EXAM PREPARATION SECTION 1

HIGHER ORDER FUNCTIONS, ORDER OF EVALUATION, ENV. DIAGRAMS

January 29 to February 2, 2018

1 Code Writing Problems

Usually, we begin with a description of the problem to be solved. What's important is not just reading the problem, but thinking critically about the details in the problems and clearing up any ambiguities. A couple concrete starting questions to ask yourself on any problem include:

- What is the domain (input) and range (output) of the program?
- Restate the intended behavior of the program in your own words.
- How will the values in this program change as the program executes?

Verify your understanding by studying the doctests. Big hints are always given away in the doctest! If we look closely enough for the patterns in the doctest, we'll often expose details in the structure of how the problem is meant to be solved.

Although they provide many hints, the doctests are not exhaustive and they usually don't show the most important cases. Develop examples that cover at least the following situations:

- What's the smallest or simplest possible input I could give to this function?
- Is there a similar small input that is invalid for this problem? How is it related to or different from the earlier case?
- Can we come up with any larger inputs to the program that are related to or rely on smaller cases? The idea is to come up with some of the subproblems we might have to solve with recursion or other techniques.

Then, after you write your initial solution, test your solution manually. Plug in the inputs from the doctests and verify that your code leads to the correct results.

1. **You Complete Me (Sp15 Midterm 1 Q3a)**
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)
    # Note: 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
```

2. **A Highly Intelligent Animal (Su15 Midterm 1 Q4c)**
   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):
    """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
```

3. **Digit Fidget (Fa15 Midterm 1 Q3c)**

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
```

2 What Would Python Print?

1. Dog Goes Woof (Fa13 Midterm 1 Q1)

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

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

```
from operator import add, mul
def square(x):
    return mul(x, x)

def dog(bird):
    def cow(tweet, moo):
        woof = bird(tweet)
        print(moo)
        return woof
    return cow

cat = dog(square)
```

Expression	Evaluates to	Interactive Output
`square(5)`	25	25
`1/0`	ERROR	ERROR
`add(square(2), mul(3, 4))`		
`print(print(print(2)))`		
`cat(3, 4)`		
`square(cat(5))`		
`cat(square(2), print(5))`		
`cat(print(square(3)), 8)`		

3 Environment Diagrams

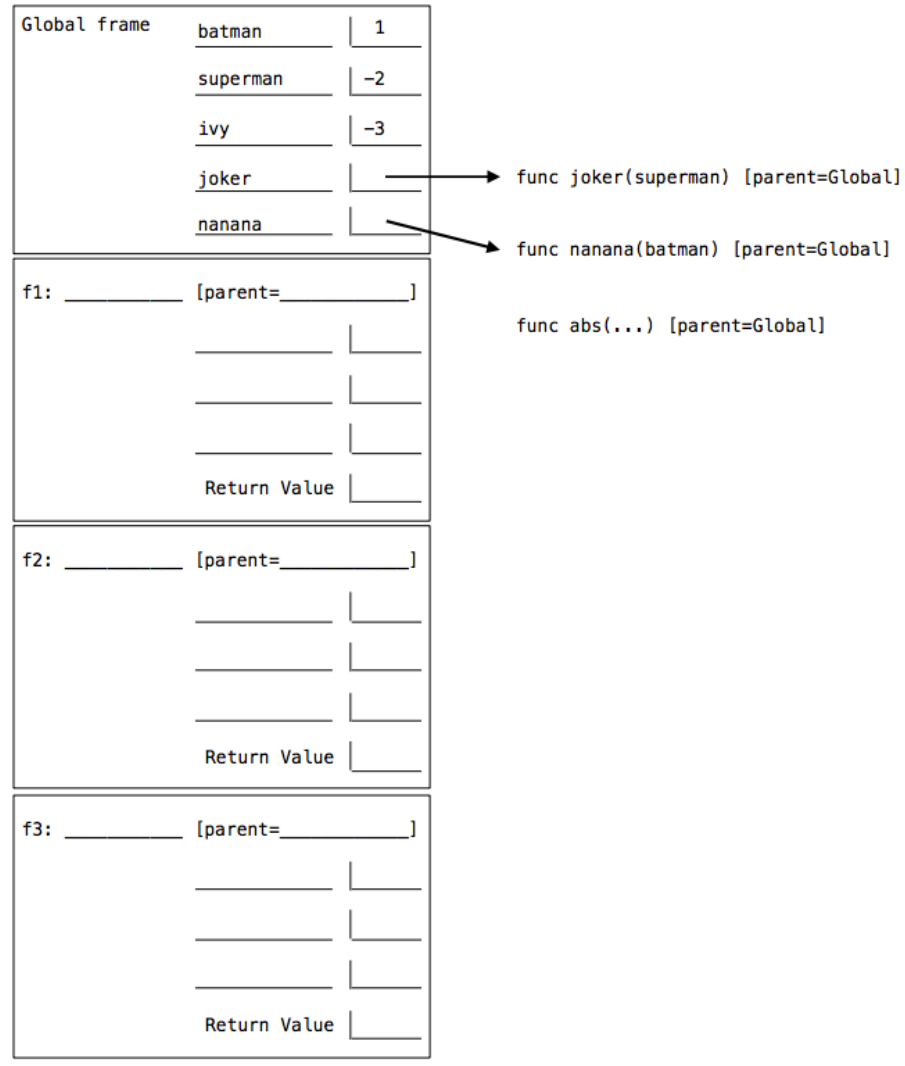
1. **Supernatural (Sp15 Midterm 1 Q2a)**
   Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.*

   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.

   You must list all bindings in the order they first appear in the frame.

```
1  batman, superman, ivy = 1, -2, -3
2
3  def nanana(batman):
4      while batman(superman) > ivy:
5          def batman(joker):
6              return ivy
7      return -ivy
8
9  def joker(superman):
10     if superman(batman):
11         ivy = -batman
12     return nanana
13
14 joker(abs)(abs)
```



2. **Envy, Iron, Mint (Fa14 Midterm 1 Q2a)**
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 peace(today):
2      harmony = love+2
3      return harmony + today(love+1)
4
5  def joy(peace):
6      peace, love = peace+2, peace+1
7      return love // harmony
8
9  love, harmony = 3, 2
10 peace(joy)
```