

Practice Final Exam

This “open-book” examination will last approximately 3 hours.

You **can** use any printed or written materials during the exam. You **cannot** use a calculator or any electronic materials, with the exception of the electronic version of the textbook and pdf notes. However, we advise you not to waste too much time leafing through your materials. Any use of the computer (other than for taking the exam itself and the e-book textbook and pdf notes) is prohibited. In particular, you cannot use a Python system, *ide.cs50.io* or any other website or application.

This exam contains 150 points; in order to properly budget your time, plan on spending approximately 1 minute per point. If you succeed in allocating your time like this, you may have 25 minutes or so left over to check your work. Note, however, some of you might find this exam challenging, so don't worry if you run out of time to check your work or if you can't completely finish solving all the problems.

Do not bother to "dummy-proof" your code (e.g., you need not check for valid input values) unless specifically asked to do so.

Show all of your work right here on the exam. If any question appears ambiguous or especially peculiar, ask for help! It is silly to misinterpret a question that you are capable of answering.

As always, *Best of luck!*

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]

Part A — 7 Multiple-Choice Questions

(7 points for each; 49 points total)

For each question in this part, clearly mark the correct answer in the appropriate space below.

1. The **random** module contains a function **random()** that generates a random *float* value uniformly in the semi-open range [0.0, 1.0) -- in other words, the value of **random()** will be ≥ 0.0 but strictly less than 1.0

Which one of the following expressions will yield a random integer value from 2 to 50 inclusive (in other words, including both 2 and 50 as possibilities)? Assume in all cases we have imported the *random* module via **from random import ***

- A. `int (random() * 50) - 2`
- B. `int (random() * 50) + 2`
- C. `int (random() * 49) + 2`
- D. `int (random() * 48) + 2`
- E. *none of the above will work!*

2. Consider the following code segment:

```
for k in range (3, n, 2):  
    print(k)
```

Which one of the following best characterizes the running time of the code?

- A. $O(\log n)$
- B. $O(n)$
- C. $O(n \log n)$
- D. $O(n^2)$
- E. $O(n!)$

3. Consider the following *recursive* method:

```
def mystery(x):  
    print (x % 10, end="")  
    if x // 10 != 0:  
        mystery (x//10)  
    print (x % 10, end="")
```

Which of the following is printed as a result of the method call **mystery (1234)** ?

- A. 1441
 - B. 3443
 - C. 12344321
 - D. 43211234
 - E. *Many digits are printed due to "infinite recursion."*
4. A *sorted* array of 121 integers is to be searched to determine whether or not the value **100** is in the list. Assuming that a *binary search* algorithm is used, what is the maximum number of elements that must be examined?

- A. 5
- B. 7
- C. 63
- D. 100
- E. 121

5. Consider the following horrible program:

```
def foo (she, who, whom):  
    print (who, "and", whom, "like", she)  
  
def main():  
    whom = "she"  
    who = "he"  
    it = "her"  
    # missing call on method foo goes here
```

Which one of the following replacements for the “missing call on method foo” would cause this main program to output the message

he and her like it

- A. foo ("it", who, it)
- B. foo (it, who, "it")
- C. foo (who, it, whom)
- D. *ALL of the above will work!*
- E. *none of the above!*

6. Which one of the following is a FALSE statement?

- A. A *class* is used to create objects that all have the same behavior.
- B. *Encapsulation* is the act of providing a public interface to users while hiding the implementation details. An object’s instance variables store the data required for executing its methods.
- C. Each object of a class has its own set of instance variables.
- D. A *mutator* method does not change the object on which it operates; it just returns the value of one or more instance variables.
- E. *None of the above! All are true!*

7. Consider the following function:

```
def prob7(a_list, num):  
    for k in range(len(a_list)-1, -1, -1):  
        if a_list[k] < num:  
            return k  
    return -1
```

Suppose **my_stuff** is properly initialized with a list of integer values (in no particular order) and that variable **n** has been initialized with an integer value.

Which one of the following best describes the contents of **my_stuff** after the following statement is executed?

```
m = prob7(my_stuff, n)
```

- A. All values in positions 0 through **m** are less than **n**.
- B. All values in positions **m+1** through the end of the array are **< n**.
- C. All values in positions **m+1** through the end of the array are **≥ n**.
- D. The smallest value is at position **m**.
- E. The largest value that is smaller than **n** is at position **m**.

Part B — 4 Short-Answer Questions (29 points total)

8. (5 points)

In one or two simple English sentences, explain why the two boolean expressions shown below return a different result from each other when typed into a *Python* interpreter:

```
>>> "IBM" < "Apple"  
False  
>>> "IBM" < "apple"  
True
```

9. Dictionary Reverse Lookup (12 points total, 6 points for each part)

Consider the following *dictionary* containing the favorite colors of certain TAs:

```
fav_colors = {'Ben': 'pink', 'Alyssa': 'blue', 'Nabib': 'red',  
              'Apekshya': 'blue', 'Tom': 'green'}
```

We wish to define a function named `reverse_lookup` that takes 2 parameters: a *dictionary*, and a value. The function should return a list containing those keys in the dictionary whose corresponding values are equal to the second parameter. For example:

```
reverse_lookup(fav_colors, 'blue') # should return with ['Apekshya', 'Alyssa']  
reverse_lookup(fav_colors, 'purple') # should return with []
```

Here is a partial solution with 2 pieces of code missing that you should write:

```
def reverse_lookup (dictionary, value):  
    answer = []  
    for ...      #replace ... with the missing code for the for loop  
  
        if ...  #replace ... with the missing code for the if statement  
  
    return answer
```

10. While Loop Simulation (12 points total, 4 points for each part)

For each call of the **mystery** function below, write the value that gets returned:

```
def mystery(x, y):  
    z = 0  
    while y > 0:  
        z = z + x  
        x = x + y  
        y -= 1  
    return z
```

Function Call**Output**

mystery(6, 0)

mystery(8, 1)

mystery(3, 3)

Part C — 4 Programming Questions (72 points total)

11. 20 points

Write a function named **cap(data, big)** that takes 2D list (i.e., a list of lists) as its first parameter, and a number as its second parameter.

Your function should replace any numbers stored in the list of lists (**data**) that are greater than the value passed in **big** with the value of **big**. Examples are shown in the table below using the 2D list named **values**. But note that the list passed as the first argument could be of any size. Note also that your function is allowed to modify the first argument (the list of lists).

```
values = [[18, 14, 29], [12, 7], [2, 22, 5]]
```

call	List of lists after the call
cap(values, 20)	[[18, 14, 20], [12, 7], [2, 20, 5]]
cap(values, 2)	[[2, 2, 2], [2, 2], [2, 2, 2]]
cap(values, 0)	[[0, 0, 0], [0, 0], [0, 0, 0]]

12. 20 points

Having a secure password is a very important practice, especially because much of our information is stored online. Write a program named **password.py** that validates a new password, following these rules:

- The password must be at least 8 characters long.
- The password must have at least one uppercase and one lowercase letter.
- The password must have at least one digit.

Write a program that asks for a password, then asks again to confirm it. If the passwords don't match or the rules are not fulfilled, prompt again. Suggestion: write a "helper" function **is_valid** that takes a password string and returns a boolean. Here is what your program might look like in action:

```
~/ $ python password.py
```

```
Type a password according to the rules: foobar123
INVALID PASSWORD, TRY AGAIN
```


Type a password according to the rules: **Foobar**
INVALID PASSWORD, TRY AGAIN

Type a password according to the rules: **Foobar123**
Now type your password again: **foobar123**
PASSWORDS DO NOT MATCH
Now type your password again: **Foobar123**
SUCCESS

13. 16 points

Implement a class **Car** with the following properties. A car has a certain fuel efficiency (measured in miles/gallon) and a certain amount of fuel in the gas tank. The car's efficiency is specified in the constructor, along with the capacity of the gas tank. Note that the initial fuel level is 0. The two instance variables should be named in such a way that they are "private" to the class.

Supply a method named **drive** that simulates driving the car for a certain distance, appropriately reducing the fuel level in the gas tank, along with accessor method **get_gas_level**, to return the current fuel level. The method **add_gas** allows the user to add however many gallons of gas to the car he or she wishes. *However*, an error should be printed if attempting to add more gas than the gas tank's capacity will allow.

Here is a sample usage:

```
my_hybrid = Car(50, 20)      #50 miles per gallon
my_hybrid.add_gas(14)        #Tank now has 14 gallons of gas
my_hybrid.drive(100)         #Drive 100 miles
print (my_hybrid.get_gas_level(), "gallons of gas remain")
```

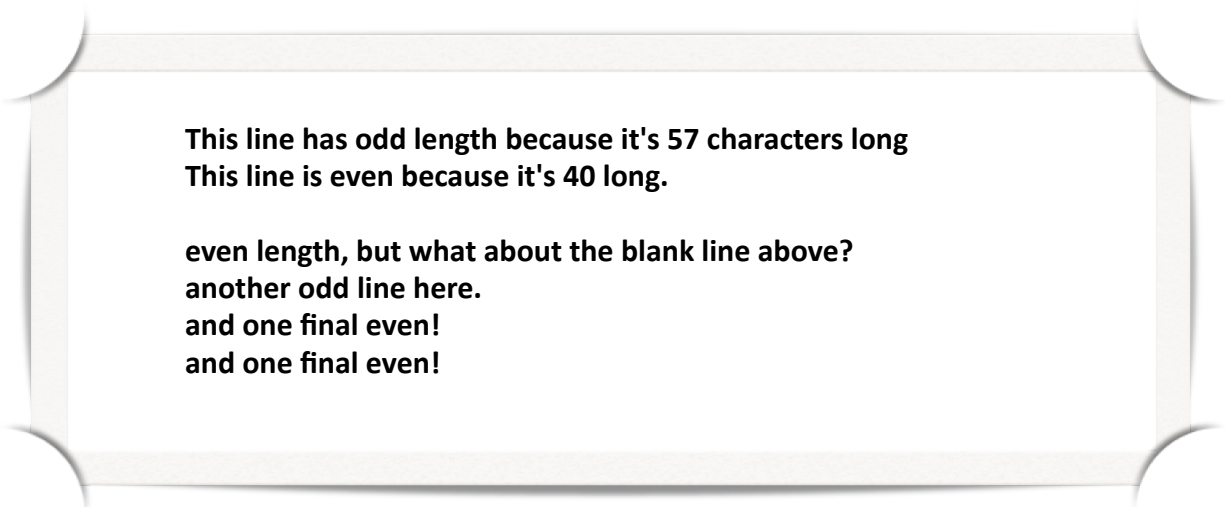
In the above example, my_hybrid was constructed as a car that averages 50 miles per gallon, with a gas tank capacity of 20 gallons. We added 14 gallons of gas to the tank, and then drove 100 miles. This consumed two gallons of gas, so the print statement should output

```
12 gallons of gas remain
```

14. 16 points

Write a program named **count_odd_length** that opens an input file that is given as the first command line argument, and then prints the total number of lines in the file, the total number of odd-length lines and the percentage of even-length lines.

For example, consider the following input file:



```
This line has odd length because it's 57 characters long
This line is even because it's 40 long.

even length, but what about the blank line above?
another odd line here.
and one final even!
and one final even!
```

Three of these input lines have odd length and there are 7 lines total, so your method should print the following output:

```
Total lines = 7
Lines of ODD length = 3
% lines of EVEN length = 57.14285714285714
```

The function **count_odd_length** should exactly reproduce the format of this output. You may assume that the input file contains at least one line. Note that the '\n' character at the end of each line should be counted as a legitimate character.

A partial solution to **count_odd_length** appears below. Fill in the 4 missing expressions denoted by the red ??? to make the method work correctly. The correct answer to each missing expression is worth 4 points.

```
from ??? import argv

def count_odd_length():

    input_file = open ( ??? )

    odd = 0

    total = 0

    for line in input_file:

        total += 1

        if ??? :

            odd += 1

    print ("Total lines =", total)

    print ("Lines of ODD length =", odd)

    print ("% lines of EVEN length = ", ??? )
```