

PRACTICE EXAM *for* MIDTERM 2 SOLUTIONS

PRACTICE MIDTERM 2

Computer Science 61A . October 20, 2015 . alvinwan.com/cs61a

- You have 2 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one hand-written 8.5" × 11" crib sheet of your own creation and the official 61A midterm 2 study guide attached to the back of this exam.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a brief explanation.

First Name	
Last Name	
SID	
Email (...@berkeley.edu)	
Login (e.g., cs61a-ta)	
TA & section time	
Name of person to your left	
Name of person to your right	
<i>All the work on this exam is my own.</i> (please sign)	

0. (0 Points) On a scale of Paul to Nathan (mod Cindy), how do you feel? This question is gibberish; feel free to ignore.

Paul (mod Cindy) _____ Nathan (mod Cindy)

PRACTICE EXAM *for* MIDTERM 2 SOLUTIONS**01. (12 Points) TO SING OR NOT TO SING**

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.** Expressions are evaluated in order, and **expressions may affect later expressions.**

Whenever the interpreter would report an error, write Error. If execution would take forever, write Forever. Assume that you have started Python 3 and executed the following statements:

```
class Person:
    genres = ['fabulous!', 'oh no.']
    def __init__(self, octaves=(1, 2), genres=0):
        self.octaves = octaves
        if genres:
            self.genres = genres

    def sing(self):
        if not self.shift_octave(1):
            print(self.genres.pop(0))
            self.genres.append('yuk.')

    def shift_octave(self, shift):
        self.octaves[0] = shift

alvin, angie = Person(), Person([0, 1, 2, 3, 4, 5, 6])
paul = Person([11, 12, 13], ['b-e-a-U-tiful'])
```

Expression	Interactive Output
alvin.sing()	Error
angie.sing()	fabulous!
alvin.octaves = [0] alvin.sing()	oh no.
paul.sing()	b-e-a-U-tiful
Person.genres	['yuk.', 'yuk.']

PRACTICE EXAM *for* MIDTERM 2 SOLUTIONS**02.** (14 Points) CHIKIN

(a) (3 Points) First, fill in the following box-and-pointer diagrams that result from the following piece of code. If the interpreter would report an error, write Error. If execution would take forever, write Forever.

```
lst = ['A', 'B', 'C', 'D', 'E']
lst[1] = ['F', 'G', 'H', 'I', 'J']
lst[1][4] = ['K', 'L', 'M', 'N', 'O']
lst[1][0] = lst[1][4]
x = lst[1]
y = x[0]
y[2] = x[2:4]
x, lst[1], y = y[2], lst, lst[1]
```

LST

A	[lst]	C	D	E
---	-------	---	---	---

x

H	I
---	---

y

[A]	G	H	I	[A]
-----	---	---	---	-----

NOTE: [x] means pointer to list x. I employ this silly notation for the solutions, because arrows are difficult to draw on the computer.

A - not a variable in the global scope

K	L	[X]	N	O
---	---	-----	---	---

(b) (5 Points) Then, fill in the following piece of code, to output 'CHIKIN'. If the interpreter failed previously, runs forever, would error, or simply cannot be completed, write Impossible. You may only retrieve indices, using square brackets and integers.

```
>>> c, h, i, k, n = lst[1][2], x[0], y[0][2][1], y[0][0][0], y[0][3]
>>> c + h + i + k + i + n
```

PRACTICE EXAM *for* MIDTERM 2 SOLUTIONS

(c) (5 Points) 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:

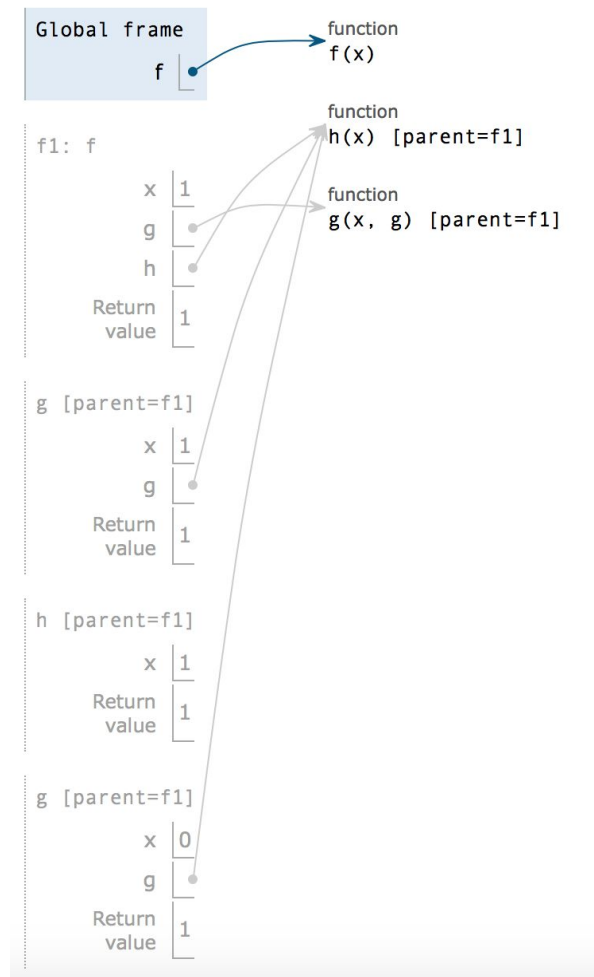
- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

Remember: Do not add a new frame when calling a built-in function (such as abs). The built-in abs function is always written as `func abs(...)` [`parent=Global`].

```
def f(x):
    def h(x):
        return x*g(x-1, h)
    def g(x, g):
        if x <= 0:
            return 1
        return g(x)
    return g(abs(x), h)
```

`f(1)`

[Python Tutor](#)



(d) (1 Point) What does the above function call `f(x)` compute, in terms of `x`?

`|x|!`

PRACTICE EXAM *for* MIDTERM 2 SOLUTIONS**03. (8 Points) TREES GONE HAYWIRE**

(a) (5 points) In lab, we implemented `has_a_cycle` for a linked list, where the last element of a cyclic list points to the head. We now define a *cyclic tree* to be any tree where a node has the root node as a child. Implement `has_a_cycle` for a tree below. Both the `Tree` class and the tree data abstraction appear on the midterm 2 study guide. *Warning: Do not violate the tree data abstraction! (Exams are flammable.)*

```
def has_a_cycle(t):
    """Returns whether or not a tree is cyclic. We define a cyclic tree as
    any tree with a node that has the root node as a child. Assume that all
    nodes in the tree contain distinct values.

    >>> t = Tree(3, Tree(5, Tree(4), Tree(6)),
    ...     Tree(8, Tree(10), Tree(1, Tree(2))))
    >>> has_a_cycle(t)
    False
    >>> t.right.left.left = t
    >>> has_a_cycle(t)
    True
    >>> t.right.left.left = Tree.empty
    >>> has_a_cycle(t)
    False
    >>> t.right.right.right = t
    >>> has_a_cycle(t)
    True
    """
    return any([detective(t, b) for b in branches(t)])
```

```
def detective(tortoise, hare=None):
    if not hare or (not hare.right and not hare.left):
        return False # no need to check tortoise, since hare moves faster

    if tortoise.value == hare.value:
        return True

    return any(any(any(detective(t, bb) for bb in branches(b))
                    for b in branches(hare)) for t in branches(tortoise))
```

PRACTICE EXAM *for* MIDTERM 2 SOLUTIONS

(b) **(2 points)** Complete the following `has_a_cycle` implementation to detect if a Python list is cyclic.

```
has_a_cycle = lambda lst: any(x == lst for x in lst)
```

(c) **(1 point)** Compute the runtime of `has_a_cycle` in 3(b) with respect to the number of elements n in the list `lst`.

Runtime: $O(n)$

(d) **(Bonus, 1 point)** Write a one-line function that can detect a nested, cyclic list of *any depth*, using `has_a_cycle_helper(x, y)`, which achieves the same result but checks if `x` exists in `y`. Assume that your lists *only* contain either integers or pointers to other lists.

```
has_a_cycle_nested = lambda lst: has_a_cycle_helper(lst, lst)
```

(e) **(Bonus, 5 points)** Now, implement the *one-line function* `has_a_cycle_nested(x, y)` used in the function above. Assume that your lists *only* contain either integers or pointers to other lists. The additional lines are added in case your code is longer in width than the space provided. Do not add additional lines of code.

```
has_a_cycle_helper = lambda x, y: any(elem == x or (False if
isinstance(elem, int) else has_a_cycle_helper(x, elem)) for elem in y)
```

AREA INTENTIONALLY LEFT BLANK

Feel free to use this space to show work, doodle, or spill coffee.

PRACTICE EXAM *for* MIDTERM 2 SOLUTIONS**04. (4 Points) PASCAL'S TRIANGLE**

Pascal's Triangle is a "triangle" of values, beginning at 1, where each subsequent row is generated by summing each pair of values in the previous row. Implement `pascals_generator()`, which returns a function that will sequentially generate the next series of `k` elements in Pascal's triangle. Hint: The list should look like: [1, 1, 1, 1, 2, 1, 1, 3, 3, 1, 1, 4, 6, 4, 1 ...].

```
def pascals_generator():
    """ Generate a subset of Pascal's triangle as a list.

    >>> pascals = pascals_generator()
    >>> pascals(3)
    [1, 1, 1]
    >>> pascals(3)
    [2, 1, 1]
    >>> pascals(4)
    [3, 3, 1, 1]
    """
    lst, row, index = [1], [1], 1
    def pascals(k):
        nonlocal row, index # notice that lst is not declared nonlocal
        while index + k >= len(lst):
            indices = range(len(row)-1)
            center = [row[i] + row[i+1] for i in indices]
            row = [1] + center + [1]
            lst.extend(row) # MUST use lst.extend, cannot lst = ... (why?)
            index += k
        return lst[index-k:index]
    return pascals
```