

CSCI-SHU 210 Data Structures

Recitation 1 Object-Oriented Programming Review

You have a series of tasks in front of you. Complete them! Everyone should code on their own computer, but you are encouraged to talk to others, and seek help from each other and from the TA/LAs.

“_____” means this is your task to complete!

Important topics for today:

- OOP
 - Please watch the videos as mentioned in our OOP review slides.
 - Why code in OOP
 - OOP syntax
 - Keyword “self” meaning
 - Operator overloading (___**__ functions)
 - Basic inheritance
- Misc Python topics
 - The purpose of if `__name__ == “__main__”`
 - The purpose of “__slots__”
 - “yield” keyword in Python and generator in Python

Topic 1 (Creating a class):

Let us create **class Student** together.

```
1. class Student:
2.     # Constructor / Initializer
3.     # name should be stored publicly;
4.     # age should be stored publicly;
5.     # GPA should be stored privately. (By convention)
6.     def __init__(self, name, age, GPA):
7.         # Your code
8.         _____
9.
10.        _____
11.
12.        _____
13.
14.    # For private variables we need getters/setters. By convention.
15.    def get_GPA(self):
16.        # Your code
17.        _____
18.
19.    def set_GPA(self, GPA):
20.        # Your code
21.        _____
```

What does keyword self do in Python?

Topic 2 (underscore ***** functions):

Python has some functions, by convention, has some other representation to make our coding easier. Those function names are reserved and looks like `*****`(self).

In this course, you will encounter some “underscore” functions much `__init__`, `__len__`, `__str__`, `__delitem__`, `__setitem__`, `__getitem__`, `__iadd__`, `__add__` etc.

****In Java/C++, those functions are called *operator overloading*.**

For example:

```
1. class Pizza:
2.     def __init__(self, price):
3.         self.price = price
4.
5.     def __add__(self, other):          # Overload + operator
6.         new_pizza = Pizza(self.price)
7.         new_pizza += other
8.         return new_pizza
9.
10.    def __iadd__(self, other):          # Overload += operator
11.        self.price += other.price
12.        return self
13.
14.    def __str__(self):
15.        return "the price is, " + str(self.price)
16.
17. def main():
18.     pizza1 = Pizza(5)
19.     pizza2 = Pizza(6)
20.     pizza1 + pizza2
21.     pizza1 += pizza2
22.     print(pizza1)
23.
24. main()
```

a) What does the code above print? Don't run the program, try to predict the output first.

What is the output for code above?

b) Complete the following table, suppose the variable name is X. When will these underscore functions get called? Answer for 1st row has been given for your convenience.

“Underscore” function	other representation(Implicit call)
X.__getitem__(self, index)	X[index]
X.__setitem__(self, index, value)	
X.__delitem__(self, index)	
X.__add__(self, other)	
X.__iadd__(self, other)	
X.__eq__(self, other)	
X.__len__(self)	
X.__str__(self)	

```
X.__repr__(self)
X.__contains__(self, value)
X.__iter__(self)
```

Topic 3 (Inheritance):

Inheritance and the “Is a” relationship. A palm is a tree.

```
1. class Tree:
2.     def __init__(self, name, age):
3.         self._name = name
4.         self._age = age
5.
6.     def get_name(self):
7.         return self._name
8.
9. class Palm(Tree): # Palm(Tree) means, Palm inherits Tree.
10.    def __init__(self, name, age, color):
11.        # First you have to initialize the parent class. What should we write here?
12.
13.        _____
14.        self._color = color
15.
16.    def get_color(self):
17.        return self._color
18.
19. def main():
20.     palm1 = Palm("Lucky", 30, "green")
21.     print(palm1.get_name())    # What does this print (1)?
22.     print(palm1.get_color())  # What does this print (2)?
23.     tree1 = Tree("Funny", 20)
24.     print(tree1.get_name())   # What does this print (3)?
25.     print(tree1.get_color()) # What does this print (4)?
26.
27.
28. main()
```

What does the code above print? Don't run the program, try to predict the output first.

What is the output for print (1)?

What is the output for print (2)?

What is the output for print (3)?

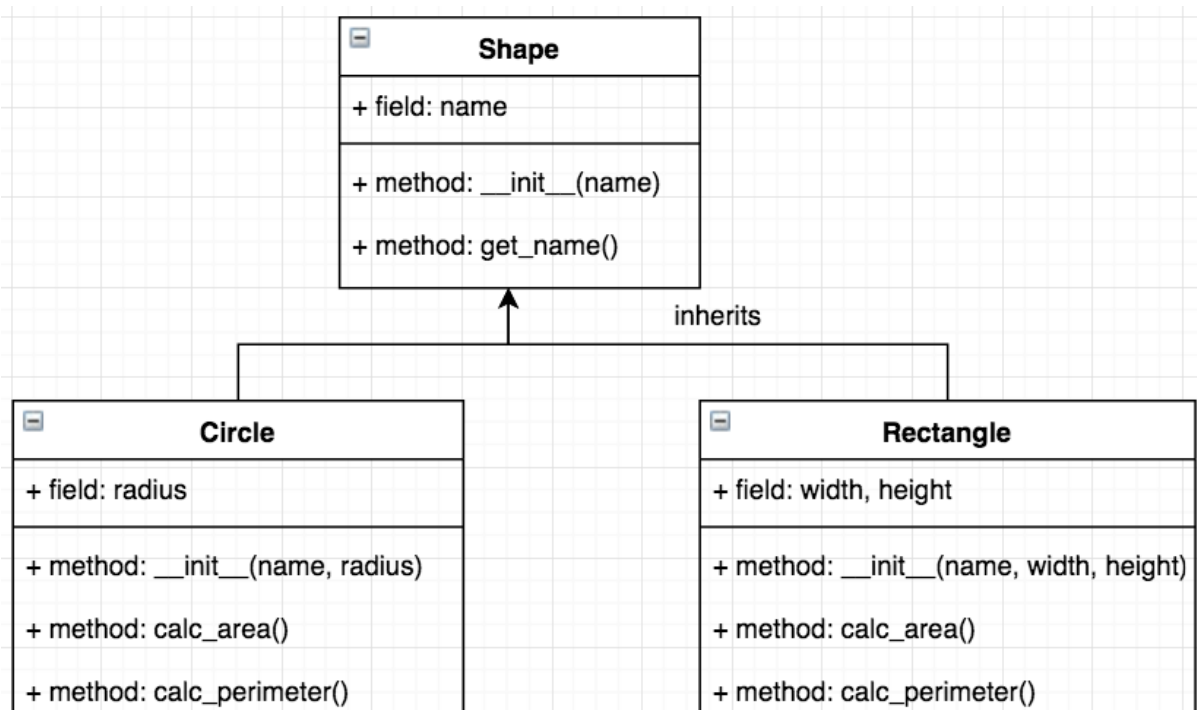
What is the output for print (4)?

Topic 4 (Misc):

Coding 1 (Creating a class using UML Diagram, and then use this class.):

Now implement the following class hierarchy: **Shape**, **Circle**, and **Rectangle**.

Note: Both Circle, Rectangle inherits Shape. Once finished, test your **Shape**, **Circle**, and **Rectangle** and see if they work correctly.



This problem is challenging, try this problem only when you are already confident with previous problems.

Coding 2 (Creating a class using example calls):

Design and implement class Polynomial. It should behave like the following way:

```
1. class Polynomial:
2.     def __init__(self, coeffs):
3.         self.coeffs = coeffs
4.         ##### To do #####
5.
6.
7.
8. def main():
9.     # 1x^4 + 2x^3 + 3x^2 + 4x + 5
10.    coeffs = [1,2,3,4,5]
11.    poly = Polynomial(coeffs)
12.    print(poly.evaluate_at(2))    # Outputs: 57
13.    print(poly.evaluate_at(3))    # Outputs: 179
14.    print(poly) # Outputs: 1x^4 + 2x^3 + 3x^2 + 4x^1 + 5
15.
16.    # 4x^3 + 6x^2 + 8x^1 + 10
17.    coeffs = [4,6,8,10]
18.    poly2 = Polynomial(coeffs)
19.    print(poly2)    # Outputs: 4x^3 + 6x^2 + 8x^1 + 10
20.    poly += poly2
21.    print(poly)    # Outputs: 1x^4 + 6x^3 + 9x^2 + 12x^1 + 15
```

Suppose I want to represent $1x^4 + 2x^3 + 3x^2 + 4x + 5$, then I input `python list` `[1,2,3,4,5]` into class Polynomial, my Polynomial should represent $1x^4 + 2x^3 + 3x^2 + 4x + 5$.

If I want to evaluate this polynomial, given `x = 2`, I simply call `Polynomial.evaluate_at(2)`.

If I want to display my polynomial, I simply `print` it.

I should also be able to add one polynomial to another using `+=`.

Topic 5 (using Python):

Problem 1: Reverse Digits

Given a 32-bit signed integer, return the reversed digits of this integer.

Example 1:

```
Input: 123  
Output: 321
```

Example 2:

```
Input: -123  
Output: -321
```

Example 3:

```
Input: 120  
Output: 21
```

Note:

Try to solve this problem using math equations. Eg: don't cast this number to str/list/etc.

Problem 2:

Write a program to check whether a given number is a **Funny** number.

Funny numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 6, 8 are **Funny** while 14 is not **Funny** since it includes another prime factor 7.

Example

Given num = 8 return **true**

Given num = 14 return **false**