HOW TO REGEX


STEP 1: OPEN YOUR FAVORITE EDITOR
@GARABATOKID
STEP 2: LET YOUR CAT PLAY ON YOUR KEYBOARD
/^([A-Z0-9_\.-

Q5: Shapeshifting Macros (Tutorial)

When writing macros in Scheme, we often create a list of symbols that evaluates to a desired Scheme expression. In this question, we'll practice different methods of creating such Scheme lists.

We have executed the following code to define `x` and `y` in our current environment.

```
(define x '(+ 1 1))
(define y '(+ 2 3))
```

We want to use `x` and `y` to build a list that represents the following expression:

```
(begin (+ 1 1) (+ 2 3))
```

What would be the result of calling `eval` on a quoted version of the expression above?

```
(eval '(begin (+ 1 1) (+ 2 3)))
```

How would we construct the scheme list for the expression `(begin (+ 1 1) (+ 2 3))` using quasiquotation?

How would we construct this scheme list using the `list` special form?

How would we construct this scheme list using the `cons` special form?

Q6: Max Macro (Tutorial)

Define the macro `max`, which takes in two expressions `expr1` and `expr2` and returns the maximum of their values. If they have the same value, return the first expression. **For this question, it's okay if your solution evaluates `expr1` and `expr2` more than once.** As an extra challenge, think about how you could use the `let` special form to ensure that `expr1` and `expr2` are evaluated only once.

```
scm> (max 5 10)
10
scm> (max 12 12)
12
scm> (max 100 99)
100
```

First, try using quasiquotation to implement this macro procedure.

```
(define-macro (max expr1 expr2)
'YOUR-CODE-HERE
)
```

Now, try writing this macro using the `list` special form.

```
(define-macro (max expr1 expr2)
'YOUR-CODE-HERE
)
```

Q7: When Macro (Tutorial)

Using macros, let's make a new special form, `when`, that has the following structure:

```
(when <condition>
      (<expr1> <expr2> <expr3> ...))
```

If the condition is not false (a truthy expression), all the subsequent operands are evaluated in order and the value of the last expression is returned. Otherwise, the entire `when` expression evaluates to `okay`.

```
scm> (when (= 1 0) ((/ 1 0) 'error))
okay
scm> (when (= 1 1) ((print 6) (print 1) 'a))
6
1
a
```

```
(define-macro (when condition exprs)
'YOUR-CODE-HERE









)
```