Regular Expressions Review - Quiz 1

```
def search(pattern, text):
    """Return true if pattern appears anywhere in text
       Please fill in the match(          , text) below.
       For example, match(your_code_here, text)"""
    if pattern.startswith('^'):
        return match(pattern[1:], text) # fill this line
    else:
        return match('.*' + pattern, text) # fill this line

```

Regular Expressions Review - Quiz 2

```
def match(pattern, text):
    """
    Return True if pattern appears at the start of text

    For this quiz, please fill in the return values for:
        1) if pattern == '':
        2) elif pattern == '$':
    """

    if pattern == '':
        return True
    elif pattern == '$':
        return (text == '')
    # you can ignore the following elif and else conditions
    # We'll implement them in the next quiz
    elif len(pattern) > 1 and pattern[1] in '*?':
        return True
    else:
        return True
```

Regular Expressions Review - Quiz 3

```
    if pattern == '':
        return True
    elif pattern == '$':
        return (text == '')
    elif len(pattern) > 1 and pattern[1] in '*?':
        p, op, pat = pattern[0], pattern[1], pattern[2:]
        if op == '*':
            return match_star(p, pat, text)
        elif op == '?':
            if match1(p, text) and match(pat, text[1:]):
                return True
            else:
                return match(pat, text)
    else:
        return (match1(pattern[0], text) and
                match(pattern[1:], text[1:])) # fill in this line
```

Matchset

```
def matchset(pattern, text):
    "Match pattern at start of text; return a set of remainders of
    op, x, y = components(pattern)
    if 'lit' == op:
        return set([text[len(x):]]) if text.startswith(x) else nul
    elif 'seq' == op:
        return set(t2 for t1 in matchset(x, text) for t2 in matchs
    elif 'alt' == op:
        return matchset(x, text) | matchset(y, text)
    elif 'dot' == op:
        return set([text[1:]]) if text else null
    elif 'oneof' == op:
        return set([text[1:]]) if text.startswith(x) else null
    elif 'eol' == op:
        return set(['']) if text == '' else null
    elif 'star' == op:
        return (set([text]) |
                set(t2 for t1 in matchset(x, text)
                    for t2 in matchset(pattern, t1) if t1 != text)
    else:
```

Filling Out The Api

```
def lit(string):  return ('lit', string)
def seq(x, y):    return ('seq', x, y)
def alt(x, y):    return ('alt', x, y)
def star(x):      return ('star', x)
def plus(x):      return seq(x, star(x))
def opt(x):       return alt(lit(''), x) #opt(x) me
def oneof(chars): return ('oneof', tuple(chars))
dot = ('dot',)
eol = ('eol',)
```

Search and Match

```
def search(pattern, text):
    "Match pattern anywhere in text; return longest earliest mat
    for i in range(len(text)):
        m = match(pattern, text[i:])
        if m: ''
            return m

def match(pattern, text):
    "Match pattern against start of text; return longest match f
    remainders = matchset(pattern, text)
    if remainders:
        shortest = min(remainders, key=len)
        return text[:len(text)-len(shortest)]
```

Alt

```
def lit(s): return lambda text: set([text[len(s):]]) if text.start

def seq(x, y): return lambda text: set().union(*map(y, x(text)))

def alt(x, y): return lambda text: x(text) | y(ext)
```

Simple Compilers

```
def match(pattern, text):
    "Match pattern against start of text; return
    remainders = pattern(text)
    if remainders:
        shortest = min(remainders, key=len)
        return text[:len(text)-len(shortest)]
```

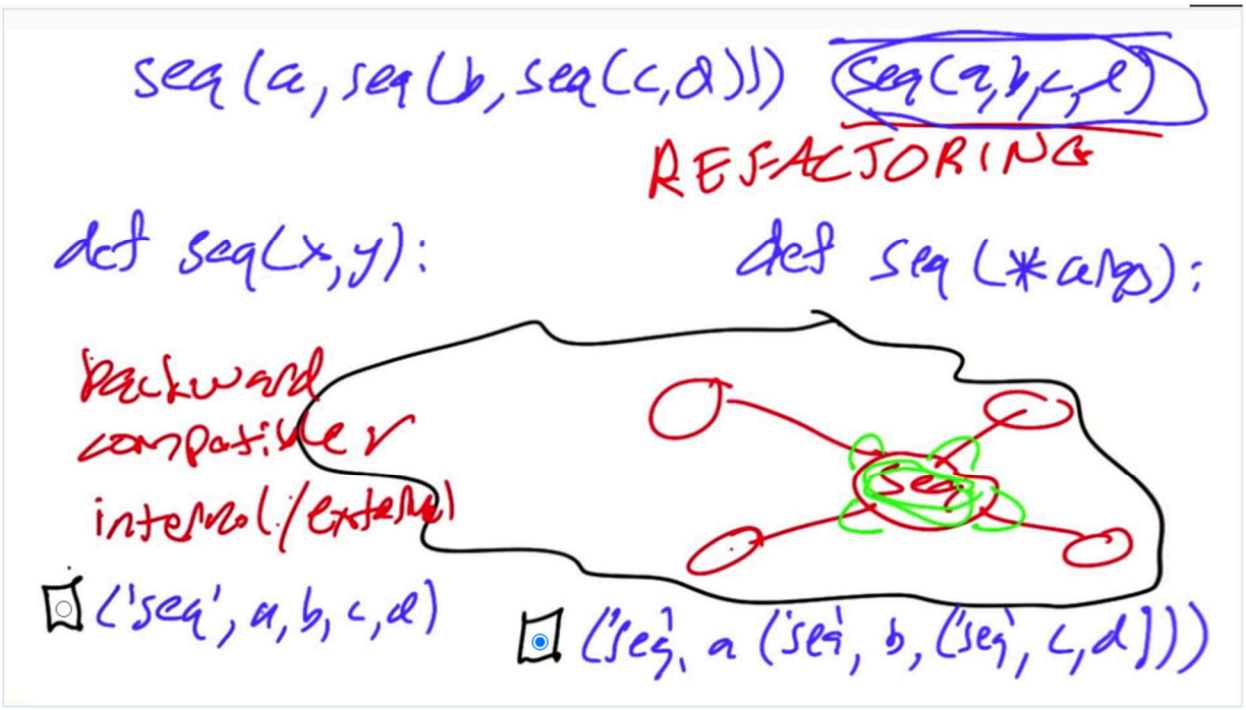
Oneof And Alt

```
def lit(s):         return lambda Ns: set([s]) if len(s) in Ns else null
def alt(x, y):      return lambda Ns: x(Ns) | y(Ns)
def star(x):        return lambda Ns: opt(plus(x))(Ns)
def plus(x):        return lambda Ns: genseq(x, star(x), Ns, startx=1) #Tricky
def oneof(chars):   return lambda Ns: set(chars) if 1 in Ns else null
def seq(x, y):      return lambda Ns: genseq(x, y, Ns)
def opt(x):         return alt(epsilon, x)
dot = oneof('?')    # You could expand the alphabet to more chars.
epsilon = lit('')   # The pattern that matches the empty string.

null = frozenset([])
```

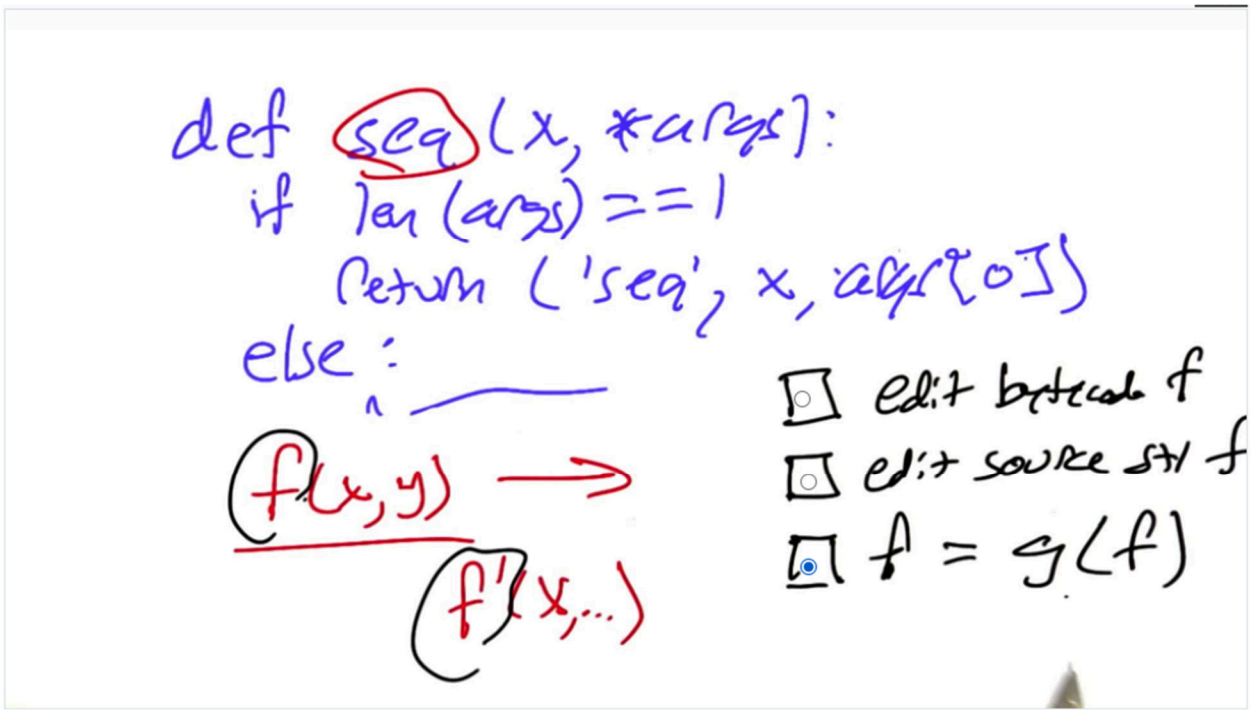
Genseq

```
3
4  def genseq(x, y, Ns):
5      Nss = range(max(Ns)+1)
6      return set(m1 + m2
7                 for m1 in x(Nss) for m2 in y(Nss)
8                 if len(m1 + m2) in Ns)
9
10
11         [○] correct for all inputs
12         [○] incorrect for some
13         [◉] correct when returns; doesn't always return
```

Changing Seq



Function Mapping



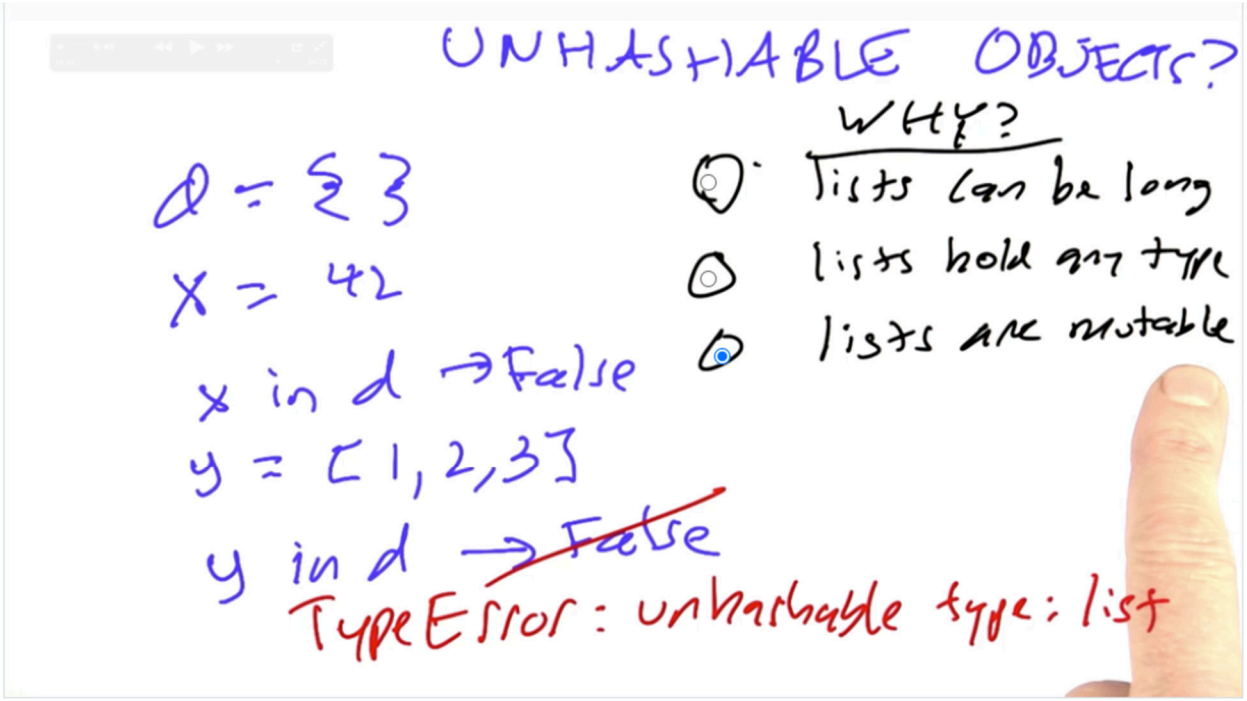
N Ary Function

```
def n_ary(f):
    """Given binary function f(x, y), return an n_ary function such
    that f(x, y, z) = f(x, f(y,z)), etc. Also allow f(x) = x."""
    def n_ary_f(x, *args):
        return x if not args else f(x, n_ary_f(*args))
    return n_ary_f
```

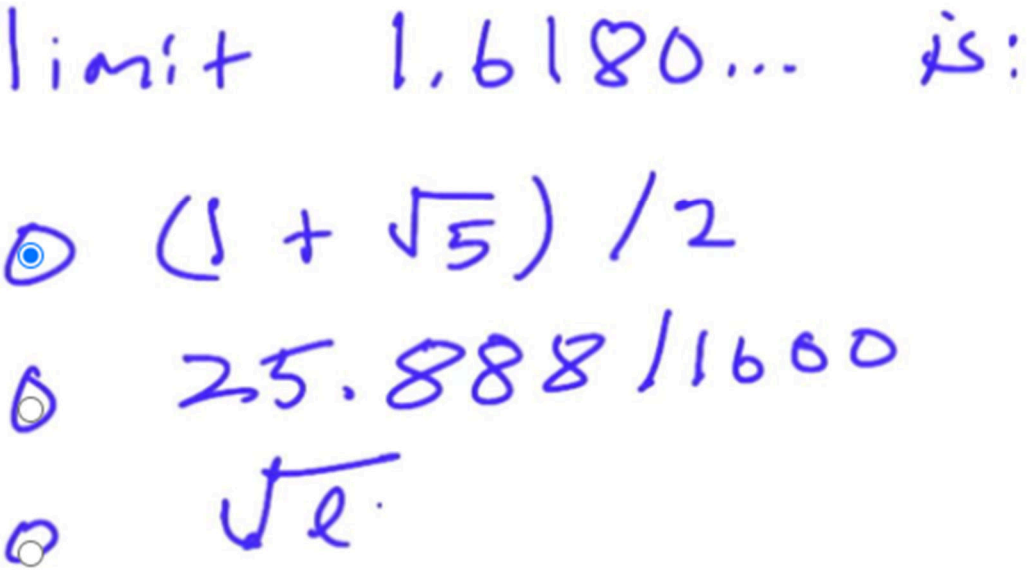
Decorated Decorators

```
def decorator(d):
    "Make function d a decorator: d wraps a function fn."
    def _d(fn):
        return update_wrapper(d(fn), fn)
    update_wrapper(_d, d)
    return _d

## QUIZ: DOES THIS WORK?

def decorator(d):
   "Make function d a decorator: d wraps a function fn. @author Darius Bacon"
   return lambda fn: update_wrapper(d(fn), fn)

decorator = decorator(decorator)

"""       (◉) Yes
          (○) No, results in an error
          (○) No, updates decorator (n_ary) but not decorated (seq)
          (○) No, my brain hurts
```

Cache Management



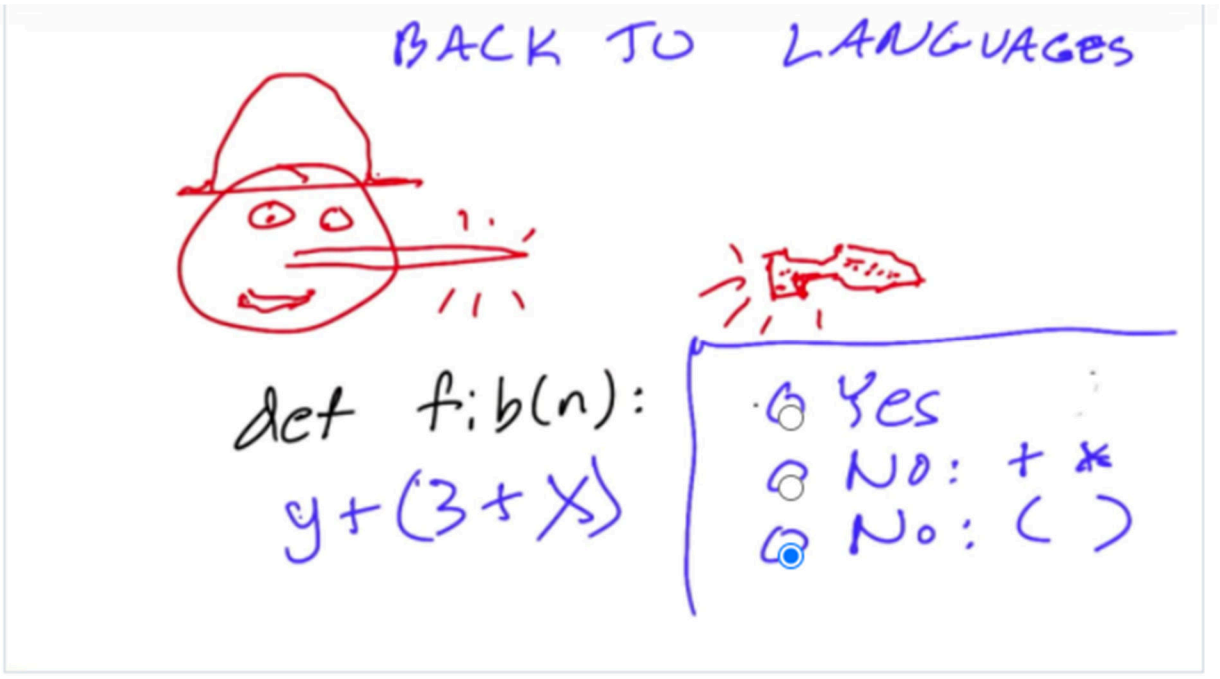
Save Time Now



Trace Tool

```
@decorator
def trace(f):
    indent = '    '
    def _f(*args):
        signature = '%s(%s)' % (f.__name__, ', '.join(map(repr, ar
        print '%s--> %s' % (trace.level*indent, signature)
        trace.level += 1
        try:
            result = f(*args)
            print '%s<-- %s == %s' % ((trace.level-1)*indent,
                                      signature, result)
        finally:
            trace.level -= 1
        return result
    trace.level = 0
    return _f
```

Back To Languages



Speedy Parsing