,0},{3,5},{2,4},{4,3}]
7>
```

z.erl

New Open Recent Revert Save Print Undo Redo Cut Copy Paste Search Preferences Help

```
-module(z).
-export([zip_with/3]).

zip_with(_F,_,[]) ->
    [];
zip_with(_F,[],_) ->
    [];
zip_with(F,[X|Xs],[Y|Ys]) ->
    [ F(X,Y) | zip_with(F,Xs,Ys) ].
```

U:--- z.erl All (2,19) (Erlang EXT Flymake)


University of
Kent