COMP 651 Data Analytics Techniques Assignment 4

10 points possible

For this assignment, you will write two Python modules. One, **`PersonData`**, contains the definition of class **`PersonData`** as well as a class derived from it, **`EmployedPersonData`**. An instance of **`PersonData`** records the age, number of children, and spouse of a person and has methods to update and access this information. The other module, **`Population`**, contains the definition of class **`Population`**. An instance of this class has a dictionary of name-data pairs, where the data is an instance of **`PersonData`**. It also has methods to update and access this information.

Class **`PersonData`** has object properties **`age`** (age in years), **`children`** (number of children), and **`spouse`** (spouse's name, a string). The default for **`children`** is 0, and the default for **`spouse`** is **`None`**. If a person has no spouse, then the corresponding **`PersonData`** object has no **`spouse`** attribute. An instance of **`PersonData`** converts to a string of the form

"A ⟨age⟩ year old person",

where ⟨age ⟩ is the value of **`self.age`**.

None of the non-special methods return anything; except for the method that outputs the attribute values, they update attribute values or add or delete the **`spouse`** attribute. Only the **`marries`** non-special method is passed an argument (the name of the new spouse).

Method **`birthday`** increments the **`age`** attribute, and method **`new_child`** increments the **`children`** attribute.

Method **`print_data`** takes no argument and does not return anything. It outputs the values of the **`age`** and **`children`** attributes; if the **`spouse`** attribute exists, it outputs that as well, otherwise it outputs **`None`**. Reserve two character positions for integer values that are the values of the **`age`** and **`children`** attributes, and leave plenty of space between the values.

Method **`marries`** is passed the name of the new spouse and adds to **`self`** a **`spouse`** attribute with that name as its value unless **`self`** already has a **`spouse`** attribute. In that case, it raises an **`Exception`** exception with a string argument of the form

"Spouse exists: ⟨spouse⟩"

where ⟨spouse⟩ is the value of **`self.spouse`**.

Method **`divorces`** deletes **`self`**'s **`spouse`** attribute unless this attribute does not exist, in which case it raises an **`AttributeError`** exception with a string argument of the form

"Not married, divorce impossible"

Class **`EmployedPersonData`** inherits from **`PersonData`** and adds an **`employer`** object property, whose value is a string (the name of the employer) and is initialized (along with the inherited properties) when **`EmployedPersonData()`** is called. It overrides the **`__str__()`** method so that an instance of **`EmployedPersonData`** converts to a string of the form

"A ⟨age⟩ year old person employed by ⟨employer⟩"

where ⟨age ⟩ is the value of **`self.age`** and ⟨employer⟩ is the value of **`self.employer`**. It also overrides the **`print_data()`** method so that (besides all the values printed with an instance of **`PersonData`**) it also prints the value of **`self.employer`**.

Include in your **`PersonData`** module the following code (available as **`test_code1.py`** on the assignment page) to be executed when the module is executed as a script.

```
if __name__ == "__main__":
   p1 = PersonData(31, spouse="Sue", children=2)
   print(p1)
   p1.print_data()
   p1.birthday()
   p1.new_child()
   p1.divorces()
   p1.print_data()
   p1.marries("Pam")

   try:
      p1.marries("Sue")
   except Exception as detail:
      print(detail)

   try:
      p1.divorces()
      p1.divorces()
   except AttributeError as detail:
      print(detail)

   p2 = PersonData(39)
   p2.print_data()
   ep1 = EmployedPersonData(31, "Google", spouse="Tom", children=3)
   print(ep1)
   ep1.print_data()
```

When your **`PersonData`** module is executed as a script, it should produce the following output.

```
A 31 year old person
31      2     Sue
32      3     None
Spouse exists: Pam
Not married, divorce impossible
39      0     None
A 31 year old person employed by Google
31      3     Google     Tom
```

Hints

Where **`o`** is an object and **`a`** is an attribute,

```
del o.a
```

removes attribute **`a`** from **`o`**, and

```
hasattr(o, "a")
```

returns **`True`** if **`o`** has attribute **`a`** (and **`False`** otherwise).

Turning to module **`Population`**, this imports module **`PersonData`** and defines class **`Population`**. The sole **`Population`** object attribute is **`people`**, a dictionary whose keys are persons' names and values are **`PersonData`** instances for the corresponding persons. This property starts out with the empty dictionary. An instance of **`Population`** converts to a string of the form

"A population with ⟨num⟩ people"

where ⟨num⟩ is the number of key-value pairs in **`self.people`**.

The only non-special method that returns something is **`len`**, which takes no argument and returns the number of people recorded in **`self.people`**.

Method **`add_person`** is used to add a key-value pair to the **`self.people`** dictionary. It is passed the person's name and age and optionally their number of children (defaulting to 0) and spouse's name (defaulting to **`None`**). It creates an instance of **`PersonData`** associated in the **`people`** dictionary with the name provided unless that name is already in the dictionary. In that case, it raises a **`NameError`** exception with a string argument of the form

"⟨name⟩ already present"

where ⟨name⟩ is the value of the name passed in.

Method **`remove_person`** is passed a name and removes the key-value pair from the **`people`** dictionary with that name as key as long as there is such a pair. If not, it raises a **`NameError`** exception with a "⟨name⟩ not present" string argument.

Method **`print_pop`** outputs a table with the information on all entries in the **`people`** dictionary. The column labels are "name", "age", "children", and "spouse". It invokes the **`print_data`** method on each value in the **`people`** dictionary.

To test this module, include code executed only when **`__name__ == "__main__"`** is true. In the test code, do the following.

- Create a (empty) population
- Add Ed, age 47, 2 children, spouse Pam.
- Add Pam, age 51, 2 children, spouse Ed.
- Execute a **`print()`** with the instance as argument (illustrating **`__str__()`**).
- Print the population.
- Print the number of persons.
- Remove Pam.
- Remove Ed.
- Print the number of persons.
- Try to remove Ed again and catch the exception.
- Add Ed, age 45.
- Try to add Ed (age 45) again and catch the exception.
- Print the number of persons.

When your **`Population`** module is executed as a script, it should produce the following output.

```
A population with 2 people
Name       Age  Children Spouse
Ed         47       2     Pam
Pam        51       2     Ed
```

```
There are 2 persons recorded.
There are now 0 persons recorded.
remove_person: Ed not present
add_person: Ed already present
There are 1 persons recorded.
```

In both modules, include docstrings throughout. Regarding docstrings, see the following.

PEP 257 -- Docstring Conventions :
http://www.python.org/dev/peps/pep-0257/
("PEP" stands for "Python Enhancement Proposal")

Example Google Style Python Docstrings:
http://sphinxcontrib-napoleon.readthedocs.org/en/latest/example_google.html
(Python is the principal scripting language used by Google.)

Google Python Style Guide (heading "Doc Strings"):
http://google-styleguide.googlecode.com/svn/trunk/pyguide.html

Zip together your two Python module files, and submit the **`.zip`** file.