# Object-Oriented Programming

UBCO Master of Data Science – DATA 533

# Introductions

Instructor: Dr. Mohammad Khalad Hasan

- Assistant Professor, Computer Science
- Research area: Human-Computer Interaction, Input and Interaction Techniques, Information Visualization
- Website: https://cmps-people.ok.ubc.ca/mkhasan/

- Office hours: Tuesday: 12:30 pm – 1:30 pm or by appointment
- Email: please use Canvas Inbox

# Teaching Assistant

TA: A.K.M. Amanat Ullah

- PhD Student in Computer Science

# Course Objectives

The overall goal of this course is for you to:

Understand and apply fundamental concepts of collaborative software development techniques (e.g., software lifecycle, testing, version control, quality control).

# Grading

iClicker: **0%**

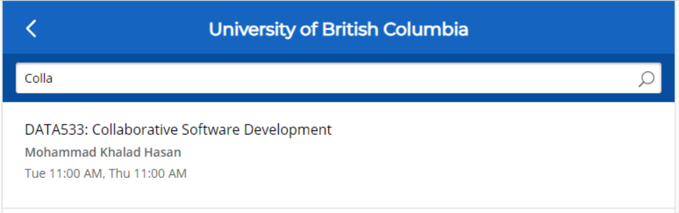
Lab (1): **10%**

Project steps (3): **60%**

Quiz (1): **30%**

# The In-Class Clicker Quizzes

There will be ~40 in-class multiple-choice questions in all lectures. Each question is worth 0 mark.



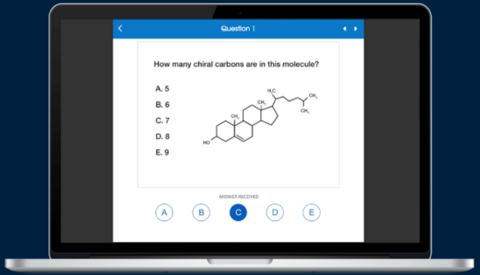
University of British Columbia

Colla

DATA533: Collaborative Software Development
Mohammad Khalad Hasan
Tue 11:00 AM, Thu 11:00 AM

You need:

- iClicker Student Account
  - https://www.iclicker.com/students/apps-and-remotes/web
- Click on + to add a course, type institution name and  type “Collaborative Software Development”
- At different times during all the lectures, questions reviewing material will be asked.



Question 1

How many chiral carbons are in this molecule?

A. 5
B. 6
C. 7
D. 8
E. 9

# The Lab Assignment

One **lab** assignment is worth **10%** of your overall grade.

Lab assignments steps may take more than the two hours lab time.

- No late submissions will be accepted.
- A lab/project step may be handed in any time before the due date.

**Lab** assignments are done **individually**

They are critical to learning the material and are designed both to prepare you for the exams and build up your skills!

# Project Steps

Three **project steps** are worth **60%** of your overall grade.

Project steps may take more than the two hours lab time.

- No late submissions will be accepted.
- A lab/project step may be handed in any time before the due date.

**Project steps** are done in a **group of 2**

# The Quizzes

One quiz: 30% of total marks

Quiz Date: https://github.com/ubco-mds-2023/Data-533

Exam format:  In-class exam

Allowed materials:

- Recorded class lectures
- Slides/pdf files that uploaded as lecture materials
- Reading materials from GitHub
- Code that you wrote as a part of lab / in-class activities
- Your written notes (e.g., pdf files or in a paper).

Compilers/Editors (e.g., Python, Jupyter Notebook) are not allowed

# Academic Dishonesty

Cheating is strictly prohibited and is taken very seriously by UBC.

A guideline to what constitutes cheating:

- Labs
  - Submitting code produced by others.
  - Working in groups to solve questions and/or comparing answers to questions once they have been solved (except for group assignments).
  - Discussing HOW to solve a particular question instead of WHAT the question involves.
- Exams
  - Only materials permitted by instructor should be in the exam.

Academic dishonesty may result in a "F" for the course and removal from the MDS program.

# How to Excel in This Course

Be here!! Pay attention!!

This course is more about **skills** than knowledge

Memorizing a bunch of facts, or reading course materials before the quizzes, is not good enough.

**Practice, practice, practice!**

“What I hear, I forget. What I see, I remember. What I do, I understand.”

# Systems and Tools

Course material is on GitHub.

- https://github.com/ubco-mds-2023/Data-533

Marks are distributed on Canvas.

- https://canvas.ubc.ca/

Your laptop will be used to install all software and run programs.

# To-Do

iClicker Student Account

- https://www.iclicker.com/students/apps-and-remotes/web

Install Jupyter Notebook

- http://jupyter.org/install

# Programming Background Question

***Question:*** How many of the following courses do you know?

1) Object Oriented Programming

2) Object Oriented Design

3) Object Oriented Software Design

**A)** 0 **B)** 1 **C)** 2 **D)** 3

# Concepts of Object-Oriented Programming

Objects

Class

Encapsulation

Polymorphism

Inheritance

Data Abstraction

# Object

An ***object*** contains ***attributes/variables*** and ***behavior/methods***

A Car:

- Has attributes (knows stuff):
  - State of an object
  - year, model, make
- Has methods (behaviors):
  - Accelerate, Brake

# Object

Other Examples:

A Circle (on the screen):

- Has attributes (knows stuff):
  - radius, center, color
- Has methods (behaviors  or can do stuff):
  - move
  - change color

# Class

A ***class*** is a blueprint for the object.

A class is a special data type which defines how to build a certain object.

All values of this type are called ***objects***

A class has:

- A name (use CamelCase notation)
- Some kind of data that it stores in each object
  - A collection of properties called attributes
- Some actions that it can perform on such objects
  - A collection of functions/methods

# Class

A class is like a form or questionnaire. It defines the needed information. After you fill out the form, your specific copy is an instance of the class.

To define a class:

```
class Person:
    pass    # An empty block
```

While the class is the blueprint, an ***instance*** is a copy of the class with *actual* values, literally an object belonging to a specific class.

To instantiate an object:

```
p1 = Person()
```

# Syntax

Class:

```
class className[(superclass)]:
        [attributes and methods]
```

Object :

```
object = className()
```

Attributes and methods:

```
object.attribute
object.method()
```

This method must have at least one argument as well as the self variable, which refers to the object itself

# Initializing an Object

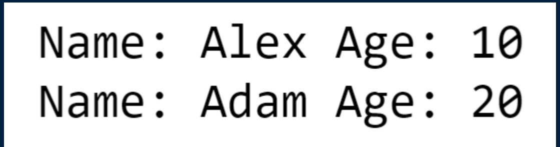
Recall: All objects contain characteristics called attributes

We use `__init__()` method to ***initialize*** an object's initial attributes

The `__init__()` method is run as soon as an object of a class is instantiated.

# Initializing an Object

```
class Person:
    def __init__(self, name, age):
        self.name = name  # instance attributes
        self.age = age    # instance attributes


p1 = Person('Alex', 10)
p2 = Person('Adam', 20)


print('Name:', p1.name, 'Age:',p1.age);
print('Name:', p2.name, 'Age:',p2.age);
```



```
Name: Alex Age: 10
Name: Adam Age: 20
```

# Init and Self

You will never have to call the `__init__()` method

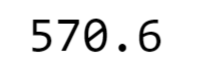
It gets called automatically when you create a new object.

The `self` parameter refers to the object (instance) itself.

Here `self.name = name` sets the name of the object `self.name` equal to the variable name.

# Classes Have Methods

```
class Stock():
    def __init__(self, name, symbol, prices=[]):
        self.name = name
        self.symbol = symbol
        self.prices = prices
    def high_price(self):
        if len(self.prices) == 0:
            return 'MISSING PRICES'
        return max(self.prices)
apple = Stock('Apple', 'APPL', [500.43, 570.6])
print(apple.high_price())
```



```
570.6
```

# Deleting Instances

In Python, we don’t have to delete or free an object explicitly.

Python supports ***automatic garbage collection***.

Python will automatically detect when all of the references to a piece of memory have gone out of scope.

There’s also no “destructor” method for classes (e.g., C++)

# Class Question

***Question:*** Which of the following represents a template or blueprint that defines objects of the same type?

**A)** A class

**B)** An object

**C)** A method

**D)** An attribute

**E)** None of the above

# Class Question

***Question:*** The program would show “Test Message” as output if we change

```
class Test:
    def __init__(self, var):
        self.var = var

    def output(self):
        print(var)

a = Test('Test Message')
a.output()
```

**A)** `self.var = var` to `self = var`

**B)** `def output(self)` to `def output()`

**C)** `print(var)` to `print(self.var)`

**D)** `a.output()` to `output()`

**E)** None of the above

# Class Question

***Question:*** What is the output of the following program?

```
class Customer():
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance
    def withdraw(self, amount):
        if amount > self.balance:
            return 0
        self.balance -= amount
        return self.balance
jeff = Customer('Jeff Knupp', 1000)
print(jeff.withdraw(100))
```

**A)** 1100

**B)** 1000

**C)** 900

**D)** 0

**E)** None of the above

# Try it: Creating Class

***Question:*** Create a class called `Rectangle`. Write the `__init__` method to take the width and height of a rectangle as arguments. Add a method called `area` to compute and return the area (i.e., width × height) of the rectangle.

```
rect = Rectangle(10,20)
print(rect.area())
```

Output:

```
200
```

# Try it: Creating Class

***Question:*** Create a class called `Line` which takes coordinates (i.e., x and y) as a pair of tuples. Write two methods `length` and `slope` to compute and return the length and slope of the line.

```
Sample test code:
coord1 = (10,10)
coord2 = (20,20)
line = Line(coord1, coord2)
print(line.length())
print(line.slope())
```

Output:

```
200
1.0
```

Hints:

```
x1,y1 = coord1
x2,y2 = coord2
```

Length:

```
((y2-y1)* (y2-y1) + (x2-x1)*(x2-x1))
```

Slope:

```
(y2-y1)/(x2-x1)
```

# Class Attributes

Class attributes are shared among all objects of that class.

```
class Car(object):
    wheels = 4                          # class attribute
    def __init__(self, make, model):
        self.make = make                    # instance attribute
        self.model = model                  # instance attribute

mustang = Car('Ford', 'Mustang')
print(mustang.wheels)     # 4
print(Car.wheels)         # 4
```

# Another Example

```
class Employee:
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
   def displayCount(self):
     print("Total Employee %d" % Employee.empCount)
   def displayEmployee(self):
      print("Name : ", self.name,  ", Salary: ", self.salary)
```

# Another Example

```
emp1 = Employee("Zara", 2000)
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print("Total Employee %d" % Employee.empCount)
```

# Accessibility

We can restrict access to methods and variables. This prevents data from direct modification which is called ***encapsulation***.

In other languages (e.g., Java), there are keywords like public, protected, and private to define accessibility.

In Python, ***all attributes are public***.

In Python, we can add "__" (two leading underscores) in front of the variable, and the function name can hide them when accessing them from out of class.

# Private Attributes

```
class Person:
    def __init__(self, name, age):
        self.name = name  # public attribute
        self.__age = age    # private attribute

p1 = Person('Alex', 10)
p2 = Person('Adam', 20)

print("Name:", p1.name, "Age:",p1.age);
print("Name:", p2.name, "Age:",p2.age);
AttributeError: 'Person' object has no attribute 'age'
```

# Get Methods

A method that gets the value of an attribute, which is often private

By convention, a `get` method name starts with `get`

`getAge()` or `get_age()` provides indirect access to `__age`

# Private Attributes

```
class Person:
    def __init__(self, name, age):
        self.name = name  # public attribute
        self.__age = age  # private attribute

    def getAge(self):
        return self.__age
p1 = Person('Alex', 10)
p2 = Person('Adam', 20)

print("Name:", p1.name, "Age:",p1.getAge());
print("Name:", p2.name, "Age:",p2.getAge());
```

# Set Methods

Sets an attribute, often private, to a value

By convention, name starts with `set`, e.g., `setAge()` or `set_age()`

# Set Methods

```
class Person:
    def __init__(self, name, age):
        self.name = name  # public attribute
        self.__age = age  # private attribute
    def getAge(self):
        return self.__age
    def setAge(self, age):
        self.__age = age
p1 = Person('Alex', 10)
p1.setAge(20);
print("Name:", p1.name, "Age:",p1.getAge());
```

# Python Property

```
class person:
    def __init__(self):
        self.__name=''
def setname(self, name):
        print('Setname() is called')
        self.__name=name
def getname(self):
        print('Getname() is called')
        return self.__name
name=property(getname, setname)
```

```
p1=person()
p1.name="John"
p1.name
```

# Python Property Decorator - @property

The property() function is used to define properties in a Python class

***@property***: Declares the method as a property.

***@<property-name>.setter***: Specifies the setter method for a property that sets the value to a property.

***@<property-name>.deleter***: Specifies the delete method as a property that deletes a property.

# Declare a Property

This method must return the value of the property.

```
class Person:
    def __init__(self, name):
        self.__name = name

    @property
    def name(self):
        return self.__name

p1 = Person('Alex')
print("Name:", p1.name);
```

We can now use the `name()` method as a property to get the value of the `__name` attribute

# Declare a Property

To modify the property value, we must define the setter method for the name property using `@property-name.setter` decorator

```
class Person:
    [Code from previous slide]
    @name.setter
    def name(self, value):
        self._name = value

p1 = Person('Alex')
print("Name:", p1.name);
p1.name = "William"
print("Name:", p1.name);
```

# Property Deleter

Use the @property-name.deleter decorator to define the method that deletes a property

```
 @name.deleter   #property-name.deleter decorator
    def name(self, value):
        print('Deleting..')
        del self.__name

p1 = Person('Alex')
print("Name:", p1.name);
del p1.name
print("Name:", p1.name);
```

# Class Attributes Question

***Question:*** How many of the following statements are TRUE?

1) Class attributes owned by the class as a whole

2) Class attributes are shared among all objects of that class

3) Class attributes are good for building a counter of how many instances of the class have been made

4) Each instance has its own value for the class attribute

**A)** 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

# Public vs Private Attributes Question

***Question:*** What is the output of the following program?

```
class Test:
    def __init__(self):
        self.a = 10
        self.__b = 10

    def getA(self):
        return self.a

test = Test()
test.a = 5
print(test.a)
```

**A)** 10

**B)** 5

**C)** 0

**D)** The program has an error because b is private

**E)** The program has an error because a is private

# Override

There are several special methods that are essential to the implementation of a class.

We can *override* built-in methods to define how our objects behave with Python operators/functions.

Some methods to override

`__init__(self,...)`: initialize a newly-created object

`__str__(self)`: String representation of the object

`__repr__(self)`: Object representation

`__cmp__(self, other)`: Compare self and other

# Override

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person('Alex', 10)
p2 = Person('Adam', 20)

print(p1)    <__main__.Person object at 0x000001C76B13C048>
print(p2)    <__main__.Person object at 0x000001C76B13CFD0>
```

# Override

```
class Person:
    ……
    def __repr__(self):
        output = '%s:' % self.name
        output += '%s' % self.age
        return output
    …
p1 = Person('Alex', 10)
p2 = Person('Adam', 20)

print(p1)
print(p2)
```

# Operators

We can overload operators as well.

| Operator | Expression | Internally |
|---|---|---|
| Addition | p1 + p2 | __add__ |
| Subtraction | p1 - p2 | __sub__ |
| Multiplication | p1 * p2 | __mul__ |
| Power | p1 ** p2 | __pow__ |
| Division | p1 / p2 | __truediv__ |
| Remainder | p1 % p2 | __mod__ |

| Operator | Expression | Internally |
|---|---|---|
| Less than | p1 < p2 | __lt__ |
| Less than or equal to | p1 <= p2 | __le__ |
| Equal to | p1 == p2 | __eq__ |
| Not equal to | p1 != p2 | __ne__ |
| Greater than | p1 > p2 | __gt__ |
| Greater than or equal to | p1 >= p2 | __ge__ |

# Overloading Operators

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    def __str__(self):
        return "({0},{1})".format(self.x, self.y)
    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x, y)
p1 = Point(1, 2)
p2 = Point(2, 3)
print(p1+p2)
```

# Inheritance

It can be useful (especially in larger projects) to have a hierarchy of classes.

Example

Animal

- Bird
  - Hawk
  - Seagull
- …

Pet

- Dog
  - …
  - …
- Cat
  - …
  - …

…

Member

- Teacher
  - Khalad
  - Apurva
  - ….
- Student
  - David
  - Alex

# Inheritance

Can have one class ***inherit*** attributes from another class.

Original class is called ***base class or parent class or super class***.

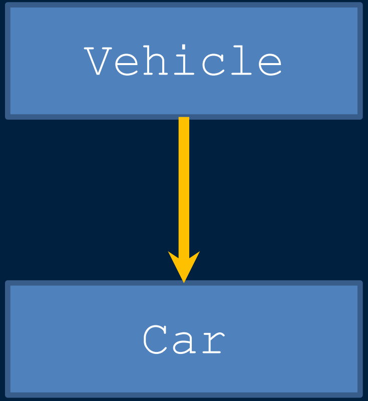
New class is called ***derived class or child class or sub class***.

Derived class inherits features from the base class where new features can be added to it (results in re-usability of code).

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

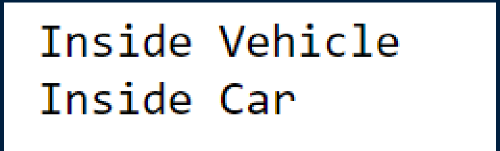
# Inheritance Example (Single Inheritance)

```
# Base class
class Vehicle:
    def Vehicle_info(self):
        print('Inside Vehicle')

# Child class
class Car(Vehicle):
    def car_info(self):
        print('Inside Car')
```



# Inheritance Example

```
# Create object of Car
car = Car()

# access Vehicle's info using car object
car.Vehicle_info()
car.car_info()
```



```
Inside Vehicle
Inside Car
```

# Inheritance Example

```
class Member:      # Super class, any university member
    def __init__(self, name, age):  # initialize name and age
        self.name = name
        self.age = age
        print('(Initialized Member: {})'.format(self.name))

    def display(self):    # display name and age
        print('Name:"{}" Age:"{}"'.format(self.name, self.age))
```

# Inheritance Example

```python
class Teacher(Member): # Teacher subclass, represents a teacher
    def __init__(self, name, age, salary):
        Member.__init__(self, name, age)
        self.salary = salary
        print('(Initialized Teacher: {})'.format(self.name))

    def display(self):
        Member.display(self)
        print('Salary: "{:d}"'.format(self.salary))
```

# Inheritance Example

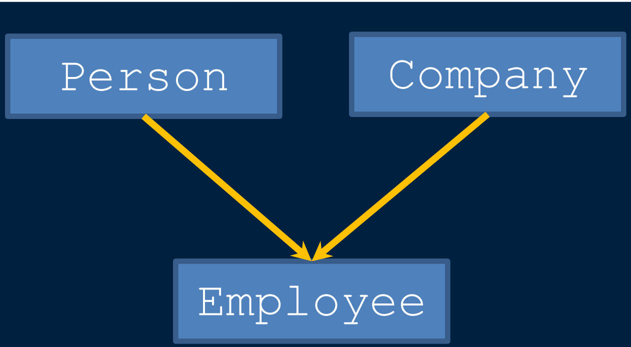
```
class Student(Member): # Student subclass, represents a student
    def __init__(self, name, age, marks):
        Member.__init__(self, name, age)
        self.marks = marks
        print('(Initialized Student: {})'.format(self.name))

    def display(self):
        Member.display(self)
        print('Marks: "{:d}"'.format(self.marks))
```

# Inheritance Example

```
teacher = Teacher('Alex', 60, 80000)
student = Student('David', 25, 75)


members = [teacher, student]
for member in members:
    member.display()
```

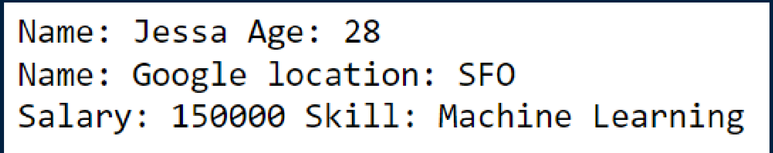
# Inheritance Example (Multiple Inheritance)

```
# Parent class 1
class Person:
    def person_info(self, name, age):
        print('Name:', name, 'Age:', age)

# Parent class 2
class Company:
    def company_info(self, company_name, location):
        print('Name:', company_name, 'location:', location)

# Child class
class Employee(Person, Company):
    def Employee_info(self, salary, skill):
        print('Salary:', salary, 'Skill:', skill)
```



# Inheritance Example (Multiple Inheritance)

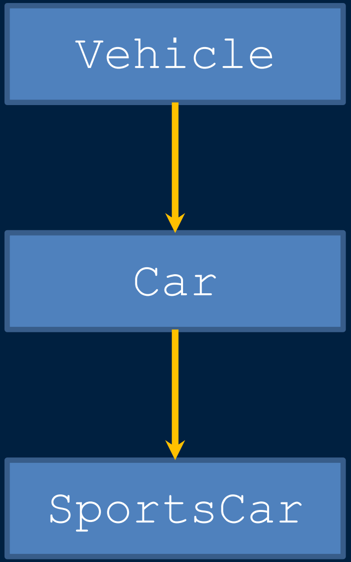
```
# Create object of Employee
emp = Employee()


# access data
emp.person_info('Jessa', 28)
emp.company_info('Google', 'SFO')
emp.Employee_info(150000, 'Machine Learning')
```



```
Name: Jessa Age: 28
Name: Google location: SFO
Salary: 150000 Skill: Machine Learning
```

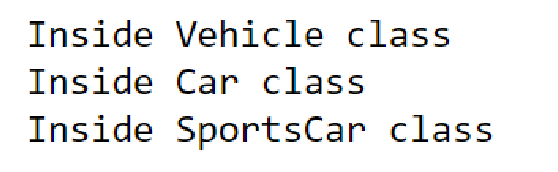
# Inheritance Example (Multilevel inheritance)

```
# Base class
class Vehicle:
    def Vehicle_info(self):
        print('Inside Vehicle class')

# Child class
class Car(Vehicle):
    def car_info(self):
        print('Inside Car class')

# Child class
class SportsCar(Car):
    def sports_car_info(self):
        print('Inside SportsCar class')
```



# Inheritance Example (Multilevel Inheritance)

```
# Create object of SportsCar
s_car = SportsCar()

# access Vehicle's and Car info using SportsCar object
s_car.Vehicle_info()
s_car.car_info()
s_car.sports_car_info()
```



```
Inside Vehicle class
Inside Car class
Inside SportsCar class
```

# Method overriding

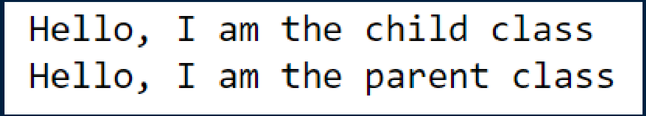
Two methods with the same name that each perform different tasks

Two prerequisite conditions for Method overriding:

- Inheritance should be present in the code, method overriding cannot be performed in the same class.
- The child class should have the same name and the same number of parameters as the parent class.

# Code

```
class Animal:
    def Walk(self):
        print('Hello, I am the parent class')

class Dog(Animal):
    def Walk(self):
        print('Hello, I am the child class')

r = Dog()  #Invoking Child class
r.Walk()
r = Animal() #Invoking Parent class
r.Walk()
```



```
Hello, I am the child class
Hello, I am the parent class
```

# Notes

New-Style and Old-Style Classes

```
class Person(object):   # new-style class
class Person:           # old-style class
```

# Inheritance Question

***Question:*** What is the output of the following piece of code?

```
class A():
    def display(self):
        print("DATA533")
class B(A):
    pass
obj = B()
obj.display()
```

**A)** Nothing will be printed

**B)** DATA533

**C)** Invalid syntax for inheritance

**D)** Error due to incomplete class B

**E)** Error due no argument in B()

# Inheritance Question

***Question:*** What is the output of the following piece of code?

```
class First:
    def one(self):
        return self.two()
    def two(self):
        return 'Welcome'

class Second(First):
    def two(self):
        return 'Hello'

object1=First()
object2=Second()
print(object1.two(),object2.two())
```

A) Hello Welcome

B) Welcome Hello

C) Hello Hello

D) Welcome Welcome

E) None of the above

# Try it: Inheritance

***Question:*** Follow the instructions below:

- Write a Rectangle class in Python language, allowing you to build a rectangle with length and width attributes.
- Create a Perimeter() method to calculate the perimeter of the rectangle and a Area() method to calculate the area of the rectangle.
- Create a method display() that display the length, width, perimeter and area of an object created using an instantiation on rectangle class.
- Create a Parallelepipede child class inheriting from the Rectangle class and with a height attribute and another Volume() method to calculate the volume of the Parallelepiped.

# Objectives

- Define classes, instantiate objects, writing methods in classes
- Know how to create and delete instances
- Access public and private members in classes
- Know how to use two kinds of attributes: data and class attributes
- Know how to use Inheritance in Python
- Be able to write a child class extending a parent class

THE UNIVERSITY OF BRITISH COLUMBIA