Ans of Question_1:

- Objects is an instance of a class. (i.e. for Shape class we create an object called circle, rectangle).
- Data are grouped or wrapped together and operate on these data into a class called data binding which is an encapsulation process, and methods are functions that define inside of a class, which are used for defining behaviors of an object.

```
class Shape:
    def __init__(self, h, w, r):
        #initial different types of data
        self.h = h
        self.w = w
        self.r = r
    def area_circles(self): #method
        PI = 3.142
        return PI * (self.r*self.r)
    def area_rectangle(self):#method
        return self.h * self.w
```

- Circle and Rectangle can be a class or object of Shape class or something like that.

```
class Circle:
    def __init__(self,r):
        self.r = r
    def area_circles(self): #method
        PI = 3.142
        return PI * (self.r*self.r)
class Rectangle:
    def __init__(self, h, w):
        self.h = h
        self.w = w
    def area_rectangle(self):#method
        return self.h * self.w
```

- When an object is bound with the functionality at runtime, then it's called runtime polymorphism, which can be achieved by method overriding.

```
class A:
    def check(self):
        print('Hello A')
class B(A):
    def check(self):
```

```
        print('Hello B')
```

Ans of Question_2:

- Stack: Stack is a temporary storage memory. Variable, function stored in stack during runtime and delete from stack when execution is complete of a task.
- Heap: Heap is dynamic memory allocation, which can be allocated by programmers. Heap memory needs to be managed manually.
- To initialize a large array(100mb) I use heap, because heap doesn't have any limit of memory size, where stack memory is very limited.

Ans of Question_3:

- To improve or speedup element-wise matrix multiplication use multiprocessing or threading. If we make each state of the outer loop as a process then we can complete multiplication in N time.

```
def innerLoop(i,ans,q):
        for j in range(len(B[0])):
            ans.append(A[i][j] * B[i][j])
        q.put(ans)

    q = Queue()
    for i in range(len(A)):
        p = Process(target=innerLoop, args=(i,ans,q,))
        processes.append(p)
        p.start()
    for p in processes: p.join()
```

Ans of Question_4:

- In this approach of counting nodes there needs to be some changes to get the correct answer. Here counting events occur several times during each recursive call, that's why it will return wrong output.

```
def traverse(self,root):
        if root == None: return 0 #base case
        self.count+=1
        self.traverse(root.left) #recursive call
        self.traverse(root.right) #recursive call
        return self.count
```