Object Oriented Programming

1. Creating a class

To get started, let us write a class Student together.

```
class Student:
    # Constructor / Initializer
    # name should be stored publicly;
    # age should be stored publicly;
    # gpa should be stored privately.
    def __init__(self, name, age, gpa):
        # Your code
        self.name = name
        self.age = age
        self.__gpa = gpa

    # For private variables we need getters/setters. By convention.
    def get_gpa(self):
        # Your code
        return self.__gpa

    def set_gpa(self, gpa):
        # Your code
        self.__gpa = gpa
```

2. Built-In Methods

Let's take a look at the built-in methods for the `List` class.

Find out how the built-in methods on the left translate in terms of operators.

You can find a table with all the overload operators on p.75 of the textbook.

`X.__getitem__(self, index)`	`X[index]`
`X.__setitem__(self, index, value)`	`X[index]=value`
`X.__delitem__(self, index)`	`del X[index]`
`X.__add__(self, other)`	`X + other`
`X.__iadd__(self, other)`	`X += other`
`X.__eq__(self, other)`	`X == other`
`X.__len__(self)`	`len(X)`
`X.__str__(self)`	`str(X)`
`X.__contains__(self, value)`	`value in X`

3. Operator overload

Consider the following program:

```
class Pizza:
    def __init__(self, price):
        self.price = price

    def __add__(self, other):        # Overload + operator
        new_pizza = Pizza(self.price)
        new_pizza += other
        return new_pizza

    def __iadd__(self, other):       # Overload += operator
        self.price += other.price
        return self

     def __str__(self):
        return "the price is " + str(self.price)

def main():
    pizza1 = Pizza(5)
    pizza2 = Pizza(6)
    pizza1 += pizza2
    print(pizza1)
main()
```

What does this program display upon its execution?

```
the price is 11
```

4. Inheritance

Consider the following program:

```
class Tree:

    def __init__(self, name, age):
        self._name = name
        self._age = age

    def get_name(self): return self._name

class Palm(Tree):
    def __init__(self, name, age, color):
        super().__init__(name, age)
        self._color = color

    def get_color(self):
        return self._color

def main():
    palm1 = Palm("Lucky", 30, "green")
    print(palm1.get_name())      #Display 1
    print(palm1.get_color())     #Display 2
    tree1 = Tree("Funny", 20)
    print(tree1.get_name())      #Display 3
    print(tree1.get_color())     #Display 4
main()
```

What does this program display upon its execution?

Display 1: Lucky

Display 2: green

Display 3: Funny

Display 4: AttributeError