CS 61A Functions, Control, Environments, HOFs
Fall 2019 Guerrilla Section 0: September 13, 2019

1 Functions

Questions

1.1 Determine what the Python interpreter will output given the following lines of code.

```
>>> from operator import add, mul
>>> mul(add(5, 6), 8)


>>> print('x')


>>> y = print('x')


>>> print(y)


>>> print(add(4, 2), print('a'))
```

1.2 Determine what the Python interpreter will output given the following lines of code.

```
>>> def foo(x):
        print(x)
        return x + 1

>>> def bar(y, x):
        print(x - y)

>>> foo(3)


>>> bar(3)


>>> bar(6, 1)


>>> bar(foo(10), 11)
```

2 Control

Questions

2.1 Which numbers will be printed after executing the following code?

```
n = 0
if n:
    print(1)
elif n < 2
    print(2)
else:
    print(3)
print(4)
```

2.2 WWPD (What would Python Display) after evaluating each of the following expressions?

```
>>> 0 and 1 / 0

>>> 6 or 1 or "a" or 1 / 0

>>> 6 and 1 and "a" and 1 / 0

>>> print(print(4) and 2)

>>> not True and print("a")
```

2.3 Define a function, count_digits, which takes in an integer, n, and counts the number of digits in that number.

```
def count_digits(n):
    '''
    >>> count_digits(4)
    1
    >>> count_digits(12345678)
    8
    >>> count_digits(0)
    0
    '''
```

2.4 Define a function, count_matches, which takes in two integers n and m, and counts the number of digits that match.

```
def count_matches(n, m):
    '''
    >>> count_matches(10, 30)
    1
    >>> count_matches(12345, 23456)
    0
    >>> count_matches(121212, 123123)
    2
    >>> count_matches(111, 11) # only one's place matches
    2
    >>> count_matches(101, 10) # no place matches
    0
    '''
```

3 Environment Diagrams

Questions

3.1 Draw the environment diagram for evaluating the following code

```
def f(x):
    return y + x
y = 10
f(8)
```

3.2 Draw the environment diagram for evaluating the following code

```
def dessef(a, b):
        c = a + b
        b = b + 1

b = 6
dessef(b, 4)
```

3.3 Draw the environment diagram for evaluating the following code

```
def foo(x, y):
        foo = bar
        return foo(bar(x, x), y)

def bar(z, x):
        return z + y

y = 5
foo(1, 2)
```

3.4 Draw the environment diagram for evaluating the following code

```
def spain(japan, iran):
        def world(cup, egypt):
                return japan-poland
        return iran(world(iran, poland))

def saudi(arabia):
        return japan + 3

japan, poland = 3, 7
spain(poland+1, saudi)
```

3.5 Draw the environment diagram for evaluating the following code

```
cap = 9
hulk = 3

def marvel(cap, thor, avengers):
    marvel = avengers
    iron = hulk + cap
    if thor > cap:
        def marvel(cap, thor, avengers):
            return iron
    else:
        iron = hulk
    return marvel(thor, cap, marvel)

def iron(man):
    hulk = cap - 1
    return hulk

marvel(cap, iron(3), marvel)
```

4 Higher Order Functions

Questions

4.1 What do lambda expressions do? Can we write all functions as lambda expressions? In what cases are lambda expressions useful?

4.2 Determine if each of the following will error:

```
>>> 1/0

>>> boom = lambda: 1/0

>>> boom()
```

4.3 Express the following lambda expression using a **def** statement, and the **def** statement using a lambda expression.

```
pow = lambda x, y: x**y
```

```
def foo(x):
    def f(y):
        def g(z):
            return x + y * z
        return g
    return f
```

4.4 Draw Environment Diagrams for the following lines of code

```
square = lambda x: x * x
higher = lambda f: lambda y: f(f(y))
higher(square)(5)

a = (lambda f, a: f(a))(lambda b: b * b, 2)
```

4.5 Write **make_skipper**, which takes in a number n and outputs a function. When this function takes in a number x, it prints out all the numbers between 0 and x, skipping every nth number (meaning skip any value that is a multiple of n).

```
def make_skipper(n):
    """
    >>> a = make_skipper(2)
    >>> a(5)
    1
    3
    5
    """
```

4.6 Write a function that takes in a function `cond` and a number `n` and prints numbers from `1` to `n` where calling `cond` on that number returns `True`.

```
def keep_ints(cond, n):
    """Print out all integers 1..i..n where cond(i) is true

    >>> def is_even(x):
    ...     # Even numbers have remainder 0 when divided by 2.
    ...     return x % 2 == 0
    >>> keep_ints(is_even, 5)
    2
    4
    """
```

4.7 Write a function similar to `keep_ints` like before, but now it takes in a number `n` and returns a function that has one parameter `cond`. The returned function prints out numbers from 1 to `n` where calling `cond` on that number returns `True`.

```
def make_keeper(n):
    """Returns a function which takes one parameter cond and prints out
    all integers 1..i..n where calling cond(i) returns True.

    >>> def is_even(x):
    ...     # Even numbers have remainder 0 when divided by 2.
    ...     return x % 2 == 0
    >>> make_keeper(5)(is_even)
    2
    4
    """
```