Python Code Reading Recitation B

Functions [1/7]

```python
#  8-1 greeter.py

def greet_user(username):
    """Display a simple greeting."""
    print("Hello, " + username.title() + "!")

greet_user('jesse')
```

```python
# 8-2 pets.py

def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")

# A dog named Willie.
describe_pet('willie')
describe_pet(pet_name='willie')

# A hamster named Harry.
describe_pet('harry', 'hamster')
describe_pet(pet_name='harry', animal_type='hamster')
describe_pet(animal_type='hamster', pet_name='harry')
```

Functions [2/7]

```python
# 8-3 formatted_name.py

def get_formatted_name(first_name, last_name, middle_name=''):
    """Return a full name, neatly formatted."""
    if middle_name:
        full_name = first_name + ' ' +  middle_name + ' ' + last_name
    else:
        full_name = first_name + ' ' + last_name
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)

musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
```

Functions [3/7]

```python
# 8-4 peson.py


def build_person(first_name, last_name, age=''):
    """Return a dictionary of information about a person."""
    person = {'first': first_name, 'last': last_name}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', age=27)
print(musician)
```

```python
# 8-5 greet_users.py

def greet_users(names):
    """Print a simple greeting to each user in the list."""
    for name in names:
        msg = "Hello, " + name.title() + "!"
        print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```

Functions [4/7]

```python
# 8-6 printing_models.py


def print_models(unprinted_designs, completed_models):
    """
    Simulate printing each design, until there are none left.
    Move each design to completed_models after printing.
    """
    while unprinted_designs:
        current_design = unprinted_designs.pop()

        # Simulate creating a 3d print from the design.
        print("Printing model: " + current_design)
        completed_models.append(current_design)

def show_completed_models(completed_models):
    """Show all the models that were printed."""
    print("\nThe following models have been printed:")
    for completed_model in completed_models:
        print(completed_model)


unprinted_designs = ['iphone case', 'robot pendant', 'dodecahedron']
completed_models = []

print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
```

Functions [5/7]

```python
# 8-7 pizza.py

def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print("\nMaking a " + str(size) +
          "-inch pizza with the following toppings:")
    for topping in toppings:
        print("- " + topping)

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Functions [6/7]

```python
# 8-8 user_profile.py

def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know about a user."""
    profile = {}
    profile['first_name'] = first
    profile['last_name'] = last
    for key, value in user_info.items():
        profile[key] = value
    return profile

user_profile = build_profile('albert', 'einstein',
                             location='princeton',
                             field='physics')
print(user_profile)
```

Functions [7/7]

```python
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print("\nMaking a " + str(size) +
          "-inch pizza with the following toppings:")
    for topping in toppings:
        print("- " + topping)
```

```python
# 8-9 making_pizzas.py

import pizza as p

p.make_pizza(16, 'pepperoni')
p.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Class Code [1/7]

```python
# 9-1 dog.py

class Dog():
    """A simple attempt to model a dog."""

    def __init__(self, name, age):
        """Initialize name and age attributes."""
        self.name = name
        self.age = age

    def sit(self):
        """Simulate a dog sitting in response to a command."""
        print(self.name.title() + " is now sitting.")

    def roll_over(self):
        """Simulate rolling over in response to a command."""
        print(self.name.title() + " rolled over!")


my_dog = Dog('willie', 6)
your_dog = Dog('lucy', 3)

print("My dog's name is " + my_dog.name.title() + ".")
print("My dog is " + str(my_dog.age) + " years old.")
my_dog.sit()

print("\nMy dog's name is " + your_dog.name.title() + ".")
print("My dog is " + str(your_dog.age) + " years old.")
your_dog.sit()
```

Class Code [2/7]

```python
# 9-2 car.py

"""A class that can be used to represent a car."""

class Car():
    """A simple attempt to represent a car."""

    def __init__(self, manufacturer, model, year):
        """Initialize attributes to describe a car."""
        self.manufacturer = manufacturer
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = str(self.year) + ' ' + self.manufacturer + ' ' + self.model
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles
```

Class Code [3/7]

```
my_used_car = Car('subaru', 'outback', 2013)
print(my_used_car.get_descriptive_name())

my_used_car.update_odometer(23500)
my_used_car.read_odometer()

my_used_car.increment_odometer(100)
my_used_car.read_odometer()
```

Class Code [4/7]

```python
# 9-3 electric_car.py


"""A set of classes that can be used to represent electric cars."""

from car import Car

class Battery():
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=60):
        """Initialize the batteery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print("This car has a " + str(self.battery_size) + "-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 60:
            range = 140
        elif self.battery_size == 85:
            range = 185

        message = "This car can go approximately " + str(range)
        message += " miles on a full charge."
        print(message)


class ElectricCar(Car):
    """Models aspects of a car, specific to electric vehicles."""

    def __init__(self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(manufacturer, model, year)
        self.battery = Battery()
```

Class Code [5/7]

```
my_tesla = ElectricCar('tesla', 'model s', 2016)

print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
```

Class Code [6/7]

```python
# 9-4 my_car.py

from car import Car

my_new_car = Car('audi', 'a4', 2015)
print(my_new_car.get_descriptive_name())

my_new_car.odometer_reading = 23
my_new_car.read_odometer()

```

```python
# 9-5 my_cars.py

from car import Car
from electric_car import ElectricCar

my_beetle = Car('volkswagen', 'beetle', 2015)
print(my_beetle.get_descriptive_name())

my_tesla = ElectricCar('tesla', 'roadster', 2015)
print(my_tesla.get_descriptive_name())

```

Class Code [7/7]

```python
# 9-6 favorite_languages.py

from collections import OrderedDict

favorite_languages = OrderedDict()

favorite_languages['jen'] = 'python'
favorite_languages['sarah'] = 'c'
favorite_languages['edward'] = 'ruby'
favorite_languages['phil'] = 'python'

for name, language in favorite_languages.items():
    print(name.title() + "'s favorite language is " +
        language.title() + ".")
```