

---

# CS 61A Functions, Control, Environments, HOFs

## Spring 2020

---

Guerrilla Section 0: February 7, 2020 **Solutions**

## 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)
```

88

```
>>> print('x')
```

x

```
>>> y = print('x')
```

x

```
>>> print(y)
```

None

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

a

6 None

- 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)
```

3

4

```
>>> bar(3)
```

Error

```
>>> bar(6, 1)
```

-5

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

10

0

## 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

4

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

```
>>> 0 and 1 / 0
```

0

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

6

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

Error

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

4

None

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

False

- 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
...

count = 0
while n > 0:
    count += 1
    n = n//10
return count
```

- 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
...

matches = 0
while n > 0 and m > 0:
    if n % 10 == m % 10:
        matches += 1
    n, m = n // 10, m // 10
return matches
```

## 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)
```

Solution: <https://goo.gl/rZnzaM>

- 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)
```

Solution: <https://goo.gl/4m3NRD>

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)
```

Solution: <https://goo.gl/7Kcx6n>

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)
```

Solution: <https://goo.gl/iddW49>

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)
```

Solution: <https://goo.gl/sofcq2>

## 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?

Lambda expressions create functions. When a lambda expression is evaluated, it produces a function. We often use lambdas to create short anonymous functions that we won't need for too long.

We can't write all functions as lambda expressions because lambda functions all have to have `return` statements and they can't contain very complex multi-line expressions.

- 4.2 Determine if each of the following will error:

```
>>> 1/0
```

Error

```
>>> boom = lambda: 1/0
```

No error, since we don't evaluate the body of the lambda when we define it.

```
>>> boom()
```

Error

- 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 pow(x, y):
    return x**y
```

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

```
foo = lambda x: lambda y: lambda z: x + y * z
```

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)
```

Solution: <https://goo.gl/LATqV9>

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

Solution: <https://goo.gl/TyriuP>



- 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 *n*th number (meaning skip any value that is a multiple of *n*).

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

    def skipper(x):
        for i in range(x + 1):
            if i % n != 0:
                print(i)
    return skipper
```

## 5 Extra Questions

### Questions

- 5.1 Define a function, `ordered_digits`, which takes in a positive integer, `x`, and returns `True` if the (base 10) digits of `x` are in non-decreasing order, and `False` otherwise.

```
def ordered_digits(x):
    '''
    >>> ordered_digits(5)
    True
    >>> ordered_digits(11)
    True
    >>> ordered_digits(127)
    True
    >>> ordered_digits(1357)
    True
    >>> ordered_digits(21)
    False
    >>> result = ordered_digits(1375) # Return, don't print
    >>> result
    False
    '''

    last = x % 10
    val = x // 10
    while x > 0 and last >= x % 10:
        last = x % 10
        x = x // 10
    return x == 0
```

- 5.2 Define a function, `cycle`, which takes in three functions, `f1`, `f2`, `f3`, and returns a function that takes in an integer `n` and returns a function that takes in an integer `x`, and returns the result of `f1(x)` the first time it's called, `f2(x)` the second time it's called, `f3(x)` the third time it's called, and then cycles back to `f1(x)` the fourth time it's called, and so on.

```
def cycle(f1, f2, f3):
    """Returns a function that is itself a higher-order function.

    >>> def add1(x):
    ...     return x + 1
    >>> def times2(x):
    ...     return x * 2
    >>> def add3(x):
    ...     return x + 3
    >>> my_cycle = cycle(add1, times2, add3)
    >>> identity = my_cycle(0)
    >>> identity(5)
    5
    >>> add_one_then_double = my_cycle(2)
    >>> add_one_then_double(1)
    4
    >>> do_all_functions = my_cycle(3)
    >>> do_all_functions(2)
    9
    >>> do_more_than_a_cycle = my_cycle(4)
    >>> do_more_than_a_cycle(2)
    10
    >>> do_two_cycles = my_cycle(6)
    >>> do_two_cycles(1)
    19
    """

    def ret_fn(n):
        def ret(x):
            i = 0
            while i < n:
                if i % 3 == 0:
                    x = f1(x)
                elif i % 3 == 1:
                    x = f2(x)
                else:
                    x = f3(x)
                i += 1
            return x
        return ret
    return ret_fn
```

- 5.3 Define a function, `is_palindrome`, which takes in an integer, `n`, and returns `True` if `n` is a palindrome and `False` otherwise.

```
def is_palindrome(n):
    """
    Fill in the blanks '_____' to check if a number
    is a palindrome.
    >>> is_palindrome(12321)
    True
    >>> is_palindrome(42)
    False
    >>> is_palindrome(2015)
    False
    >>> is_palindrome(55)
    True
    """
    x, y = n, 0
    f = lambda: _____
    while x > 0:
        x, y = _____, f()
    return y == n

x, y = n, 0
f = lambda: y * 10 + x % 10
while x > 0:
    x, y = x // 10, f()
return y == n
```