30 points.

Please submit your solution using the **handin** program. Submit your solution as `cs418 mini1`.

Your submission should consist of one file called `mini1.erl`.

You can get a template for `mini1.erl` and a test module, `mini1_test.erl` [here](#).

The tests in `mini1_test.erl` are not exhaustive. If your code doesn’t work with these, it will almost certainly have problems with the test cases used for grading. The actual grading will include other test cases as well. I’ve included some notes on the test cases after the problem statements.

The Questions

1. **who_am_i/0**, 5 points.
   Write an Erlang function with no arguments that returns a tuple of two elements. Let `{Name, StudentNumber}` be this tuple, where:
   - (a) `Name` is an Erlang string – your name.
   - (b) `StudentNumber` is an Erlang integer – your student number.

2. **FindCats/1**, 10 points.
   Let `findCats(String) -> CatIndices` be a function where the argument `String` is a string, and the return value, `CatIndices` is a list of integers that give the starting positions of each occurrence of the string "cat" in `String`. The list `CatIndices` should be in ascending order. For example:

   ```erlang```
   mini1:findCats("My cat scattered cattle in Sascatchewan.").
   ```

   should return `[4,9,18,31]`

   Hint: this problem has a very simple solution using pattern-matching. My solution has `findCats(String)` call `findCats(String, Pos)` where `Pos` is the current position in the original string – for the call from `findCats(String), Pos=1.`

3. **rpn/1**, 15 points.
   For this problem you will write a simple, RPN calculator. The input is a list where each element is a number or an operator. The calculator maintains a stack that is initially empty. Processing each element from left to right as described below. Assume that the stack has at least two elements; let `X1` denote the top element; and let `X2` denote the next-to-top element on the stack.

   - `X` (when `X` is an integer or floating point number): pushes `X` onto the stack.
   - `'+':` pops `X1` and `X2` off the stack and then pushes their sum, `X2 + X1`, onto the stack.
   - `'-':` pops `X1` and `X2` off the stack and then pushes their difference, `X2 - X1`, onto the stack.
   - `'*':` pops `X1` and `X2` off the stack and then pushes their product, `X2 * X1`, onto the stack.
   - `'/':` pops `X1` and `X2` off the stack and then pushes their quotient, `X2 / X1`, onto the stack.
   - `'p':` Prints the top-of-stack element and leaves the stack unchanged.
   - `'P':` Prints the stack and leaves the stack unchanged.

   There are several error conditions such as stack underflow, divide-by-zero, or using an invalid operator. Because this is the first mini-assignment, we will only test your code on valid inputs. For subsequent
assignments, we’ll check to make sure that your code handles error conditions in a sensible way (and
take of points if it doesn’t).
The calculator function should be

\[ \text{rpn(InputList)} \rightarrow \text{Stack}. \]

where \text{Stack} is the stack produced after the operations in \text{InputList} have been performed. The \text{Stack}
should be represented as an Erlang list where the top-of-stack element is the first element of the list.
For example,

\[
\text{rpn([2, 3, '+'])} \rightarrow \text{[5]}. \\
\text{rpn([2, 3, '*', 4, 5, '*', '+'])} \rightarrow \text{[26]}. \\
\text{rpn([2, 3, '*', 4, 5, 'p', '*', '+'])} \rightarrow \text{[26]}. \text{ % and prints 5} \\
\text{rpn([2, 3, '*', 4, 5, 'P', '*', '+'])} \rightarrow \text{[26]}. \text{ % and prints [5, 4, 6]} \\
\text{rpn([2, 3, '*', 4, 5])} \rightarrow \text{[5, 4, 6]}. \\
\]

A few more remarks:

- I’m guessing that most of you have seen RPN, “reverse polish notation” before, but I don’t
believe it’s part of any official pre-requisite for this course. Wikipedia provides a fine description:
  \url{https://en.wikipedia.org/wiki/Reverse_Polish_notation}.
- For the ‘p’ and ‘P’ operators, Erlang’s \text{io:format} is similar to C’s \text{printf}. For example, if \text{X = 42},
  then

\[
\text{io:format("X = ~w\n", \[X\])}
\]

prints

\[
X = 42
\]

The control sequence \text{~w} means print a value in the “default” format for its type. The control
sequence \text{~n} prints a newline. If you want to print more than one value, use a \text{~w} for each one,
and replace \text{[X]} with the list of values you want printed. If you want more control over how the
value is printed, read the documentation for \text{io:format}.
- I often debug my code by just adding a few \text{io:format} expressions. For example, I’ll add an
\text{io:format} as the first expression in a function body to show how it’s called and print the values of
its arguments.
- After writing this problem, I realized there’s in RPN calculator as an example in \text{Learn You Some
Erlang}. No problem! The calculator in \text{Learn You Some Erlang} takes a string as an argument,
and returns a stack. It also has different operators. If you transform that code into a solution for
this assignment:
  - Give a clear citation of \text{Learn You Some Erlang} in your solution.
  - You’ll learn at least as much about Erlang coding by reading the example code in \text{Learn You
Some Erlang} and modifying it as you will by writing your own code from scratch. I’m happy
with either approach.

\textbf{EUnit tests}

The test cases are written using the \text{EUnit} package. To run them, in the Erlang shell, compile your code and
the tests and then run the tests:
1> c(mini1). % compile your code
2> c(mini1_test). % compile the EUnit tests
3> mini1_test:test(). % run the tests

You should get

All 3 tests passed.

If instead you get something like:

**error:{assertEqual,[{module,mini1_test},
  {line,22},
  {expression,"rpn ( [ 2 , 3 , ' * ' , 4 , 5 , p , ' * ' , '+' ] )"},
  {expected,[26]},
  {value,[25, 20, 5, 6, 3, 2]]}

That means that the test that is specified in file mini1_test.erl at line 22 evaluates the expression:

rpn([2, 3, '* ', 4, 5, p, '* ', '+ '])

According to the test description in mini1_test.erl, this should have produced the singleton list [26]. Running the code actually produced the list [25, 20, 5, 6, 3, 2], and EUnit is reporting the error – looks like the buggy code isn’t popping the operands off the stack before pushing the result.

Why?

**Question 1: who_am_i**
Give you a chance to write a really simple function – kind of like “hello world”.

**Question 2: findCats**
Get some practice with pattern matching.

**Question 3: rpn**
Write an “interesting” function.