```
# Tutorial OOP
# Week 9
# Scope of Variables, Multiple Inheritance, Composition
and Aggregations
# author: B. Schoen-Phelan
# date: Nov 2020

# scope of a variable, first without classes

greeting = "Hello World"

def change_greeting(new_greeting):
    greeting = new_greeting

def greeting_world():
    world = "World"
    print(greeting, world)

change_greeting("Hi")
greeting_world()

# with global scope
greeting = "Hello World"

def change_greeting(new_greeting):
    global greeting
    greeting = new_greeting

def greeting_world():
    world = "World"
    print(greeting, world)

change_greeting("Hi")
greeting_world()

# enclosing scope
# try to change first_num from within inner()
# not working
```

```
def outer():
 first_num = 1

 def inner():
   first_num = 0
   second_num = 1
   print("inner - second_num is: ", second_num)

 inner()
 print("outer - first_num is: ", first_num)
outer()

def outer():
 first_num = 1

 def inner():
   nonlocal first_num
   first_num = 0
   second_num = 1
   print("inner - second_num is: ", second_num)

 inner()
 print("outer - first_num is: ", first_num)


outer()



# instance vs class variables
class Person:
   TITLES = ('Dr', 'Mr', 'Mrs', 'Ms')

   def __init__(self, title, f_name, l_name):
       if title not in self.TITLES:
           raise ValueError("Not a valid title: ",
title)

       self.title = title
```

```
        self.first_name = f_name
        self.last_name = l_name


p = Person("Ms","Bianca", "Phelan")
print(p.TITLES)
print(Person.TITLES)
Person.first_name



#
https://www.geeksforgeeks.org/multiple-inheritance-in-p
ython/
# Python Program to depict multiple inheritance
# when method is overridden in both classes

class ClassA:
    def play_game(self):
        print("Playing in ClassA")

class ClassB(ClassA):
    def play_game(self):
        print("Playing in ClassB")

class ClassC(ClassA):
    def play_game(self):
        print("Playing in ClassC")

class ClassD(ClassB, ClassC):
    pass


d = ClassD()
d.play_game()



# Python Program to depict multiple inheritance
```

```
# when method is overridden in one of the classes

class ClassA:
 def play_game(self):
       print("Playing in ClassA")

class ClassB(ClassA):
 pass

class ClassC(ClassA):
 def play_game(self):
       print("Playing in ClassC")

class ClassD(ClassB, ClassC):
 pass

d = ClassD()
d.play_game()

# Python Program to depict multiple inheritance
# when every class defines the same method

class ClassA:
 def play_game(self):
       print("Playing in ClassA")

class ClassB(ClassA):
 def play_game(self):
       print("Playing in ClassB")

class ClassC(ClassA):
 def play_game(self):
       print("Playing in ClassC")

class ClassD(ClassB, ClassC):
 def play_game(self):
       print("Playing in ClassD")

d = ClassD()
```

```
d.play_game()


#
# # Want to explicitly call others?
#
b = ClassB()
b.play_game()

c = ClassC()
c.play_game()

a = ClassA()
a.play_game()
#

# calls with super
# Python program to demonstrate
# super()

class ClassA:
    def play_game(self):
        print("Playing in ClassA")


class ClassB(ClassA):
    def play_game(self):
        print("In ClassB")
        super().play_game()


class ClassC(ClassA):
    def play_game(self):
        print("In ClassC")
        super().play_game()


class ClassD(ClassB, ClassC):
    def play_game(self):
```

```
        print("Playing in ClassD")
        super().play_game()

d = ClassD()
d.play_game()

# with calling explicit Classes
class ClassA:
    def play_game(self):
        print("Playing in ClassA")


class ClassB(ClassA):
    def play_game(self):
        print("In ClassB")
        super().play_game()


class ClassC(ClassA):
    def play_game(self):
        print("In ClassC")
        super().play_game()


class ClassD(ClassB, ClassC):
    def play_game(self):
        print("Playing in ClassD")
        ClassA.play_game(self)

d = ClassD()
d.play_game()




print(ClassD.mro())
```

```
#  example from
https://www.programiz.com/python-programming/multiple-i
nheritance

class X:
    pass

class Y:
    pass

class Z:
    pass

class A(X, Y):
    pass

class B(Y, Z):
    pass

class M(B, A, Z):
    pass

print(M.mro())
#

# Composition and Aggregation with
# Employee and Salary example
class Salary:
    def __init__(self, pay, bonus):
        self.pay = pay
        self.bonus = bonus

    def annual_salary(self):
        return (self.pay*12) + self.bonus
# composition:

class Employee:
    def __init__(self, name, age, pay, bonus):
        self.name = name
```

```
        self.age = age
        self.salary_object = Salary(pay, bonus)

    def total_salary(self):
        return self.salary_object.annual_salary()

anna = Employee("Anna", 25, 2500, 10000)
print(anna.total_salary())

# aggregation:

class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age= age
        self.salary_object = salary

    def total_salary(self):
        return self.salary_object.annual_salary()

sal = Salary(2500, 10000)
anna = Employee("Anna", 25, sal)
print(anna.total_salary())
```