In class Practice 2/14/2020

Part I

1. Which method provides a customized way to represent the state of an object as a string?
   a. _ _ str _ _
   b. _ _ init _ _
   c. _ _ print _ _
   d. print

2. Methods in a class have *self* as their _____ parameter.
   a. first
   b. last
   c. default
   d. only

3. Class definitions can contain _____ methods.
   a. an unlimited number of
   b. only accessor and mutator
   c. one constructor, two accessor and two mutator
   d. a maximum of 10

4. The statement

   object = className(arg1, arg2, …)

   is said to _____ an object.
   a. instantiate
   b. inherit
   c. encapsulate
   d. polymorphize

5. When using inheritance to define a new class, the new class is called a _____.
   a. subclass
   b. child class
   c. derived class
   d. all of the above

6. If a method defined in the subclass has the same name as a method in its superclass, the child's method will _____.
   a. override the parent's method

b. be ignored
c. cause a Throwback error
d. confuse the programmer

7. The _ _ init _ _ method must be explicitly called by the programmer when a new object is created.
   a. false
   b. true

8. The *isinstance* function can be applied to both built-in and user-defined functions.
   a. true
   b. false

Part II Short Answers

1. Explain the purpose of the *self* parameter in a method that is defined in a class.
2. Give two reasons why instance variables should only be accessed from outside of a class definition via class methods?
3. What does the following line of Python code return?

   isinstance([ ], list)

4. What does the following line of Python code return?

   isinstance({ }, set)

1. This parameter references the object created so that the method know which object to operate on.
2.a) validity-checking code can be inserted into methods to make programs more robust.
b)one of the objectives of oop is to hide implementation details from interface.
3.True
4.False

Part III

1. Consider the following class PrivateClass. The constructor __init__ intends for __private_date to be private, which is why it is preceded by two underscores.

```
# private.py
"""Class with public and private attributes."""

class PrivateClass:
    """Class with public and private attributes."""

    def __init__(self):
        """Initialize the public and private attributes."""
        self.public_data = "public"  # public attribute
        self.__private_data = "private"  # private attribute
```

Create an object called it *obj1* and then use *obj1* to output its public_data, and __private_data.

```
obj1 = PrivateClass():
print(obj.public_data)
print(obj._PrivateClass__private_dat)
```

2. Here is a Time class definition from Deitel's textbook.

```
class Time:
    """Class Time with read-write properties."""

    def __init__(self, hour=0, minute=0, second=0):
        """Initialize each attribute."""
        self.hour = hour  # 0-23
        self.minute = minute  # 0-59
        self.second = second  # 0-59

    @property
    def hour(self):
        """Return the hour."""
        return self._hour

    @hour.setter
    def hour(self, hour):
        """Set the hour."""
        if not (0 <= hour < 24):
            raise ValueError(f'Hour ({hour}) must be 0-23')

        self._hour = hour
    @property
    def minute(self):
        """Return the minute."""
        return self._minute

    @minute.setter
    def minute(self, minute):
        """Set the minute."""
        if not (0 <= minute < 60):
            raise ValueError(f'Minute ({minute}) must be 0-59')
        self._minute = minute

    @property
    def second(self):
        """Return the second."""
        return self._second

    @second.setter
    def second(self, second):
        """Set the second."""
        if not (0 <= second < 60):
            raise ValueError(f'Second ({second}) must be 0-59')
        self._second = second
```

```
    def set_time(self, hour=0, minute=0, second=0):
        """Set values of hour, minute, and second."""
        self.hour = hour
        self.minute = minute
        self.second = second

    def __repr__(self):
        """Return Time string for repr()."""
        return (f'Time(hour={self.hour}, minute={self.minute}, ' +
                f'second={self.second})')

    def __str__(self):
        """Return Time string in 12-hour clock format."""
        return (('12' if self.hour in (0, 12) else str(self.hour % 12)) +
                f':{self.minute:0>2}:{self.second:0>2}' +
                (' AM' if self.hour < 12 else ' PM'))
```

Study the above Time class and then write a similar Python class called it ***Invoice***. Class ***Invoice*** contains four pieces of attributes: a part number (type str), a description (type str), quantity (type int) and price per item (type Decimal). Your __init__ will initialize these four attributes. For each attribute, there is a getter and a setter. Note you need to raise error when the price to set is less than 0. Similar for quantity. Also def a method calculate_invoice which return quantity * price.

NOTE: The **@property decorator** precedes the property's *getter* method, which receives only a self parameter. A decorator adds code to the decorated function, the *getter* method's name is the property name

A decorator of the form **@property_name.setter** (for example, *@hour.setter* if the property name is *hour*) precedes the property's *setter* method. It receives two parameters—self and a parameter (in our example *hour*) representing the value being assigned to the property.

```
from decimal import Decimal
class Invoice:
    def __init__(self,part_numer,part_description,quantity,price):
        self.part_number =part_number
        self.part_description = part_description
        self.quantity = quantity
        self.price = price

    @property
    def part_number(self):
        return self.part_number

    @part_number.setter
    def part_number(self, part_number):
        self.part_number = part_number

    @property
    def part_description(self):
        return self.part_description

    @part_description.setter
    def part_description(self, part_description):
        self.part_description = part_description


    @property
    def quantity(self):
        return self.quantity

    @quantity.setter
    def quantity(self, quantity):
        self.quantity = quantity

    @property
    def price(self):
        return self.price

    @price.setter
    def price(self, price):
        self.price = price

    def calculate_invoice(self):
        return self.quantity*self.price

    def __repr__(self):
        return (f"Invoice(part_number = (self.part_number),part_description = (self.part_description)," +
                f"quantity = (self.quantity),price = (self.price))")
```