Fall 2015 Midterm 1: #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 Error
`print(None, print(None))`	None None None
`jazz(5)`	11
`(lambda out: jazz(8))(9)`	12
`twist(2, lambda x: x-2)(4)`	2 7 0 4
`twist(5, print)(out)`	5 5 7 None 3
`twist(6, lambda hands: hands-out, 2)(-1)`	6 2 3 None 0 -1

Fall 2015 Midterm 1: Problem #2a

2. (12 points) Environmental Policy

(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.*

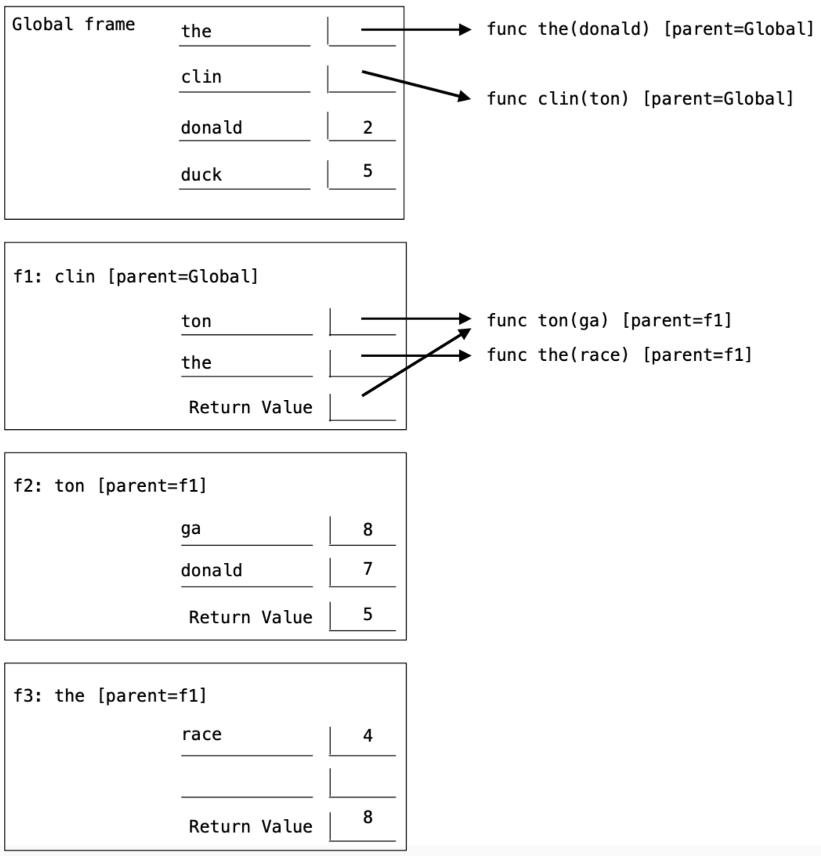
A complete answer will:

- 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.

```
def the(donald):
    return donald + 5

def clin(ton):
    def the(race):
        return donald + 6
    def ton(ga):
        donald = ga - 1
        return the(4) - 3
    return ton

donald, duck = 2, clin(the)
duck = duck(8)
```



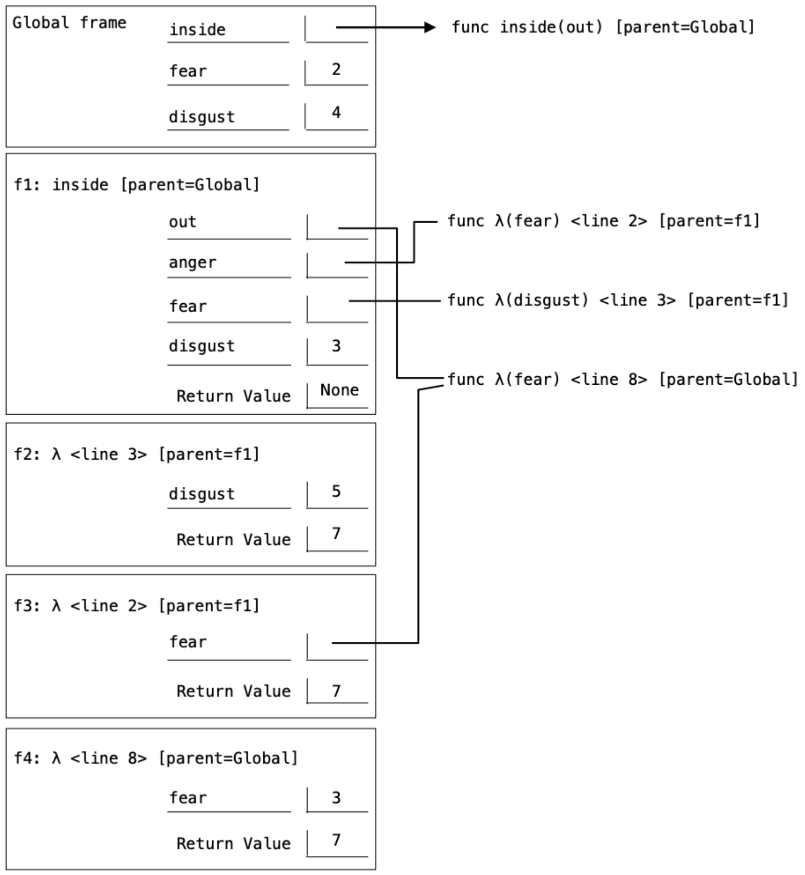
Fall 2015 Midterm 1: #2b

(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)
```



Global frame
inside
fear 2
disgust 4
func inside(out) [parent=Global]
f1: inside [parent=Global]
out
anger
fear
disgust 3
Return Value None
func λ(fear) <line 2> [parent=f1]
func λ(disgust) <line 3> [parent=f1]
func λ(fear) <line 8> [parent=Global]
f2: λ <line 3> [parent=f1]
disgust 5
Return Value 7
f3: λ <line 2> [parent=f1]
fear
Return Value 7
f4: λ <line 8> [parent=Global]
fear 3
Return Value 7

Spring 2015 Midterm 1: #3a

3. (14 points) You Complete Me

(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:

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


        if last < m:


            m, suffix, k = last, suffix + k * last, 10 * k

        else:

            return suffix

    return suffix
```

Summer 2015 Midterm 1: #4c

(c) **(4 pt)** A number `n` contains a *sandwich* if a digit in `n` is surrounded by two identical digits. For example, the number 242 contains a sandwich because 4 is surrounded by 2 on both sides. 1242 also contains a sandwich, while 12532 does not contain a sandwich.

Implement the `sandwich(n)` function, which takes in a nonnegative integer `n`. It returns `True` if `n` contains a sandwich and `False` otherwise. If `n` has fewer than three digits, it cannot contain a sandwich.

```
def sandwich(n):
    """Returns 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
    """
    a, b = (n // 10) % 10, n % 10

    n = n // 100

    while n > 0:

        if n % 10 == b:

            return True

        else:

            a, b = n % 10, a

            n = n // 10

    return False
```

Fall 2015 Midterm 1: #3c

(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 * multiplier

        return (x // 10) + x % 10

    total, multiplier = 0, 1

    while n:

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

        total = total + luhn_digit(last)

        multiplier = 3 - multiplier

    return total
```