**Practice Quiz: Understanding the Problem**

**TOTAL POINTS 5**

1.Question 1

When a user reports that an "application doesn't work," what is an appropriate follow-up question to gather more information about the problem?

1 point



Is the server plugged in?



Why do you need the application?



Do you have a support ticket number?



What should happen when you open the app?

2.Question 2

What is a heisenbug?

1 point



The observer effect.



A test environment.



The root cause.



An event viewer.

3.Question 3

The compare\_strings function is supposed to compare just the alphanumeric content of two strings, ignoring upper vs lower case and punctuation. But something is not working. Fill in the code to try to find the problems, then fix the problems.

import re

def compare\_strings(string1, string2):

  #Convert both strings to lowercase

  #and remove leading and trailing blanks

  string1 = string1.lower().strip()

  string2 = string2.lower().strip()

  #Ignore punctuation

  punctuation = r"[.?!,;:-']"

  string1 = re.sub(punctuation, r"", string1)

  string2 = re.sub(punctuation, r"", string2)

  #DEBUG CODE GOES HERE

  print(\_\_\_)

  return string1 == string2

print(compare\_strings("Have a Great Day!", "Have a great day?")) # True

print(compare\_strings("It's raining again.", "its raining, again")) # True

print(compare\_strings("Learn to count: 1, 2, 3.", "Learn to count: one, two, three.")) # False

print(compare\_strings("They found some body.", "They found somebody.")) # False

Solution:

4.Question 4

How do we verify if a problem is still persisting or not?

1 point



Restart the device or server hardware



Attempt to trigger the problem again by following the steps of our reproduction case



Repeatedly ask the user



Check again later

5.Question 5

The datetime module supplies classes for manipulating dates and times, and contains many types, objects, and methods. You've seen some of them used in the dow function, which returns the day of the week for a specific date. We'll use them again in the next\_date function, which takes the date\_string parameter in the format of "year-month-day", and uses the add\_year function to calculate the next year that this date will occur (it's 4 years later for the 29th of February during Leap Year, and 1 year later for all other dates). Then it returns the value in the same format as it receives the date: "year-month-day".

Can you find the error in the code? Is it in the next\_date function or the add\_year function? How can you determine if the add\_year function returns what it's supposed to? Add debug lines as necessary to find the problems, then fix the code to work as indicated above.

import datetime

from datetime import date

def add\_year(date\_obj):

  try:

    new\_date\_obj = date\_obj.replace(year = date\_obj.year + 1)

  except ValueError:

    # This gets executed when the above method fails,

    # which means that we're making a Leap Year calculation

    new\_date\_obj = date\_obj.replace(year = date\_obj.year + 4)

  return new\_date\_obj

def next\_date(date\_string):

  # Convert the argument from string to date object

  date\_obj = datetime.datetime.strptime(date\_string, r"%Y-%m-%d")

  next\_date\_obj = add\_year(date\_obj)

  # Convert the datetime object to string,

  # in the format of "yyyy-mm-dd"

  next\_date\_string = next\_date\_obj.strftime("yyyy-mm-dd")

  return next\_date\_string

today = date.today()  # Get today's date

print(next\_date(str(today)))

# Should return a year from today, unless today is Leap Day

print(next\_date("2021-01-01")) # Should return 2022-01-01

print(next\_date("2020-02-29")) # Should return 2024-02-29

Solution: