CS 61A Structure and Interpretation of Computer Programs
Spring 2k17

Quiz 1 Solutions

1. (2 points) Back to the Stone Age

Are these primitive expressions or call expressions? (Answer for each one individually.)

(a) `5`
Primitive. 5 is a number.

(b) `add(7, 7)`
Call expression. `add(7, 7)` is calling the function `add`.

(c) `print(3)`
Call expression. `print(3)` is calling the builtin function `print`.

(d) `4 + 5`
Call expression. `4 + 5` is calling the builtin function `__add__`, and is equivalent to `(4).__add__(5)`.

2. (3 points) Control Yourself

Imagine you have the boolean values `isRaining`, `fallsBehind`, `doesProcrastinate`, `numExamsTaken == 45`, and `addictedToCandy`. Using only these values and the operators `and`, `or`, and `not`, fill in the blank with a boolean expression that represents – based on the following quote – whether or not you will ace this class:

> "I will ace this class if I keep up with the material, don't procrastinate, and take all the past exams*. Even if I don't do that stuff, I will ace this class if it is raining. Conversely, I will fail this class if I eat too much candy."

** There are 45 past exams.*

```
will_ace_this_class = \
(not fallsBehind and not doesProcrastinate and numExamsTaken == 45 \
or isRaining) and not addictedToCandy
```

3. (5 points) Functions of a Higher Order

Write a function that

(a) takes a single function as input, and

(b) returns a function that does the same thing as the input function, but also prints "`gr8 m8 i r8 8/8`" every eighth time the function is called.

To deal with variability in the number of arguments to the input function, use `*args` as both your inner function's formal parameters and the input function's arguments. (`*args` packs an arbitrary number of arguments into a tuple in the first case, and unpacks all of them into separate positional arguments in the second case.)

A function skeleton has been provided for you on the next page. Fill in your answer there.

```
def hof8(input_fn):
    """Returns a ver. of the input fn that trolls you upon every 8th call.
    >>> f = hof8(max)
    >>> f(1, 2) + f(f(f(f(f(f(3, 4), 5), 6), 7), 8), 9) # 7 calls
    11
    >>> f(10, 11) # 8th call
    gr8 m8 i r8 8/8
    11
    >>> f(12, 13) # 9th call; message should not be printed
    13
    """
    callCount = [0] # this is a list; update w/ callCount[0] = <whatever>
    def inner(*args):
        if (callCount[0] + 1) % 8 == 0:
            print('gr8 m8 i r8 8/8')
        callCount[0] = callCount[0] + 1
        return input_fn(*args)
    return inner
```

—

Of note: there is a reason I've made `callCount` a list and not just a number! If the line were instead `callCount = 0`, we would not be able to reassign `callCount` within an inner function – *because assignments are made within the current frame*, even if the relevant name is already defined inside a parent frame. This is important to understand.