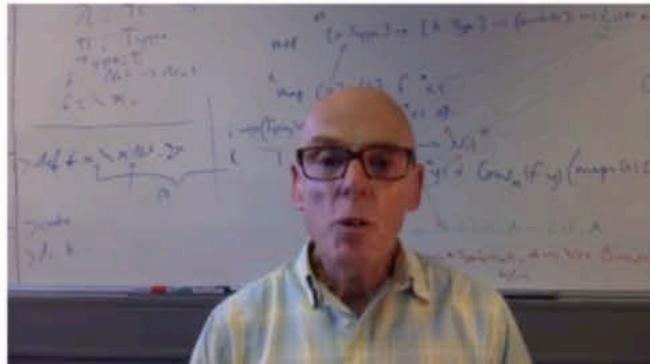

University of
Kent

University of Kent

Exceptions: `throw` and `catch`



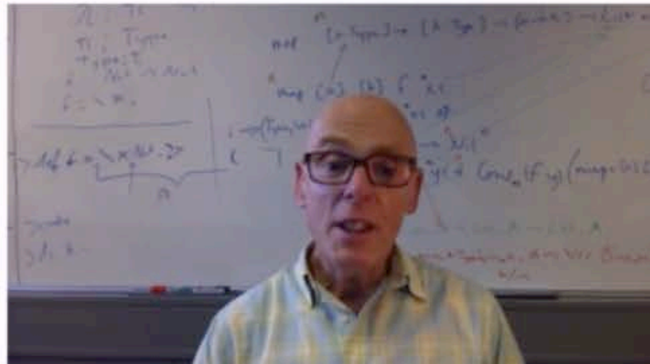
University of Kent

Internal and external errors

We've learned about the *"let it fail!"* philosophy for dealing with Erlang errors.

Crucial when the error is due to something outside the component we're in.

How can we handle errors that come from within (the part of) the system?



University of Kent

Tagged results: `ok` or `error`

Function returns an explicitly tagged return value.

The result is either

```
{ok,Value}
```

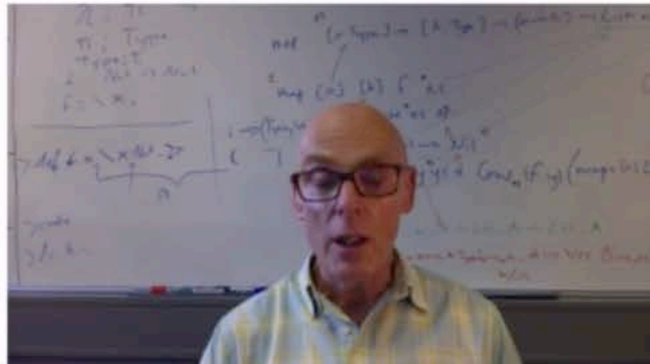
or

```
{error, Reason}
```

```
area(H, W)
  when H>0, W>0 ->
    {ok, H*W};

area(H, W) ->
    {error, negative_args}.
```

The error is manifest, and requires the client code to check.



University of Kent

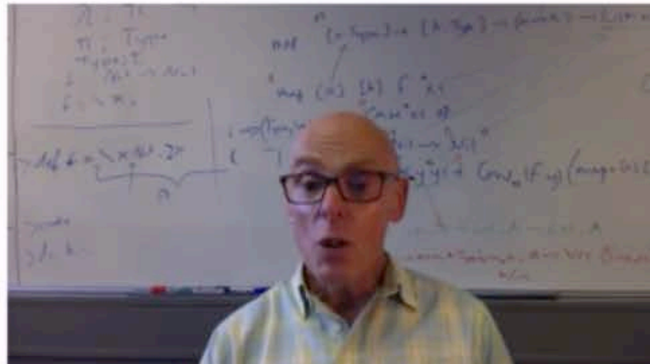
`eval` example

Handling the potential for division by zero.

We can tag the result, with an appropriate tag …

… but we have to change all the rest of the code too, to explicitly check whether recursive calls are tagged `ok` or `error`.

```
eval(Env,{div,Num,Denom}) ->
  N = eval(Env,Num),
  D = eval(Env,Denom),
  case D of
    0 ->
      {error, div_by_zero};
    _NZ ->
      {ok, N div D}
  end;
```



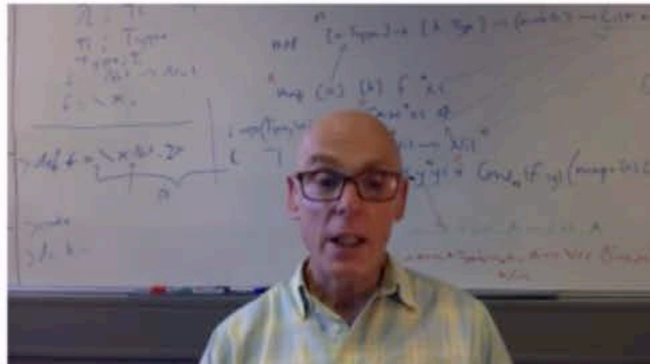
University of Kent

`eval` example

Instead, `throw` an exception on division by 0 …

…and just return the result in the normal case.

```
eval(Env,{div,Num,Denom}) ->
  N = eval(Env,Num),
  D = eval(Env,Denom),
  case D of
    0 ->
      throw(div_by_zero);
    _NZ ->
      N div D
  end;
```



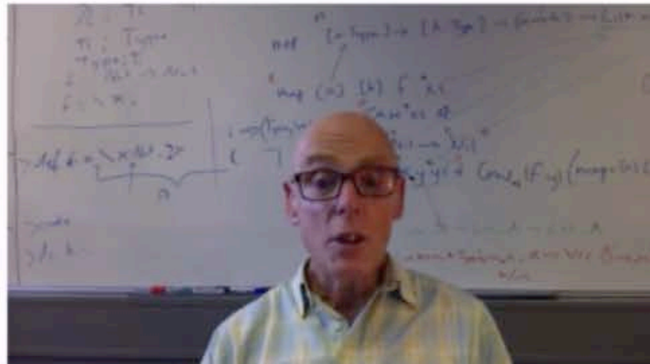
University of Kent

Catch exceptions with `try … catch …`

When we call `eval` we need to check whether it raises an exception …

…and deal with it accordingly.

```
try eval (Env,Exp) of
  Res ->
    {ok, Res}
catch
  throw:div_by_zero ->
    {error,div_by_zero}
end
```

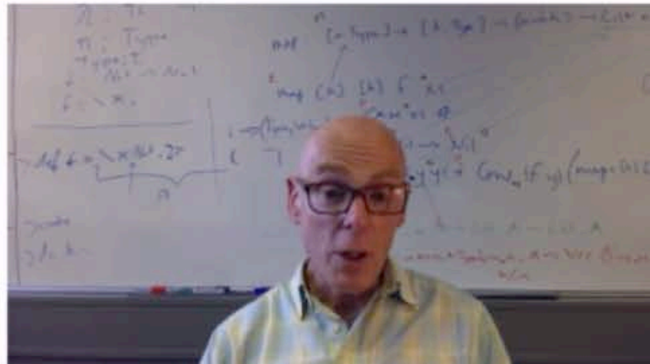


University of Kent

Catch exceptions with `try … catch …`

If it `throws` the `div_by_zero` exception, then we return a tuple saying so …

```
try eval (Env,Exp) of
  Res ->
    {ok, Res}
catch
  throw:div_by_zero ->
    {error,div_by_zero}
end
```

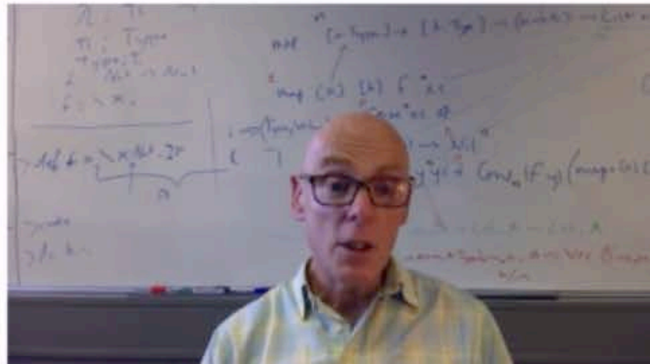


University of Kent

Catch exceptions with `try … catch …`

Alternatively, we could just return a number, giving the dummy value `0` on exception.

```
try eval (Env,Exp) of
  Res ->
    Res
catch
  throw:div_by_zero ->
    0
end
```



University of Kent

Exception kinds

We can catch three kinds of exception …

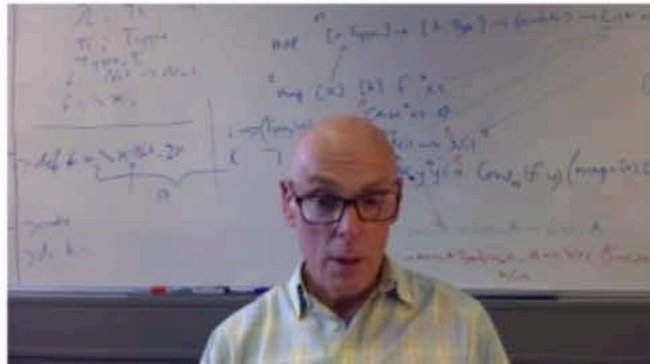
… ones that have been `thrown` explicitly

… `errors` raised by the execution environment,

… `exit` calls or exit signals.

We can also match particular exceptions, too.

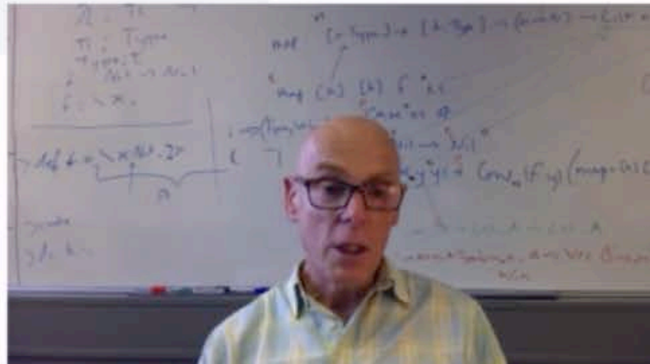
```
try eval (Env,Exp) of
    …
catch
  throw:_ -> …
     …
  error:_ -> …
    …
  exit:_  -> …
    …
end
```



University of Kent

Errors

`badmatch`	Pattern match error: often already bound variable
`badarg`	BIF called with arguments of the wrong type
`badarith`	Arithmetic error: e.g. divide by zero
`undef`	Function not defined: did you export it?
`function_clause`	No matching `function` clause
`if_clause`	No matching `if` clause
`case_clause`	No matching `case` clause




University of
Kent