# Exam-Prep Section (Week of 9/3)

Problem types:

- What Would Python Do (Without Lists)
- Environment Diagram (Without Lists)
- Non-Recursive Fill-In (Without Lists)

Attendance: links.cs61a.org/482

# What Would Python Do (Without Lists) #1

**1. (12 points) Evaluators Gonna Evaluate**

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write "Error".

*Hint*: No answer requires more than 5 lines. (It's possible that all of them require even fewer.)

The first two rows have been provided as examples.

*Recall:* The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

Assume that you have started `python3` and executed the following statements:

```
def jazz(hands):
    if hands < out:
        return hands * 5
    else:
        return jazz(hands // 2) + 1

def twist(shout, it, out=7):
    while shout:
        shout, out = it(shout), print(shout, out)
    return lambda out: print(shout, out)

hands, out = 2, 3
```

| Expression | Interactive Output |
|---|---|
| `pow(2, 3)` | 8 |
| `print(4, 5) + 1` | 4 5<br>Error |
| `print(None, print(None))` | |
| `jazz(5)` | |
| `(lambda out: jazz(8))(9)` | |
| `twist(2, lambda x: x-2)(4)` | |
| `twist(5, print)(out)` | |
| `twist(6, lambda hands: hands-out, 2)(-1)` | |

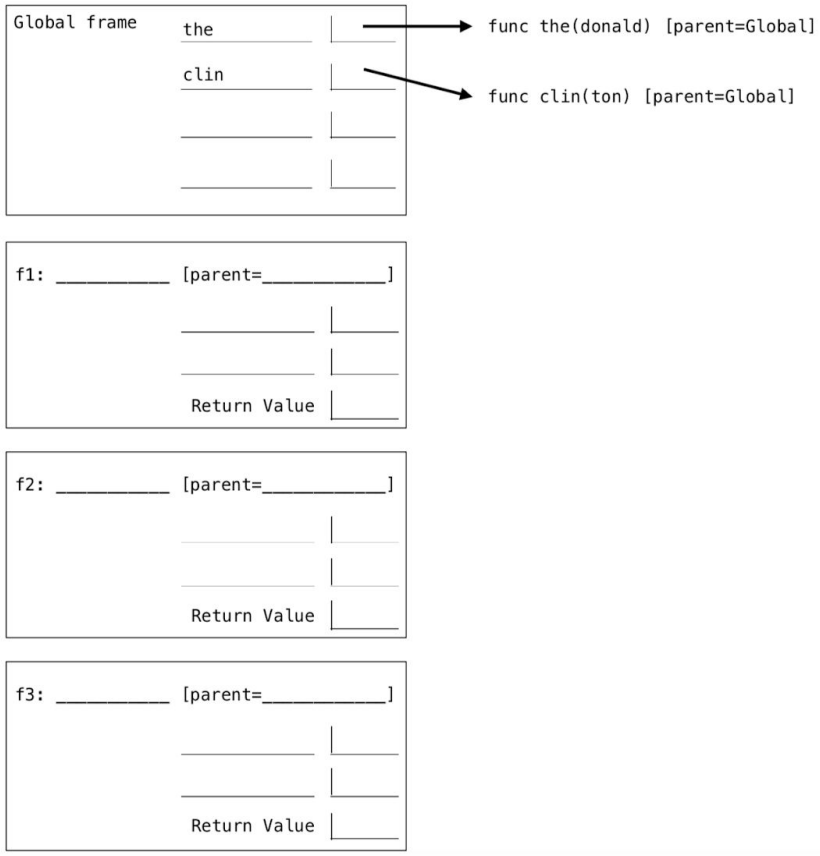
Fall 2015 Midterm 1: #1

# Environment Diagram (Without Lists) #1

**(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 and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
1  def the(donald):
2      return donald + 5
3
4  def clin(ton):
5      def the(race):
6          return donald + 6
7      def ton(ga):
8          donald = ga-1
9          return the(4)-3
10     return ton
11
12 donald, duck = 2, clin(the)
13 duck = duck(8)
```




Fall 2015 Midterm 1: Problem #2a

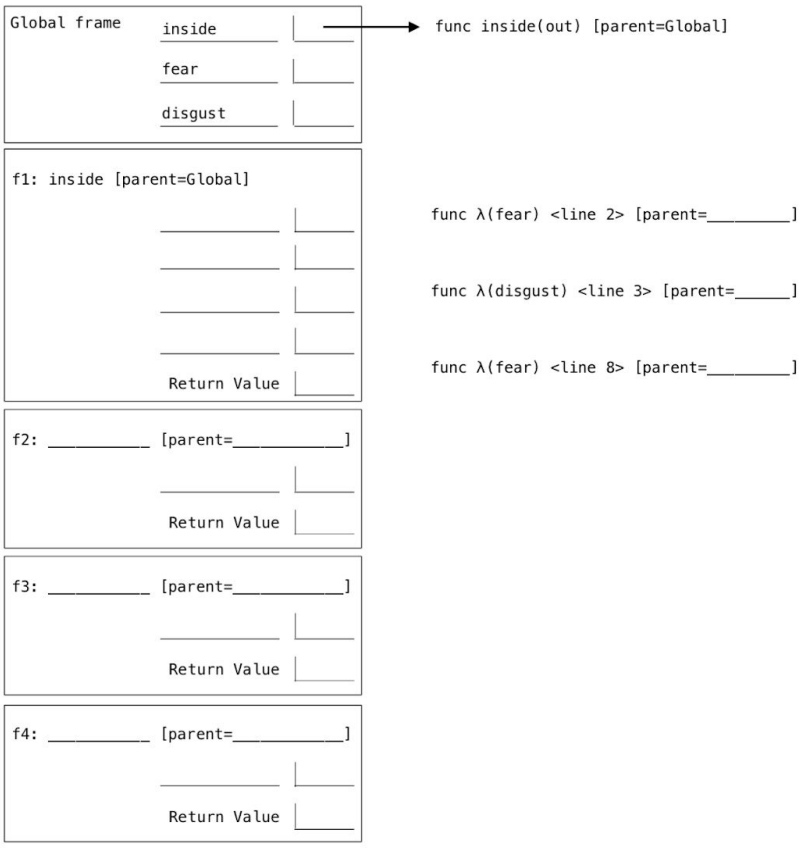
# Environment Diagram (Without Lists) #2

**(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.* The `<line ...>` annotation in a lambda value gives the line in the Python source of a lambda expression.

A complete answer will:

- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Add all missing parents of function values.
- Show the return value for each local frame.

```
1 def inside(out):
2     anger = lambda fear: fear(disgust)
3     fear = lambda disgust: anger(out)
4     disgust = 3
5     fear(5)
6
7 fear, disgust = 2, 4
8 inside(lambda fear: fear + disgust)
```




Fall 2015 Midterm 1: #2b

# Non-Recursive Fill-In (Without Lists) #1

**(a) (4 pt)** Implement the `longest_increasing_suffix` function, which returns the longest suffix (end) of a positive integer that consists of strictly increasing digits.

```
def longest_increasing_suffix(n):
    """Return the longest increasing suffix of a positive integer n.

    >>> longest_increasing_suffix(63134)
    134
    >>> longest_increasing_suffix(233)
    3
    >>> longest_increasing_suffix(5689)
    5689
    >>> longest_increasing_suffix(568901) # 01 is the suffix, displayed as 1
    1
    """

    m, suffix, k = 10, 0, 1

    while n:

        ______________________________________________, last = n // 10, n % 10


        if ____________________________________________________________________:


            m, suffix, k = _____________, ___________________________, 10 * k

        else:

            return suffix

    return suffix
```

Spring 2015 Midterm 1: #3a

# Non-Recursive Fill-In (Without Lists) #2

```
def sandwich(n):
    """Return True if n contains a sandwich and False otherwise

    >>> sandwich(416263)    # 626
    True
    >>> sandwich(5050)      # 505 or 050
    True
    >>> sandwich(4441)      # 444
    True
    >>> sandwich(1231)
    False
    >>> sandwich(55)
    False
    >>> sandwich(4456)
    False
    """

    tens, ones = ________________, ________________

    n = n // 100


    while ______________________________________:


        if ______________________________________:

            return True

        else:

            tens, ones = ________________, ________________


            n = _________________________________

    return False
```

Summer 2015 Midterm 1: #4c

# Non-Recursive Fill-In (Without Lists) #3

**(c) (3 pt)** Implement `luhn_sum`. The *Luhn sum* of a non-negative integer $n$ adds the sum of each digit in an *even* position to the sum of doubling each digit in an *odd* position. If doubling an odd digit results in a two-digit number, those two digits are summed to form a single digit. *You may not use recursive calls or call* ***`find_digit`*** *in your solution.*

```
def luhn_sum(n):
    """Return the Luhn sum of n.

    >>> luhn_sum(135)      # 1 + 6 + 5
    12
    >>> luhn_sum(185)      # 1 + (1+6) + 5
    13
    >>> luhn_sum(138743)   # From lecture: 2 + 3 + (1+6) + 7 + 8 + 3
    30
    """
    def luhn_digit(digit):

        x = digit * ____________________________________________________

        return (x // 10) + _____________________________________________

    total, multiplier = 0, 1

    while n:

        n, last = n // 10, n % 10

        total = total + luhn_digit(last)

        multiplier = ___________________________________ - multiplier

    return total
```

Fall 2015 Midterm 1: #3c