MIDTERM 1 REVIEW

COMPUTER SCIENCE MENTORS 61A

February 15, 2017

1 Control

1. Fill in the blanks of `sum_k_digits`. `sum_k_digits` takes in two integers, `n` and `k` and sums the rightmost `k` digits of `n`. If `k` is greater than `n`, sum up all of the digits. See the doctests for more details:

```
def sum_k_digits(n, k):
        """ Returns the sum of the rightmost k digits of n.
           Assume n has >= k digits.
              >>> sum_k_digits(11111, 3)
              3
              >>> sum_k_digits(12345, 2)
              9
        """
        total = _____________________________________

        while _____________________________________:

                total = _____________________________________

                n = _____________________________________

                k = _____________________________________

        return _____________________________________
```

Solution:

```
def sum_k_digits(n, k):
        """ Returns the sum of the rightmost k digits of n.
            Assume n has >= k digits.
             >>> sum_k_digits(11111, 3)
             3
             >>> sum_k_digits(12345, 2)
             9
        """
        total = 0
        while k > 0:
                total = total + (n % 10)
                n = n // 10
                k = k - 1
        return total
```

2. **Extra:** How would you solve it recursively?

Solution:

```
def sum_k_digits(n, k):
        """ Returns the sum of the rightmost k digits of n.
            Assume n has >= k digits.
             >>> sum_k_digits(11111, 3)
             3
             >>> sum_k_digits(12345, 2)
             9
        """
        if k == 0:
           return 0
        return n % 10 + sum_k_digits(n//10, k-1)
```

2 Environment Diagrams

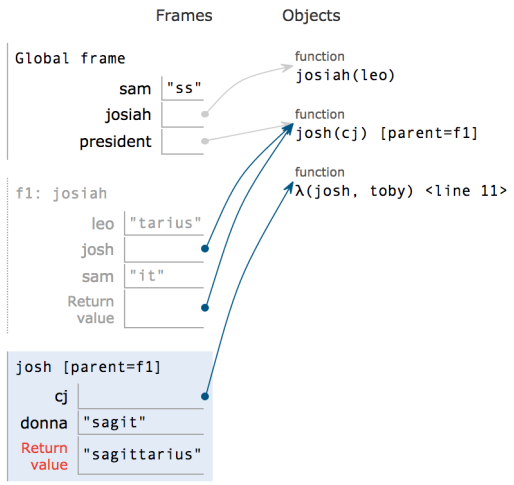
1. Fill in the environment diagram that results from executing the code below until the entire program is finished or an error occurs. You may not need to use all of the spaces or frames. A complete answer will:
   - Add all missing names and parent annotations to all local frames.
   - Add all missing values created or referenced during execution.
   - Show the return value for each local frame.

```
sam = "ss"

def josiah(leo):
    sam = "it"
    def josh(cj):
        donna = cj(sam, "sag")
        return donna + leo
    return josh

president = josiah("tarius")
password = president(lambda josh, toby: toby + josh)
```

Solution:

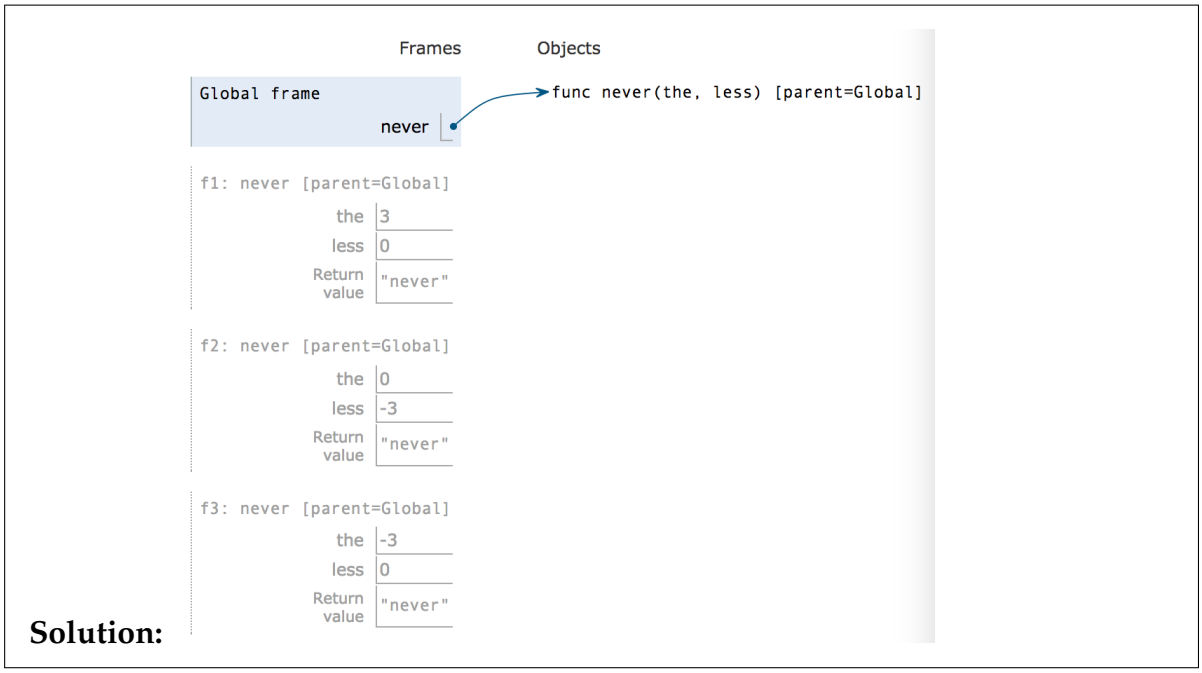


2. Fill in the environment diagram that results from executing the code below until the entire program is finished or an error occurs. You may not need to use all of the spaces or frames. A complete answer will:
   - Add all missing names and parent annotations to all local frames.
   - Add all missing values created or referenced during execution.
   - Show the return value for each local frame.

```
def never(the, less):
    if the < less:
        return 'never'
    elif not less:
        print('always')
    if less == -3:
        return never(less, the)
    return never(less, less - the)

never(3, 0)
```

Solution:



3 What Would Python Display

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

```
def pup(bark):
    woof = 10;
    def yip(yap):
        if bark % yap == 0:
            return woof * 3
        return yap + woof
    return yip
def spot(dog):
    per = 39
    if dog > 5:
        print("pup")
    if dog > 10:
        return pup(per)
def cloud(grr):
    print(grr * 3)
woof = 9
```

Expression	Interactive Output
`py = woof % 3`	
`pet = spot(13)`	
`print(cloud(woof + 6))`	
`pet(py)`	
`pet(woof)`	
`pup(py)`	
`pet(3)`	

Solution:

Expression	Interactive Output
`py = woof % 3`	
`pet = spot(13)`	`pup`
`print(cloud(woof + 6))`	`45` `None`
`pet(py)`	`Error`
`pet(woof)`	`19`
`pup(py)`	`Function`
`pet(3)`	`30`

2. **Extra:**

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

```
def dell(ta):
    lamduh = [3, 1, 4, 1, 5, 9]
    pie = lamduh
    def eye(ota):
        return lambda pie, kye: pie[:-2] + kye[ota[0]::-1]
    lamduh = lamduh + ta
    print(pie)
    return eye

def bay(ta):
    row = [ca - 9 for ca in ta]
    print(row, ta)
    return dell
```

Expression	Interactive Output
`fie = [2, 17, 20, 17]`	
`nul = [0, -1, -50]`	
`gam = bay(fie)`	
`tao = gam(nul)`	
`print(tao([1, 5, 7])(fie, nul))`	

Solution:

Expression	Interactive Output
`fie = [2, 17, 20, 17]`	
`nul = [0, -1, -50]`	
`gam = bay(fie)`	`[-7, 8, 11, 8] [2, 17, 20, 17]`
`tao = gam(nul)`	`[3, 1, 4, 1, 5, 9]`
`print(tao([1, 5, 7])(fie, nul))`	`[2, 17, -1, 0]`

4 Higher Order Functions

1. Fill in the blanks so that the doctest passes.

```
def repeated(f, n):
    """
    >>> repeated(lambda x: x*x, 2)(2)
    16
    """
     g = lambda x: x

     while ____________________________________:

         g = ____________________________________

         ____________________________________

     return g
```

Solution:

```
    g = lambda x: x
    while n > 0:
        g = lambda x: f(g(x))
        n -= 1
```

```
    return g
```

5 Recursion

1. Fill in the blanks to `replace`, so that it returns a number identical to n, but where every digit old is replaced with digit new.

```
def replace(n, old, new):
    """
    Return a number that is identical to n except that all
       instances of old in n are replaced with new. Assume that
        old and new are both positive integers and less than
       10.
    >>> replace(10101, 1, 2)
    20202
    """
    if _____________________________________________:

        return 0

     last = _____________________________________________

     rest = _____________________________________________

     if last == old:

          _____________________________________________


     else:

          _____________________________________________
```

Solution:

```
    if n == -:
        return 0
     last = n % 10
     rest = n //10
```

```
    if last == old:
        return replace(rest, old, new) * 10 + new
    else:
        return replace(rest, old, new) * 10 + last
```

2. Write a function that takes as input a number `n` and a list of numbers `lst` and returns `True` if we can find a subsequence of `lst` that sums up to `n`

```
def addup(n, old, new):
    """
    >>> addup(10, [1, 2, 3, 4, 5])
    True
    >>> addup(8, [1, 2, 3, 4, 5])
    True
    >>> addup(-1, [1, 2, 3, 4, 5])
    False
    >>> addup(100, [1, 2, 3, 4, 5])
    False
    """
    if _____________________________________________:

        return True

    if lst == []:

        ____________________________________________________________

    else:

        first, rest = _____________________,
           ____________________

        return _____________________________________________
```

Solution:

```
    if n == 0:
        return True
    if lst == []:
        return False
    else:
```

```
        first, rest = lst[0], lst[1:]
        return addup(n - first, rest) or addupt(n, rest)
```

6 Linked Lists and Trees

Here are the constructors and selectors of `link` and textttttree:

```
def tree(label, branches=[]):
    return [label] + list(branches)

def label(t):
    return t[0]

def branches(t): # Always returns a list of trees
    return t[1:]
```

1. Write a function that takes in a linked list, lnk, and returns a linked list that is the reverse of lnk. That is, the first element of the returned list is the last element of lnk, the second element is the second to last element of lnk, and so on.

```
def reverse(lnk):
    >>> reverse(link(1, empty)))
    link(1, empty)
    >>> reverse(link(2, link (4, empty))
    link(4, link(2, empty))
    def reverse(lnk):

        reversed = _____

        while _____:

            reversed = _____

            lnk = _____

        return _____
```

Solution:

```
        reversed = empty
        while lnk != empty:
```

```
            reversed = link(first(lnk), reversed)
            lnk = rest(lnk)
        return reversed
```

2. Write a function that returns true only if there exists a path from root to leaf that contains at least n instances of elem in a tree t.

```
def contains_n(elem, n, t):
    >>> t1 = tree(1, [tree(1,tree(2)])
    >>> contains(1, 2, t1)
    True
    >>> contains(2, 2, t1)
    False
    >>> contains(2, 1, t1)
    True
    >>> t2 = tree(1, [tree(2), tree(1, [tree(1), tree(2)])
    >>> contains(1, 3, t1)
    True
    >>> contains(2, 2, t1) # Not on a path
    False
    if n == 0:

        return True

    elif n == 1 and ____________________________________:

        return True

    elif ____________________________________:

        return ____________________________________

    elif label(t) == elem:

        return ____________________________________________________

    else:

        return ____________________________________
```

Solution:

```
if n == 0:
    return True
elif n == 1 and is_leaf(t) and label(t) == elem:
    return True
elif is_leaf(t):
    return False
elif label(t) == elem:
    return True in [contains_n(elem, n - 1, b) for b in
      branches(t)]
else:
    return True in [contains_n(elem, n, b) for b in
      branches(t)]
```