Exceptions and Exception Handling

Exceptions are an important practical tool for controlling a program's flow. There are often situations where the normal execution needs to be stopped, for example trying to divide a number by 0, or trying to find something in an empty list. Some times you can represent that failure using option types, but it is often convenient to have another way.

Exception mechanisms typically consist of the following three components:

- A way to "raise" an exception at some point in the program.
- A way to set up a "handler" to respond to a risen exception.
- A way to create new types of exceptions.

Raising an exception

Raising an exception is syntactically simple. We have an expression `raise e` where `e` must be an "exception" and we "raise it". The interesting question is: what is the type of `raise e`?

If you think about it, it has to be any type, because it needs to type-check no matter where we place it in a program. This is often called the "bottom" type, it is a "subtype" of any other type.

The semantics of raising an exception is more interesting. The current execution is completed aborted, and control returns to some previous time where a handler was established. The handler is then allowed to continue the evaluation.

Handling an exception

The main construct for dealing with exceptions is the "try-with" block. It looks as follows:

```
try e with
| p1 -> e1
| p2 -> e2
| p3 -> e3
```

where e, e1, e2, e3 are expressions and p1, p2, p3 are patterns that match exceptions. The evaluation of this block proceeds as follows: `e` is evaluated, and if there are no raised exceptions its value is returned. If an exception is raised then it is compared to the patterns in order, until one matches, then the corresponding expression is executed. If none of the patterns matches, the raised exception is send to an earlier handler, if one was set up.

From a typechecking point of view, e, e1, e2, e3 must have the same type, and the patterns must agree with the exception type.

Creating new exceptions

The exception type, exn is a very special variant type. You can imagine it looking like this:

```
type exn = Not_found
         | Failure of string
         | Invalid_argument of string
         | ....
```

There are some system-provided exceptions, but the user/programmer and the various libraries add their own. This makes exn a very unusual type: We can add more variants to it at any given time. We do this with a syntax like:

```
exception OurOwn of t
exception DivisionByZero
```

In a raise e expression, the expression e must be of type exn.