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:

```python
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
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pow(2, 3)</code></td>
<td>8</td>
</tr>
<tr>
<td><code>print(4, 5) + 1</code></td>
<td>4 5</td>
</tr>
<tr>
<td><code>print(1)</code></td>
<td>Error</td>
</tr>
<tr>
<td><code>print(None, print(None))</code></td>
<td></td>
</tr>
<tr>
<td><code>jazz(6)</code></td>
<td></td>
</tr>
<tr>
<td><code>(lambda out: jazz(8))(9)</code></td>
<td></td>
</tr>
<tr>
<td><code>twist(2, lambda x: x-2)(4)</code></td>
<td></td>
</tr>
<tr>
<td><code>twist(5, print)(out)</code></td>
<td></td>
</tr>
<tr>
<td><code>twist(6, lambda hands: hands-out, 2)(-1)</code></td>
<td></td>
</tr>
</tbody>
</table>
(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.

```python
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)
```

Environment Diagram (Without Lists) #1

---

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)
```

Global frame

| inside | _____ |
| _____ | _____ |
| _____ | _____ |

func inside(out) [parent=Global]

f1: inside [parent=Global]

| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |
| _____ | _____ |

func λ(fear) <line 2> [parent=_____]

func λ(disgust) <line 3> [parent=_____]

func λ(fear) <line 8> [parent=_____]

f2: _____ [parent=_____]

Return Value

f3: _____ [parent=_____]

Return Value

f4: _____ [parent=_____]

Return Value

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
(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.*

```python
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