INSTRUCTIONS

- You have 2 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one hand-written 8.5" × 11" crib sheet of your own creation and the official 61A midterm 1 study guide attached to the back of this exam.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a brief explanation.

Last name

First name

SID

Login

TA & section time

Name of the person to your left

Name of the person to your right

All the work on this exam is my own. (please sign)

For staff use only

<table>
<thead>
<tr>
<th>Q. 1</th>
<th>Q. 2</th>
<th>Q. 3</th>
<th>Q. 4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>/12</td>
<td>/12</td>
<td>/14</td>
<td>/12</td>
<td>/50</td>
</tr>
</tbody>
</table>
1. (12 points) The Call Express is Delayed

For each of the following call expressions, write the value to which it evaluates and what would be output by the interactive Python interpreter. The first two rows have been provided as examples.

- In the Evaluates to column, write the value to which the expression evaluates. If evaluation causes an error, write Error.
- In the column labeled Interactive Output, write all output that would be displayed during an interactive session, after entering each call expression. This output may have multiple lines. Whenever the interpreter would report an error, write Error. You should include any lines displayed before an error.

Assume that you have started Python 3 and executed the following statements:

```python
from operator import add, mul
def square(x):
    return mul(x, x)
def delay(arg):
    print('delayed')
def g():
    return arg
return g
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Evaluates to</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>square(5)</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>1/0</td>
<td>Error</td>
<td>Error</td>
</tr>
<tr>
<td>print(square(4))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>square(square(print(2)))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>delay(square)(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add(delay(square)(),(2), 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>delay(delay)()()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. (12 points) Protect the Environment

(a) (6 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. You may not need to use all of the spaces or frames.

A complete answer will:
- Add all missing names, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.

```python
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)
horse(mask)
```

Global frame

```
Global frame

| horse   |
|         |
| mask    |

func horse(mask)

func \(\lambda\)(horse)

Return Value

```
(b) (6 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. You may not need to use all of the spaces or frames.

A complete answer will:

- Add all missing names, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.

```python
p, s, y = 1, 2, 3
def gang(p):
    nam = style(p)
    return (nam(4), 5)
def style(s):
    return lambda y: (p, s, y)
gang(3)
```
3. (14 points) Sequences

(a) (2 pt) Fill in the blanks so that the final call expression below evaluates to a tuple value.

```python
def pair(x):
    if x == 30:
        return lambda: (1, 2, 3)
    else:
        return lambda: 4

(lambda__________, soda: hall_____________)(pair, "sequence")
```

(b) (2 pt) Draw a box and pointer diagram for the following rlist:

```python
a = rlist(1, rlist(rlist(2, (3, empty_rlist)), rlist((4, 5, 6), empty_rlist)))
```

(c) (2 pt) What is the element at index 1 of this rlist, returned by `getitem_rlist(a, 1)`?

```python
def getitem_rlist(s, i):
    """Return the element at index i of recursive list s.""
    while i > 0:
        s, i = rest(s), i - 1
    return first(s)
```

(d) (2 pt) What is the length of this rlist, returned by `len_rlist(a)`?

```python
def len_rlist(s):
    """Return the length of recursive list s.""
    length = 0
    while s != empty_rlist:
        s, length = rest(s), length + 1
    return length
(e) (6 pt) When the int constructor is called on a float value, it “truncates toward zero,” meaning that it returns the largest integer less than any positive argument, or the least integer greater than any negative argument. For example:

```python
>>> int(2)
2
>>> int(2.7)
2
>>> int(-1.5)
-1
```

Assume that you have started Python 3 and executed the following statements:

```python
def alt(f, g, z):
    while g(z) > 0 and z != 5:
        f, g = g, f
        z = g(z)
    return z

def grow(x):
    return int((x * 3) / 2)

def shrink(x):
    return x - 2

def flip(x):
    return int(10 / (x-2))
```

For each of the following call expressions, write the value to which it evaluates. If evaluation causes an error, write ERROR. If evaluation would run forever, write FOREVER.

- alt(grow, shrink, 3)
- alt(grow, shrink, 4)
- alt(flip, shrink, 3)
4. (12 points) In Verse

<table>
<thead>
<tr>
<th>The inverse of some function $F$</th>
<th>There once was a rhyming device</th>
</tr>
</thead>
<tbody>
<tr>
<td>is a function of argument $X$</td>
<td>That was built to make any sound, twice,</td>
</tr>
<tr>
<td>that returns you the $Y$,</td>
<td>But used orthography</td>
</tr>
<tr>
<td>such that when you apply</td>
<td>And not phonology</td>
</tr>
<tr>
<td>$F$ to $Y$ you recover the $X$.</td>
<td>To decide if a rhyme would suffice.</td>
</tr>
</tbody>
</table>

An invertible function is a function that takes and returns a single numeric value, is differentiable, and never returns the same value for two different arguments. Some examples:

```python
def double(y):
    """Return twice the value of y."""
    return 2 * y

def cube(y):
    """Return y raised to the third power."""
    return pow(y, 3)

def pow2(y):
    """Return 2 raised to the power of y."""
    return pow(2, y)
```

(a) (4 pt) Implement a function `invert` that takes an invertible function argument and returns its inverse. You may call `find_root`, `newton_update`, `approx_deriv`, and/or `iter_improve`. You cannot use any assignment, conditional, while, or for statements.

```python
def invert(f):
    """Return the inverse of invertible function $f$.
    >>> halve = invert(double)
    >>> halve(12)
    6.0
    >>> cube_root = invert(cube)
    >>> cube_root(27)
    3.0
    >>> log2 = invert(pow2)
    >>> log2(32)
    5.0
    """
```
A sight rhyme is a pair of words that do not rhyme, but have the same endings, such as *device* and *office*. Two numbers that end in the same digit can be sight rhymes. For example:

- (13, 53) are pronounced *thirteen* and *fifty-three*, despite both ending with the same one's digit 3.
- (0, 30) are pronounced *zero* and *thirty*, despite both ending with the same one's digit 0.

(b) *(4 pt)* A numpair is a pair of integers that have the same one's digit. Fill in the two missing expressions in the constructor below, which takes two non-negative integers less than 100, asserts that they have the same one's digit, and returns a numpair represented as a pair of tens digits and the shared one's digit.

```python
from operator import floordiv, mod # Use these functions or // and %

def numpair(first, second):
    """Return a numpair as a pair of ten’s digits and a shared one’s digit."

    >>> numpair(23, 53)
    ((2, 5), 3)
    >>> numpair(67, 7)
    ((6, 0), 7)

    assert __________________________ , "different one’s"
    return ________________________________
```

(c) *(4 pt)* Fill in four missing expressions below so that *sight_rhyme* returns whether the numbers in a numpair p do not end with the same sound when pronounced. Your implementation cannot depend on the representation of a numpair; use selector functions. You cannot use the boolean operators and or or.

```python
def ones(p):
    return p[1]

def first_tens(p):
    return p[0][0]

def second_tens(p):
    return p[0][1]

def sight_rhyme(p):
    """Return whether the two numbers in a numpair do not rhyme."

    >>> sight_rhyme(numpair(13, 53))
    True
    >>> sight_rhyme(numpair(0, 30))
    True
    >>> sight_rhyme(numpair(53, 23))
    False
    """

    if _____________________________:
        return ________________________________
    elif ones(p) == 0:
        if first_tens(p) == 0:
            return ________________________________
        else:
            return ________________________________
    else:
        return False
```
Evaluation rule for and expressions:
1. Evaluate the left operand.
2. If the result is a true value, execute the right operand.
3. Otherwise, return the result of the left operand.

Evaluation rule for or expressions:
1. Evaluate the left operand.
2. If the result is a true value, return the result.
3. Otherwise, execute the right operand.

Execution rule for conditional statements:
1. Evaluate the header's expression.
2. If it is a true value, execute the suite, then skip the while.
3. Otherwise, skip to the next line.

Execution rule for assignment statements:
1. Evaluate the header's expression.
2. Create a new variable (name) with the same parent as the function that was applied.
3. Bind the arguments to the function's formal parameter names in that frame.
4. Execute the body of the function in the environment beginning at that frame.

Execution rule for def statements:
1. Create a new function value with the specified name, formal parameters, and function body.
2. Its parent is the first frame of the current environment.
3. Bind the name of the function to the function value in the first frame of the current environment.

Evaluation rule for call expressions:
1. Evaluate the header's expression.
2. Simultaneously bind the names on the left to those values, in the first frame of the current environment.
3. Apply the function that is the value of the operator subexpression to the arguments that are the values of the operand subexpressions.

Applying user-defined functions:
1. Create a new local frame with the same parent as the function that was applied.
2. Bind the arguments to the function's formal parameter names in that frame.
3. Execute the body of the function in the environment beginning at that frame.

Execution rule for nested def statements:
A function that takes a function as an argument value or returns a function as a return value.

Higher-order function:
A function that takes a function as an argument value or returns a function as a return value.

Function of a single argument (not called term):
A formal parameter that will be bound to a function argument when called.

The global environment: the environment with only the global frame.
When a frame or function has no label (parent=___) then its parent is always the global frame.

Nested def statements: functions defined within other functions bodies are bound to names in the local frame.

The cube function is passed as an argument value.

The function bound to term gets called here.
A function

with formal parameters x and y and body "return x + y"

Must be a single expression

```python
def square(x):
    return x * x
def add_three(x):
    return x + 3
def make_adder(n):
    return lambda x: x + n
def pair(x, y):
    return (x, y)
def square(x):
    return x * x
def add_three(x):
    return x + 3
```

A function that returns a function

```python
def make_adder(n):
    return lambda x: x + n
```

The name add_three is bound to a function

```python
def add_three(x):
    return x + 3
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

Every user-defined function has a parent frame

```python
def square(x):
    return x * x
```

The parent of a frame is the frame in which it was defined

```python
def square(x):
    return x * x
```

The second element of the pair is the rest of the list

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```

The parent of a function is the frame in which it was defined

```python
def square(x):
    return x * x
```