

# DATA 200

Paramdeep Saini, SJSU

# Week 2 - DATA 200

- Questions from Week 1 & 2?
  - HW?
  - If/elif, while/loop, Conditional Expression?
  - Negative Subscript?
  - List Operations: insert,sort,pop,reverse?
  - Slices?
  - Tuple/Dictionary/Set and their operations?
  - Zip/Enumerator/Iterator

# Structured Programming

Technique that is considered as precursor to OOP

- Consists of well-structured and separated modules

- It is a subset of procedural programming

- Programs are divided into small programs or functions

- It generally follows “Top-Down Approach”

- It gives more importance of code

- Less flexible and abstraction as compared to object-oriented programming

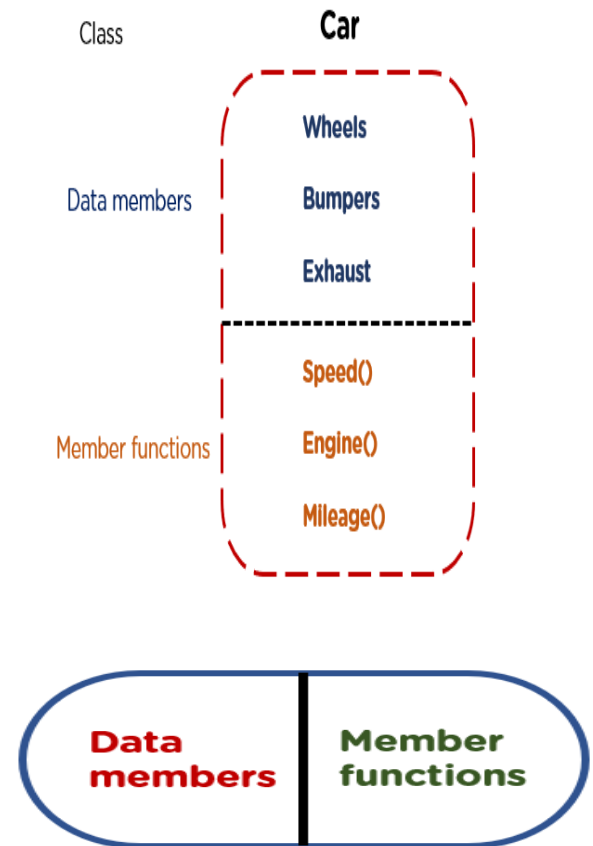
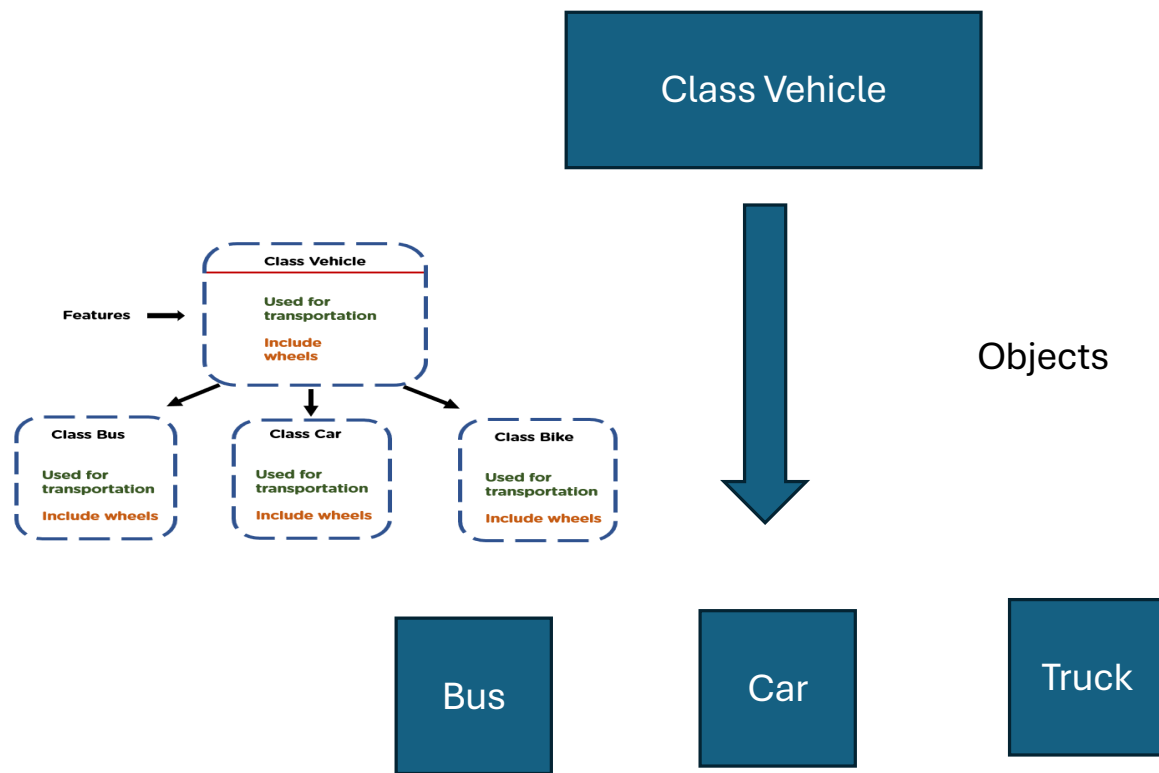
# Structured Programming

```
def get_soccer_playerinfo()  
def set_wrestler_player_info()  
def get_soccer_ticket_info()  
def get_wrestling_ticket_info()  
total_wrestler=30  
soccer_teams=30
```

```
def get_soccer_playerinfo()  
def get_soccer_ticket_info()  
soccer_teams=30
```

```
def set_wrestler_player_info()  
def get_wrestling_ticket_info()  
total_wrestler=30
```

# Object Oriented Programming



# Object Oriented Programming

- Programs are structured around objects rather than functions and logic
- Separation of data and functions, and helps make the code flexible and modular
- It represent data as objects that have attributes and functions
- It generally follows “Bottom-Up Approach”
- It gives more importance to data
- Code Re-use

# OOP Concepts

- Object-oriented programming is founded on these ideas:
  - **object/class**: An object is a collection of data and behaviors that operate on that data. A class refers to a type or category of objects.
  - **information hiding (encapsulation)**: The ability to isolate some of the object's components from external entities ("private").
  - **inheritance**: A class ("subclass") can extend or override the functionality of another class ("superclass").
  - **polymorphism**: The ability to replace an object with its sub-objects to achieve different behavior from the same piece of code.
  - **interface**: A specification of method signatures without supplying implementations, as a mechanism for enabling polymorphism.

# Object Oriented Design

Before you start writing Object Oriented Program, ask following question for Object Oriented Design:

- What is the description of a particular problem domain or software system

- What are its necessary features in high-level general terms

- Identify items that might be good to represent as classes if the system were to be implemented

Hints:

- Classes and objects often are generally referred as nouns in the problem description.

- Some nouns are too trivial to represent as entire classes; maybe they are simply data (fields) within other classes or objects.

- Behaviors of objects are generally referred as verbs

- Classes that can be part of inheritance



# OOD Exercise -1

- Develop a tool for MealPlan. The main idea is that Health app should create a meal plan for the customers based on their health condition. Customers have targets such as reducing weight and customer must be able to track the progress.
  - Basic Requirements
    - The Health app must support add, edit, delete and list customers.
    - The Health app should be able to check detailed information for a selected client, this includes the meals plan, body type and weight etc.
    - The Health app must be able to design meal plan based on calories and nutrients.
    - The health app can provide weekly report and should be able to different meals such as Dinner/Breakfast/Lunch

## OOD Exercise - 2

What classes are in this Audio store kiosk system?

- The software is for an audio kiosk that replaces human clerks
- A customer with an account can use their membership and credit card at the kiosk to check out an audio
- The software can look up audio and actors by keywords
- A customer can check out up to 4 audio, for 7 days each
- Late fees can be paid at the time of return or at next checkout

# Quiz

We need to use members (data/functions) of used-vehicle inventory for the object type named AvailableVehicleLots. Select True if the item should become part of the AvailableCarLots object type, and False otherwise.

- 1) vehicle\_sticker\_price
- 2) today\_temperature
- 3) available\_days
- 4) original\_purchase\_price
- 5) sales\_people
- 6) count\_available\_vehicle\_days()
- 7) increase\_vehicle\_sticker\_price()
- 8) identify\_best\_salesman()

# Classes

- A class defines a new type that can group data and functions to form an object.
- The object maintains a set of **attributes** that determines the data and behavior of the class.

```
class Date:
    """ A class that represents a date """
    def __init__(self):
        self.day = 0
        self.month = 0
        self.year = 0
```

Init method is a Constructor, it is responsible for setting initial state. Special method starts with \_\_ underscore. You should avoid using method start with \_\_ underscore.

- An **instantiation** operation is performed by "calling" the class, using parentheses like a function call as in `today_date = date()`
- An instantiation operation creates an **instance**, which is an individual object of the given class
- An instantiation operation automatically calls the `__init__` method defined in the class definition

# Classes

```
class Date:
```

```
    """ A class that represents a date """
```

```
    def __init__(self):
```

```
        self.day = 0
```

```
        self.month = 0
```

```
        self.year=0
```

```
today_date= Date()
```

```
today.day=10
```

```
today.month=2
```

```
today.year=2025
```

```
print('{} / {} / {} date'.format(today.day,today.month,today.year))
```

# Classes

```
class Date:
    """ A class that represents a date """
    def __init__(self):
        self.day = 0
        self.month = 0
        self.year = 0

today_date = Date()
today.day = 10
today.month = 2
today.year = 2025

yesterday_date = Date()
yesterday.day = 09
yesterday.month = 2
yesterday.year = 2025

print('{} / {} / {} date'.format(today.day, today.month, today.year))
print('{} / {} / {} date'.format(yesterday.day, yesterday.month, yesterday.year))
```

# Quiz

A class group related variables?

True/False

The `__init__` method calls automatically

True/False

`d=Date()`, `d` refers the instance of date class?

True/False

given the definition of the Date class, what is the value of `date2.day` after the following code executes?

```
date1 = Date()  
date1.day = 7  
date2 = date1
```

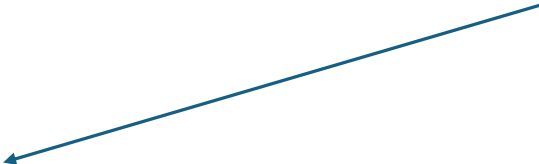
# Instance Method

- A function defined within a class is known as an ***instance method***
- The Python object that encapsulates the method's code is thus called a ***method object***
- A method object is an attribute of a class, and can be referenced using dot notation



# Classes

```
class Date:
    """ A class that represents a date """
    def __init__(self):
        self.day = 0
        self.month = 0
        self.year = 0
    def print_date(self):
        print('{} / {} / {}'.format(self.day, self.month, self.year))
```



```
today_date = Date()
today.day = 10
today.month = 2
today.year = 2025
date1.print_date()
```

print\_date is a Date class method.  
Parameter self which provides  
reference to class instance. A  
Programmer does not specify self  
while calling

# Quiz

Consider the following:

Class Vehicle:

```
def __init__(self):  
    #....  
def getColor(self,color):  
    #.....
```

- 1) Write a statement that creates an instance of vehicle class called car? -----
- 2) Write a statement that calls the getColor method of the car instance with the argument "red".
- 3) What should the first item in the parameter list of every class method while defining?

# Quiz

Is this program correct?

```
class Date:
    """ A class that represents a date """
    def __init__(self):
        self.day = 0
        self.month = 0
        self.year = 0
    def print_date():
        print('{} / {} / {} date'.format(self.day, self.month, self.year))

today_date = Date()
today_date.day = 10
today_date.month = 2
today_date.year = 2025
today_date.print_date()
```

# Frames and Objects

```

1.  class Date:
    """ A class that represents a date """
2.  def __init__(self):
3.      self.day = 0
4.      self.month = 0
5.      self.year = 0
5.  def print_date(self):
7.      print('{} / {} / {}'.format(self.day, self.month, self.year))

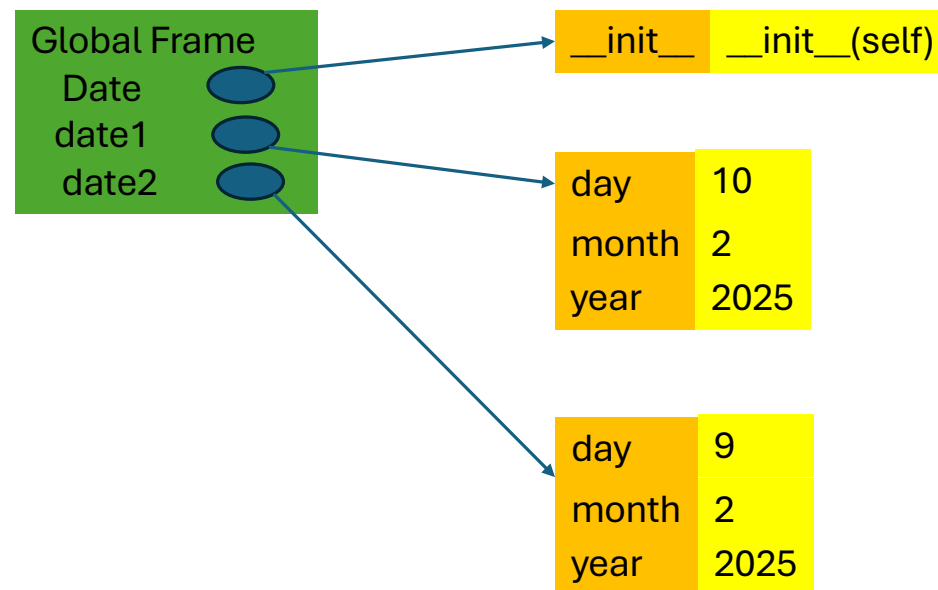
8.  date1 = Date()
9.  date2.day = 10
10. date3.month = 2
11. date4.year = 2025

12. date2 = Date()
13. date2.day = 8
14. date2.month = 2
15. date2.year = 2025
16. date1.print_date()

```

Line	Frame	Object
1	Global Frame Date	Date Class  __init__ - >__init__(self)
2	__init__ self	Date instance
3/4/5	Self.day/month and year	Set to 0
8	Date date1	Date Class date1=Date Instance Day=0,month=0,year =0
9/10/11	date1	date1.day=10,date1. month=2,date.year= 2025
12	date2	Date Instance

# Frames and Objects



# Class Attributes

```
class Date:
    """ A class that represents a date """
    timezone="PST"
    def __init__(self):
        self.day = 0
        self.month = 0
        self.year=0
    def print_date():
        #---print date
```

```
today_date= Date()
today.day=10
today.month=2
today.year=2025
```

```
yesterday_date= Date()
yesterday.day=09
yesterday.month=2
yesterday.year=2025
```

# Quiz

**What is a class attribute?**

**What is instance attribute?**

# Class Attributes vs Instance Attributes

**class** Date:

""" A class that represents a date """

timezone="PST"

**def** \_\_init\_\_(self):

self.day = 0

self.month = 0

self.year=0

**def** print\_date():

#---print date

today\_date= Date()

today\_date.day=10

today\_date.month=2

today\_date.year=2025

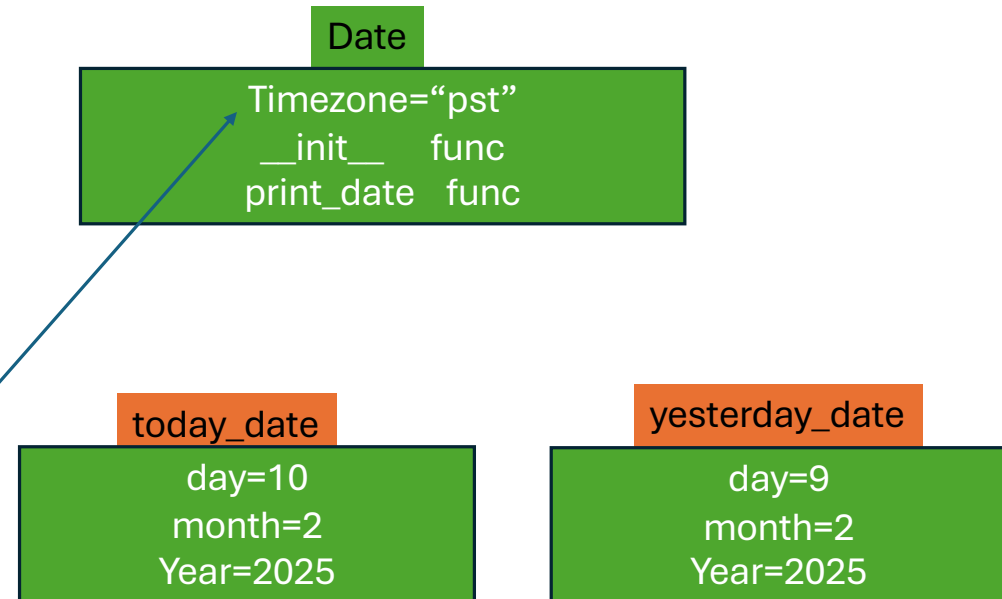
yesterday\_date= Date()

yesterday\_date.day=09

yesterday\_date.month=2

yesterday\_date.year=2025

print(today\_date.timezone,yesterday\_date.timezone)





# Quiz

Instance Attributes, Class Object, Instance Method, Instance Object, Class Attribute

A variable shared with all the instances of a class. -----

Single Instance of a class. -----

Parent which creates new class instances. -----

A variable part of a single instance. -----

Functions which are class attributes. -----

# Quiz

Is it a class attribute or instance attribute?

```
class animal:  
    def __init__(self):  
        self.type= 'mammal'
```

Is it correct code?

```
class animal:  
    def __init__(self):  
        self.sound= None  
cat=Animal()  
cat.sound= 'Meow'
```

# Quiz

Is it a class attribute or instance attribute?

```
class animal:  
    def __init__(self):  
        self.type= 'mammal'
```

Is it correct code?

```
class animal:  
    def __init__(self):  
        self.sound= None  
cat=Animal()  
cat.sound= 'Meow'
```

**type** is an Instance or class attribute?

```
class animal:  
    type=['mammal','reptiles']  
    def __init__(self):  
        self.sound=None
```

# Python Class Constructor

The `__init__` class method can be populated with more than one argument

Constructors can have default variable values like other functions.

```
class Animal:
    """ Class for Animals """
    def __init__(self, name, type, sound):
        self.name = name
        self.type = type
        self.sound = sound

a2 = Animal('cat', 'Mammal', 'meow')
print(a2.name, a2.type, a2.sound)
```

# Quiz

```
class Animal:
    """ Class for Animals """
    def __init__(self):
        self.name = None
        self.type = None
        self.sound = None

    def __init__(self, name, type, sound):
        self.name = name
        self.type = type
        self.sound = sound

a1 = Animal()
print(a1.name, a1.type, a1.sound)
a1.name = 'Monkey'
a1.type = 'Mammal'
a1.sound = 'oookhikhi'
print(a1.name, a1.type, a1.sound)
a2 = Animal('cat', 'Mammal', 'meow')
print(a2.name, a2.type, a2.sound)
```

# Quiz

Is it correct? If yes, what would be the output?

```
class Animal:
    def __init__(self, name, sound, type='mammal'):
        self.name=name
        self.type=type
        self.sound=sound

a1=Animal('cat', 'meow')
print(a1.name, a1.type, a1.sound)
```

# Quiz

Is it correct? If yes, what would be the output?

```
class Animal:
def __init__(self,name,type='mammal',sound):
self.name=name
self.type=type
self.sound=sound
a1=Animal('cat','meow')
print(a1.name,a1.type,a1.sound)
```

# Class Interface

- A class interface in Python defines the methods and attributes that a class should implement, establishing a contract for how objects of that class should behave.
- Python doesn't have an explicit `interface` keyword like some other languages, but it achieves similar functionality through abstract base classes (ABCs) and informal protocols (duck typing).
- A ***class interface*** consists of the methods that a programmer calls to create, modify, or access a class instance.



# Class Interface - Example

```
class Vehicle:
    def __init__(self, name, type, starthrs, startmins, endhrs, endmins, dist):
        self.name = name
        self.type = type
        self.starthrs = starthrs
        self.startmins = startmins
        self.endhrs = endhrs
        self.endmins = endmins
        self.distance = dist

    def get_time(self):
        finaltime = self._adjust_time()
        print('Vehicle time to complete the distance: {}:{}'.format(finaltime[0], finaltime[1]))

    def get_pace(self):
        finaltime = self._adjust_time()
        finalminutes = finaltime[0]*60 + finaltime[1]
        print('Avg pace (mins/mile): {:.2f}'.format(finalminutes / self.distance))

    def _adjust_time(self):
        """Calculate total distance time. Returns a 2-tuple (hours, minutes)"""
        if self.endmins >= self.startmins:
            minutes = self.endmins - self.startmins
            hours = self.endhrs - self.starthrs
        else:
            minutes = 60 - self.startmins + self.endmins
            hours = self.endhrs - self.starthrs - 1

        return (hours, minutes)

distance = 5.0

starthrs = int(input('Enter starting time hours: '))
startmins = int(input('Enter starting time minutes: '))
endhrs = int(input('Enter ending time hours: '))
endmins = int(input('Enter ending time minutes: '))
```

# Quiz

A class interface consists of the methods that a programmer should use to modify or access the class

True/False

Internal methods used by the class should start with an underscore in their name

True/False

Internal methods can not be called; e.g., `Vehicle._AdjustSpeed()` results in an error.

True/False

# Class with Special Methods

```
class Animal:
    def __init__(self, name, sound, type='mammal'):
        self.name=name
        self.type=type
        self.sound=sound

a1=Animal('cat', 'meow')
print(a1)
```

```
class Animal:
    def __init__(self, name, sound, type='mammal'):
        self.name=name
        self.type=type
        self.sound=sound

    def __str__(self):
        return ('It is {} and its type is {} and the sound is {}'.format(self.name, self.type, self.sound))

a1=Animal('cat', 'meow')

print(a1)
```

```
<__main__.Animal object at 0x10bd34fd0>
It is cat and its type is mammal and the sound is meow!
```

# Rich Comparison Method

Rich Comparison Method	Overloaded Operator
<code>__lt__(self, other)</code>	less-than (<)
<code>__le__(self, other)</code>	less-than or equal-to (<=)
<code>__gt__(self, other)</code>	greater-than (>)
<code>__ge__(self, other)</code>	greater-than or equal-to (>=)
<code>__eq__(self, other)</code>	equal to (==)
<code>__ne__(self, other)</code>	not-equal to (!=)

# Quiz

Fill in the missing code as described in each question to complete the rich comparison methods.

```
class Vehicle:
    def __init__(self, name, type, price, condition):
        self.name = name
        self.type = type
        self.price = price
        self.condition = condition
```

1) A vehicle is less than another truck if the price is lower

```
def __lt__(self, truck):
```

```
    if -----
```

```
        return True
```

```
    return False
```

2) A vehicle less than or equal to another truck if the price is at most the same.

```
def __lt__(self, truck):
```

```
    if -----
```

```
        return True
```

```
    return False
```

# Inheritance – Derived Classes

## Vehicle Class

```
+ name  
+ seats  
+ __init__(self)  
+ setName(self)  
+ getName(self)  
+set_quantity(self,seats)  
+get_quantity(self,seats)  
+ display(self)
```

## GasVehicle Class

```
+ fuelcapacity  
+ expiration  
+ __init__(self)  
+ setexpiration(self)  
+ getexpiration(self)  
+setfcapacity(self,capacity)  
+getfcapacity(self)
```

Access to all the attributes/functions of Vehicle class

## ElectricVehicle Class

```
+ chargecapacity  
+ expiration  
+ __init__(self)  
+ setexpiration(self)  
+ getexpiration(self)  
+setecapacity(self,capacity)  
+getecapacityself,)
```

Access to all the attributes/functions of Vehicle class

# Inheritance – Derived Classes ( Better?)

## Vehicle Class

```
+ name  
+ seats  
+ expiration  
+ __init__(self)  
+ setName(self)  
+ getName(self)  
+ set_quantity(self,seats)  
+ get_quantity(self,seats)  
+ display(self)  
+ setexpiration(self)  
+ getexpiration(self)
```

## GasVehicle Class

```
+ fuelcapacity  
+ __init__(self)  
+ setfcapacity(self,capacity)  
+ getfcapacity(self)
```

## ElectricVehicle Class

```
+ chargecapacity  
+ __init__(self)  
+ setecapacity(self,capacity)  
+ getecapacityself,)
```

Access to all the attributes/functions of Vehicle class

Access to all the attributes/functions of Vehicle class

# Quiz

A class that can serve as the basis for another class is called a ----- class?

Class "Vehicle" has the method set\_name(). Class "GasVehicle" is derived from Vehicle and has the methods set\_capacity() and Speec\_limit(). After gv = GasVehicle() executes, how many different methods can gv call, ignoring constructors?

-----



# Overriding

A derived class may define a method having the same name as a method in the base class. Such a member function ***overrides*** the method of the base class.

# Quiz

Which of the following derived class is wrong? And why?

## Base Class

```
Class A:  
    def __init__(self):  
        self.a=""  
    def display():  
        print(a)
```

## Derived Class

```
Class B:  
    def __init__(self):  
        A.__init__(self)  
        self.b=""  
    def display():  
        A.display()  
        print(self.b)
```

## Derived Class

```
Class C:  
    def __init__(self):  
        A.__init__(self)  
        self.c=""  
    def display(self):  
        A.display(self)  
        print(self.c)
```

## Derived Class

```
Class B:  
    def __init__(self):  
        A.__init__(self)  
        self.b=""  
    def display(self):  
        self.display(self)  
        print(self.b)
```

# Is-a Vs has-a Relationship

Is Inheritance and Composition same?

- When we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. (Inheritance? Or composition?)
- We only define a type which we want to use and which can hold its different implementation also it can change at runtime. (Inheritance or Composition?)
- Is-A (Inheritance? Or Composition)
- HAS-A (Inheritance? Or Composition)

# Quiz

Indicate the type of relationship

Apple/Fruit

IS-A/HAS-A

House/Window

IS-A/HAS-A

Cow/Owner

IS-A/HAS-A

# Unit Testing Module

The Python standard library ***unittest*** module implements unit testing functionality

<https://docs.python.org/3/library/unittest.html>

Method	Checks that	New in
<a href="#"><code>assertEqual(a, b)</code></a>	<code>a == b</code>	
<a href="#"><code>assertNotEqual(a, b)</code></a>	<code>a != b</code>	
<a href="#"><code>assertTrue(x)</code></a>	<code>bool(x)</code> is True	
<a href="#"><code>assertFalse(x)</code></a>	<code>bool(x)</code> is False	
<a href="#"><code>assertIs(a, b)</code></a>	<code>a is b</code>	3.1
<a href="#"><code>assertIsNot(a, b)</code></a>	<code>a is not b</code>	3.1
<a href="#"><code>assertIsNone(x)</code></a>	<code>x is None</code>	3.1
<a href="#"><code>assertIsNotNone(x)</code></a>	<code>x is not None</code>	3.1
<a href="#"><code>assertIn(a, b)</code></a>	<code>a in b</code>	3.1
<a href="#"><code>assertNotIn(a, b)</code></a>	<code>a not in b</code>	3.1
<a href="#"><code>assertIsInstance(a, b)</code></a>	<code>isinstance(a, b)</code>	3.2
<a href="#"><code>assertNotIsInstance(a, b)</code></a>	<code>not isinstance(a, b)</code>	3.2

# Quiz

What is the Python standard library module that allows the definition of unit tests?

Write an assertion that checks if `c1.valid` is `True`.

```
def test_val(self):  
    c1=Circle()  
    self.-----
```

Write an assertion that checks if `c1.val` is less than 5.

```
def test_val(self):  
    c1=Circle()  
    self.-----
```

# Quiz

LIST/SET/TUPLE/Dictionary

# Quiz

What is the output of the following program?

```
temps = [60, 62, 32, 35]
temps.append(79)
print(temps[-1])
print(temps[:-1])
print(temps[-1:])
```

What is the output of the following program?

```
temps = ['Sam', 'Markus']
temps.insert(1, 'Peter')
print(temps[0], temps[1], temps[2])
```



# Quiz

Write the simplest two statements that first sort list1, then remove the largest value element from the list, using list methods.

Write a statement that counts the number of elements of list1 that have the value 12.

# Quiz

Assume that list1 is [0, 10, 20, 25].

What value is returned by sum(list1)?

What value is returned by max(list1)?

What value is returned by any(list1)?

What value is returned by all(list1)?

Assume that the following code has been evaluated:

```
nums = [1, 1, 2, 3, 5, 8, 13]
```

What is the result of nums[1:5]?

What is the result of nums[5:10]?

What is the result of nums [3:-1]?

The result of evaluating nums[0:5:2]

The result of evaluating nums[0:-1:3]

# Quiz

my\_list = [1, 1, 2, 3, 5, 8, 13, 21, 34]

Sno	Expression	Output
1		[5, 8, 13, 21, 34]
2		[]
3		[1, 1, 2, 3, 5, 8, 13, 21, 34]
4		[5]
5		[2, 3, 5]
6		[3, 5, 8]

# List comprehensions - Quiz

Twice the value of each element in the list variable x.

The absolute value of each element in x. Use the `abs()` function to find the absolute value of a number.

The largest square root of any element in x. Use `math.sqrt()` to calculate the square root.

# Conditional List - Quiz

Write a list comprehension that contains elements with the desired values. Use the name 'i' as the loop variable. Use parentheses around the expression or condition as necessary.

Only negative values from the list x

Only negative odd values from the list x

Hint: `new_list = [expression for name in iterable if condition]`

# List Sorting

Difference between sort and sorted method?

# Dictionary

Dictionary entries can be modified in-place – a new dictionary does not need to be created every time an element is added, changed, or removed.

```
### Quiz 1 – What would be the output?
my_dict = {'Sam': 55, 'Marcus': 42}
my_dict.clear()
print(my_dict)

#### Quiz 2: What would be the output?
my_dict = {'Sam': 55, 'Marcus': 42}
print(my_dict.get('Peter', 'N/A'))
print(my_dict.get('John', 'N/A'))

### Quiz 3: What would be the output?
my_dict = {'Sachin': 20, 'Sam': 42, 'Marcus': 30}
my_dict.update({'John': 50})
print(my_dict)

### Quiz 4: What would be the output?
my_dict = {'Sachin': 20, 'Sam': 42, 'Marcus': 30}
my_dict.update({'Sam': 50})
print(my_dict)

### Quiz 5: What would be the output?
my_dict = {'Sam': 1, 'Marcus': 42, 'Sachin': 32}
val = my_dict.pop('Sam')
print(my_dict)
```

# Dictionary Concepts

Read About:

- Dictionary Update
- Dictionary key removal using pop
- Reading Key
- Removing Items from dictionary



# Set

A **set** is an unordered collection of unique elements.

```
# Q: Will the output be in given order ?
# Create a set using the set() function.
nums1 = set([1, 2, 3,4,5,6,8,9,23,56,7,8,9,1,55])
# Create a set using a set literal.
nums2 = { 5,7, 8, 9 , 11, 12, 13, 15, 17,23,25,67,89,90}
# Print the contents of the sets.
print(nums1)
print(nums2)

# Q: is it correct?
set(10, 20, 25,11,12)
```

# Set

```
# Question: Add the literal 'Sam' to the set names.  
# Question: Add all of the elements of set A into set B.  
# Question: Remove all of the elements from the A set.  
# Question: Get the number of elements in the set A.  
# What would be the output of following?  
num1=(set([1,2,3,4,5,6]))  
print(num1.pop())  
  
num1=(set([1,2,3,4,5,6,7,8,11,12,13,45,44,32,33,67,65]))  
print(num1.pop())
```