In class Practice 2/21/2020

Part I

1. Which one of the following is ***not*** a benefit of using a ***with*** statement to open a file?

   a. The file is automatically closed.
   b. This file is closed even if an exception occurs while the file is being processed.
   c. The resources used by the file are released when the file is closed.
   d. You don't have to specify the path for the file.

2. To read the rows in a CSV file, you need to

   a. get a reader object by using the reader() function of the file object
   b. get a reader object by using the reader() function of the csv module
   c. get a row object by using the row() function of the file object
   d. get a rows object by using the rows() function of the file object

3. The finally clause of a try statement

   a. is required
   b. is executed whether or not an exception has been thrown
   c. can be used to display more information about an exception
   d. can be used to recover from an exception

4. To throw an exception with Python code, you use the

   a. throw statement
   b. raise statement
   c. built-in throw() function
   d. build-in raise() function

5. A recursive algorithm that uses two recursive calls to split into two directions is known as tree recursion. When using tree recursion,

   a. the code runs extremely efficiently
   b. the number of recursive calls grows exponentially
   c. Python never calculates a branch of the tree more than once
   d. there can only be two branches of the tree

6. Given a class named Customer, which of the following creates a Customer object and assigns it to the variable named cust1:

   a. `cust1 = new Customer()`
   b. `cust1 = Customer()`
   c. `cust1 = Customer.init()`
   d. `cust1 = Customer.create()`

7. When you code the dot operator after an object, what can you access?

   a. the constructor of the object
   b. the iterator of the object

c. the public attributes and methods of the object
d. only the public methods of the object

8. If you have a class named Vehicle, and you want to code a class named Truck that inherits the Vehicle class, you can begin by writing this code:

a. `class Truck(Vehicle):`
b. `class Truck : Vehicle`
c. `class Truck inherits Vehicle:`
d. `class Truck extends Vehicle:`

Part II Short Answers

```
import csv
def main():
    courses = [["Python", 3],
               ["Trig", 3],
               ["Physics", 4],
               ["Yoga", 2]]
    with open("courses.csv", "w", newline="") as file:
        writer = csv.writer(file)
        writer.writerows(courses)
    course_list = []
    with open("courses.csv", newline="") as file:
        reader = csv.reader(file)
        for row in reader:
            course_list.append(row)
    for i in range(len(course_list) - 2):
        course = course_list[i]
        print(course[0] + " (" + str(course[1]) + ")")

main()
```

1. What happens if the courses.csv file doesn't exist when the first with open statement is executed?
2. If the first with open statement works, what is written to the file?
3. What will display on the console after the code executes?

1.A new file named courses.csv is created.
2.The list named courses.

3.Python (3)
Trig(3)

Part III

1. Write an abstract class A with two abstract methods a1() and a2(). Create two subclasses B and C from class A which implement these two methods. The following screen shots show the test program and its output.

```
aces = [B('Jone'), B('Jane'), C('Paul'), C('Jame')]

for i in aces:
    print (i.name,  ': ' , i.a1(), ' and ', i.a2())
```

```
Jone :  Jone in B running a1!  and  Jone in B running a2!
Jane :  Jane in B running a1!  and  Jane in B running a2!
Paul :  Paul in C running a1!  and  Paul in C running a2!
Jame :  Jame in C running a1!  and  Jame in C running a2!
```

```
class A(object):
    def __init__(self,name):
        self._name = name

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self,name):
        self._name = name

    def a1(self):
        raise NotImplementedError("Subclass must implement abstract method")

    def a2(self):
        raise NotImplementedError("Subclass must implement abstract method")

class B(A):
    def a1(self):
        return self.name+ " in B running a1!"

    def a2(self):
        return self.name+ " in B running a2"

class C(A):
    def a1(self):
        return self.name+ " in C running a1!"

    def a2(self):
        return self.name+ " in C running a2"
```

2. For the following Fibonacci function definition, write a decorator call it my_fib_decorator which will use a dict to memorize the Fibonacci number to speed up the process.

```
@my_fib_decorator
def fib(n):
    if n < 2:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

```
def my_fib_decorator(fn):
    memory = {}

    def fast_fib(param):
        if param in memory:
            return memory[param]
        else:
            val = fn(param)
            memory[param]=val
            return val
    return fast_fib

@my_fib_decorator
def fib(n):
    if n==1 or n==0:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```