**Made by Max Poole**

**True or False?**

1. **All functions must have a return statement** False
2. **An integer is mutable** False
3. **You cannot do this: “Hello” + “Goodbye”** False, you can do that. It is called concatenating strings
4. **x+=1 adds one to the value of x (assume that x is an integer)** True, note that x++ does not work in Python 3
5. **Dictionaries can have multiple values attached to one key** False, each key has one value attached to it

**Write a program that simulates connect 4 or write a program that for numbers between 0 and 100 (inclusive of 100) prints out “no” if the number is divisible by 3, “nope” if the number is divisible by 5, and “nonono” if the number is divisible by both 3 and 5. If the number is not divisible by 3 or 5, then print the number.**

def nono(): #Note that this is just the classic "fizzbuzz" problem

for number in range(1, 101):

if number % 15 == 0: #You may also write as “if number % 3 == 0 and number % 5 == 0”

print("nonono")

elif number % 3 == 0:

print("no")

elif number % 5 == 0:

print("nope")

else:

print(number)

**Convert from base 8 to base 10**

*130* 88

**Convert from base 4 to base 2**

*121* In decimal: 25 In base 2: 11001

**Convert from base 10,000,000,000 to base 89:**

*1* 1

**Evaluate whether the following statements are true or false:**

*not(True or False) and True* False

*True and False and True and True and False and False or True* True (notice that this is very easy if you look at the “or” statement at the end)

**What part of the below code will not get evaluated due to short circuit evaluation?**

*hasATire = True*

*hasAWheel = False*

*numberTire = 4*

*if hasAWheel and hasATire and numberTire == 4:*

*print(“This is probably a car”)*

This code will stop after “hasAWheel” as that is false and there is an “and” statement following it.

**Describe any three emacs commands that you did not use on your midterm:**

[Here](https://www.cs.rutgers.edu/LCSR-Computing/some-docs/emacs-chart.html) is a nice collection of commands.

**Given a non-empty string like "Code" return a string like "CCoCodCode".**

**Examples:**

**string\_splosion('Code') → 'CCoCodCode'**

**string\_splosion('abc') → 'aababc'**

**string\_splosion('ab') → 'aab'**

My solution:

def string\_splosion(myString):

newString = ""

for i in range(len(myString)+1):

newString += myString[:i]

return newString

This problem is taken from CodingBat, you can see their solution [here](http://codingbat.com/prob/p118366)

**Describe a bubble sort algorithm using words, pictures, code, or interpretive dance:**

[Here](http://www.sorting-algorithms.com/) is a nice site describing all the sorting algorithms you would like. [Here](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0CCAQyCkwAA&url=http%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3DlyZQPjUT5B4&ei=TtZQVe7BN4SoNsaWgGA&usg=AFQjCNHK3Ave1qVJMWKY9ZOHUNGoPi03Tw&bvm=bv.92885102,d.eXY) is a dance representation of Bubble Sort.

**Sort the following sorting algorithms by their average speed in Big O(n) notation (slowest to fastest):**

1. *Bubble Sort*
2. *Selection Sort*
3. *Merge Sort*
4. *Quick Sort*
5. *Bogo Sort (randomizes a list and then checks if it is sorted)*

Bogo, Selection Sort, Bubble Sort, Merge Sort, Quick Sort

Also accepted as correct:

Bogo, Bubble Sort, Selection Sort, Quick Sort, Merge Sort

Note that sorting algorithms can be faster than other algorithms that are normally faster in specific scenarios. Also note that there are some “best-case” scenarios where normally slow sorting algorithms can be very fast. If you happened to get lucky Bogo Sort could be one of the fastest sorting algorithms by the pure power of randomness.

**What is the Big O(n) for the following program?**

def testQuestion(n):

for i in range(n):

for j in range(n):

print(“yuperdoodle”)

Answers of n^2

**What will the below code print?**

def my\_function(param=[ ]):

param.append("thing")

return param

print(my\_function())

print(my\_function())

Will print out [“thing”] and then [“thing”, “thing”]

This question is taken from [Code Habitude](http://codehabitude.com/2013/12/24/python-objects-mutable-vs-immutable/) they have a more in depth discussion of mutability and immutability in Python that is worth looking over.

**What will the below code print?**

def my\_function(param = 3):

param += 2

return param

print(my\_function())

print(my\_function())

This will print 5 and then 5. This is due to the immutability of integers.

**Write a function that takes the contents of “test.txt” and prints out each character in the file. The file contains the message:**

**“***I am a file, yes”*

**There are no line breaks or tabs.**

Here are three different ways to do this:

def readFile(filename):

f = open(filename, 'r')

for line in f:

print(line)

f.close()

def readFile2(filename):

f = open(filename, 'r')

text = f.read()

print(text)

f.close()

def readFile3(filename):

with open(filename, 'r') as f:

text = f.read()

print(text)

I might avoid using the third solution on an actual exam (even though I prefer that version) as the “with” keyword has not been covered in class. I like it because you do not type a separate line that closes the file. You can learn more about file I/O and the benefits/cons of each of these solutions in the [Python 3 documentation](https://docs.python.org/3/tutorial/inputoutput.html).

**Make a set, list, and tuple containing the numbers 1, 2, and 3**

set: nums = {1,2,3}

list: nums = [1,2,3]

tuple: nums = 1,2,3

**What are the differences between these three?**

A set is unordered without duplicates

A list is ordered and can contain duplicates, it is also mutable

A tuple is like an immutable list

Get more info [here](https://docs.python.org/3/tutorial/datastructures.html)

**Sally loves Iguanas, Sam loves dogs, and Robert loves only himself. What data structure would be best to store this information? Use that data structure to store this information.**

This a good time to use dictionaries:

dictOfLove = {'Sally': 'Iguana', 'Sam': 'dog', 'Robert': 'Robert'}

**Write two functions: One that finds the sum of the entries in a list iteratively and one that finds the sum recursively.**

def sumIterative(myList):

total = 0

for number in myList:

total += number

return total

#Another way to do this (though the above way is easier to read):

def sumIterative2(myList):

i = 0

total = 0

while i < len(myList):

total += myList[i]

i += 1

return total

def sumRecurs(myList):

if len(myList) < 1: #This prevents error when an empty list is set as the parameter

return 0

if len(myList) == 1:

return myList[0]

else:

return myList[0] + sumRecurs(myList[1:]) #The first value in the list plus the sum of the rest of the list

def fib(n):

I took the idea for this problem from the [svpino](https://blog.svpino.com/2015/05/07/five-programming-problems-every-software-engineer-should-be-able-to-solve-in-less-than-1-hour) blog. There is not any extra solutions for this problem on there and I wouldn’t recommend the blog personally.

**Write a recursive function that returns the nth fibonacci number:**

def fib(n):

if n == 0:

return 0

elif n == 1:

return 1

else:

return fib(n-1) + fib(n-2)

If you want a lot of additional information on this problem, then go [here](http://stackoverflow.com/questions/494594/how-to-write-the-fibonacci-sequence-in-python)

**Find all the errors in the following code:**

*myList = [1,2,3]*

*for i in range(myList):* #You need to put a len() operator on myList

*if i = 1:* #Use == to compare values

*print “not again”* #Python 3 uses parentheses around the print statement

*else:*

*print(max(myList)* #Needs a parentheses to close it

If you got two of these then you are probably good