1. **part (a):**

Task: **Communication protocol will allow the charging of bots efficiently.** If a bot is 100% charged and is currently on charge, it will yield the charging point for other bots

Description of the code:

1. GP is the geometric progression sum of count number of terms with first term as 1 and common differenece as 2
2. Bots will be loaded in batches of size batch.
3. First batch loaded if (it cannot be assigned full power P and hence is assigned power p), then second batch will be assigned power(p/2)
4. This process is repeated recursively
5. We also ensure manually input of which bots be charged and what power supply they should be provided by providing override option at the time of charging. Override however does not allow assignment in which power is exceeding the maximum allowable power. If in override power provided is greater than maximum allowable power, the bots which are assigned power will be loaded in heap and provided power as per priority _of_charge method

```
def GP(count):
    return 2*(pow(0.5,count)-1)
define priority_of_charge():
    // if charging is >50% and <80% then priority will be more
    // Followed by Charging <50%
    // Bots with charge >80% will have the least priority in charging
    // We set a dictionary priority
    // each bot will have two attributes for sorting [Class, dist]
    // Class 2 if Battery percentage is in [50%, 80%]
    // Class 1 if Battery Percentage is in [0%,50%)
    // Class 0 if Battery percentage is in (80%, 100%]
    // distance will be current value - Min_of_corp_class
    // The attributes for bot wih 70% charge on the basis of above protocol will be (2,10)
    //  Sorting will be done on the basis of these attributes using dictionary comparison
    // Ties will be arbitrarily broken
    overRide,List = set_override_by_user();// waits for user input for 2mins of inactivity. If
no action taken during 2 mins, overRide is automatically set as false and list is set as empty.
    if overRide:
        // Allow assignment of Power in which total power does not exceed the maxallowable
power.
        // If power assigned exceed max_allowable power
        heap = create_heap(List)// Create heap of the list input by user
        thresh = set_threshold(); Minimum number of bots that should be kept on charging
        Max_Load = get_Max(); Maximum Load of the charging House that the grid can hold
        batch = set_batch(); the batch value
        count = 0;
        Number_of_bots = len(heap)
        heap = create_heap();//Recreate the heap
        while(!Empty(heap)):
            curr = GP(Number_of_bots/batch);
            power_qunata = min(Max_Load/(curr*batch), max_req_load);
            //Assign maxLoad to all the bots in batch
            Number_of_bots -=batch;
            Max_Load -= power_quanta*curr;
            return ;
    heap = create_heap();// of all the bots which need to be charged
    thresh = set_threshold(); Minimum number of bots that should be kept on charging
    Max_Load = get_Max(); Maximum Load of the charging House that the grid can hold
    batch = set_batch(); the batch value
    count = 0;
```

```
    Number_of_bots = len(heap)
    heap = create_heap();//Recreate the heap
    while(!Empty(heap)):
        curr = GP(Number_of_bots/batch);
        power_qunata = min(Max_Load/(curr*batch), max_req_load);
        //Assigne maxLoad to all the bots in batch
        Number_of_bots -=batch;
        Max_Load -= power_quanta*curr;
    return;
def Charging_station_reassign():
    while True:
         priority_of_charge();
        wait(t_minutes)// set as 30 default. Can be changed
def Manual_Reboot():
    reset_and_reload_all_variables();
    Charging_station_reassign();
```

2. **part (b):**

Task: If a bot crashes/has low battery communication protocol can be used to allow a nearby free bot with sufficient energy levels to replace itself with the crashed bot.(Giving customer information about the exchange at the same time). If no nearby bot is available, we can request a bot from the main docking/charging station to replace this bot. Then while the other bot is deployed/is on the way, the current bot can go into low power mode where it would slowly follow the customer while the other bot arrives. It keeps checking if the deployed bot is near and is available/free, then it would turn off and proceed to give control to the other bot(change the variable of being used or not)

Description of code:

The following code will check if the current bot
that is being used has a healthy battery so as to accompany the customer
throughout their shopping. nearby_bot() is a function that would check for
nearby bot using transmitters and receivers, and would put them in a
min heap on the basis of the minimum distance of it to current bot, provided
it is free. Also every bot has an attribute named state which is an integer
which tells whether the bot is free, not free or in low power mode

first of all keep an int that would monitor the state of the bot
state = 0 means bot is being used
state = 1 means bot is free
state = 2 means bot is low in power

main() function simply executes a while loop, that keeps on checking
the bot state, and if it is being used, would check the battery levels
of the current bot. If the battery of the bot is above a threshold, it would
wait a fixed time and then run the while loop condition again. This timer
is used to save battery life. If the bot's battery is below a certain threshold,
then it calls nearby_bot() function to get a bot that is available and nearby this
bot. If the function does not return a NULL, means that a bot is found, then it
proceeds to ask the bot to come to the current location. Else it would contact the
charging station to ask for bots. During this procedure, the current bot will be
slowed down a bit to preserve energy and keeps on checking the distance with the new bot.
When the new bot is closer than a certain threshold, it would ask the customer
to transfer their items on to the new one, and go to low power state.

```
// first of all keep an int that would monitor the state of the bot
//state = 0 means bot is being used
//state = 1 means bot is free
//state = 3 means bot is low in power
```

```
//check int state

BOT nearby_bot(){
    //BOT is a class that can be defined to store variables

    heap = create_heap() //initialise a heap which stores the bots in order of the minimum
distance from
                         // the current bot

    //fill the heap with nearby bots, using the transmitters and receivers
    while(heap.is_notempty()){
        new_bot = heap.head()
        if(distance bw 2 bots is less than a threshold && newer_bot.state == 1){
            //that means a nearby bot is found and it is free
            return new_bot;
        }
        //else keep looking
    }
    //if the code comes here, that means no nearby bot is found

    return NULL;
}

int main(){


    while(current_bot.state = 0){
        //check for battery level
        if(current_bot.battery > threshold){
            wait(for some t minutes); //t will be decided based on experience, for now
                                      // keep it at 10 minutes
        }

        if(current_bot.battery < threshold){
            current_bot.state = 2;
            if(nearby_bot(current_bot) != NULL){
                //means a nearby bot to be used is present
                call_bot();  //this will call the other bot to the current bots location
                slow_down_current_bot();
                distance_from_new_bot = calculate_this_from_transmitters;
                time = current time;
                int stuck = 0;
                while(distance_from_new_bot < threshold_of_nearby_distnce){
                    //this while loop keeps on checking if the bot is near or not
                    //if code enters this while loop, means new bot is still far

                    //so recalculate the distance
                    distance_from_new_bot = calculate_this_from_transmitters;

                    //also need to check if the other bot is not stuck
                    if(current_time > threshold_time){
                        stuck = 1;
                        break;
                    }
                }
                //if while loop is exited, either bot is close to person, or
                //the new bot got stuck and alarm should be raised
                if(stuck){
                    //means new bot is stuck
                    activate_alarm();
```

```
                }
                else{
                    //exit the while loop and stay in low battery state
                    break;
                }
            }

            else{
                //this means no nearby bot is available
                //in this case inform the main charging station and see if there is any bot
                //that is available

                //if available, then do the same, send that bot here and
            }
        }

    }

    //check if the bot is above some battery
    while(current_bot.state = 2){
        //check for battery levels
        if(current_bot.battery > threhold_2){
            current_bot.state = 1;
            break;
        }
    }
}
```

3. **part (c)**

Task: For customers with baggage needs of more than 40Kg Communication protocol can be used to bring a new bot that can be merged with the current bot(One bot will follow other) and satisfy the requirement

Description of the code

1. Some variables: WEIGHT_LIMIT - This is the min weight after which the bot would consider itself overweight. In this case, it is set to 40.FOLLOW_DISTANCE - This is the max distance that would be maintained among the two following bots. In this case, it is set to 2 metre.
2. If the bot is overweight, then the bot would first lock its motion and send the user a notification implying that it would only move if the weight is under the limit.
3. Then it would ask the user to either call another bot or reduce the weight.
4. If the user chooses to call another bot, then another bot is called to go to the current bot location.
5. The new bot would continuously update its path every 5 sec so as to reach the real-time location of the caller bot.
6. The new bot upon reaching would start following the caller bot.
7. The first bot would notify the user to reduce weight from the old bot and distribute it to the new bot.
8. If the weight is under the limit, then the bot would unlock its motion.

```
def bot_call(tofollow):
    ##################################
    #FOLLOW_DISTANCE is a fixed variable showing what max distance when
# we can say that the new bot is with first bot
    ##################################
    FOLLOW_DISTANCE = 2 metre
    newbot = Bot()
    myloc = newbot.location
    toloc = tofollow.location
    while ( Distance(myloc,toloc) < FOLLOW_DISTANCE):
```

```
    path = PathFindingAlgorithm(myloc,toloc)
    newbot.move(path, 5 sec) # go on that path for 5 sec, then update the path
    myloc = newbot.location
    toloc = tofollow.location
```

```
if thisbot.load_weight > WEIGHT_LIMIT:
    thisbot.lockmotion() # bot would refuse to move
    thisbot.user_notify(" It seems that your weight limit has exceeded ")
    User_response = thisbot.user_option("Call another Bot", " Reduce Weight" )
    if user_response == "Call another Bot":
            bot_call(tofollow = thisbot)
        thisbot.user_notify("A new bot will be available soon. Please share the weight among
this bot and new upcoming bot")
    else:
        thisbot.user_notify("Please reduce weight to move forward")
    while(thisbot.load_weight > WEIGHT_LIMIT):
        wait(10 sec);
        thisbot.usernotify("Please reduce weight to move forward")
    thisbot.unlockmotion()
```

4. **part (d)**

Task: After a while, the low powered bot would try to find its way back to the charging dock. It may be possible that the low powered bot cannot find its way to the station, in that case it would shut down and manual assistance would be required to take it back

Description of the code:

1. Critical_limit is the battery level of the bot slightly before it is completely discharged (or its power is off).
2. If the battery of the bot is at the critical_limit and it has not reached the charging station, it will signal that it needs manual assistance sharing its location to the staff.
3. And finally, it will switch off its power.

```
if(thisbot.battery==critical_limit): #critical_limit is the battery level slightly before
complete discharge, like 2%
    if(thisbot.location!=charging_station.location):
        thisbot.signal_staff("Need_manual_assistance");
        thisbot.power=off;
```

part (e)

Task: Special tasks like loading material from the basement to a room in the mall using bots can be done using the Communication protocol. Bots with material going in the same rule will follow on after the other and unloading will be done in the room(like worker ants)

Description of code:

This function will request the given list of bots to make a round trip to a particular store.

1. We fisrt find the path of the first bot from the current location to the destination location namely "loc".
2. We call the follow method of the of each bot other than the first bot. We set the follow parameter of Bot "i" as Bot "i-1".
3. Then we instruct the first bot to go the particular destination location "loc".

4. Rest all the bots will just follow the bot immediatly in front of it.
5. We set the follow parameter of avalilable_bots[1:bot_count-1] to NONE. (First bot is bot "0")

```
def request_bots(loc, bots_req):    # 'loc' is the location of the goods that the bots have to
carry
    avalilable_bots = get_freebots(bots_req)    # This function will return a list of bots_req
number of available bots
    bot_count = len(avalilable_bots)
    first_bot = avalilable_bots[0]
    bot_location = first_bot.curr_location()
    path = get_path(bot_location, loc)
    for i in range(1, bot_count):
        # Bot 'i' will follow Bot 'i-1'
        available_bots[i].follow(available_bots[i-1])
    first_bot.move(path)
    # All the other bots are going to just follow the bot just infront of it so all the bots
will reach the destination.

    for i in range(1, bot_count):
        # Bot 'i' will stop following Bot 'i-1'
        available_bots[i].follow(NONE)
```

This function will request the given list of bots to make a round trip to a particular store.

1. We fisrt find the path of the first bot from the current location to the destination location namely"loc".
2. We call the follow method of the of each bot other than the first bot. We set the follow parameter of Bot "i" as Bot "i-1".
3. Then we instruct the first bot to go the particular destination location "loc".
4. Rest all the bots will just follow the bot immediately in front of it.
5. Then the bots will wait at the location "loc" for the goods to unload.
6. Then the shop owner will select the prompt so that the bots can return to the initial location.
7. We calculate the path from "loc" to initial destination.
8. We instruct the first bot to go the initial location.
9. We set the follow parameter of list_bots[1:bot_count-1] to NONE. (First bot is bot "0")

```
def round_trip(list_bots, loc):
    bot_count = len(list_bots)
    first_bot = list_bots[0]
    innitial_location = first_bot.curr_location()
    path = get_path(innitial_location, loc)
    for i in range(1, bot_count):
        # Bot 'i' will follow Bot 'i-1'
        list_bots[i].follow(list_bots[i-1])
    first_bot.move(path)
    # All the other bots are going to just follow the bot just infront of it so all the bots
will reach the destination.

    user_prompt()   # Wait for the goods to be unloaded at the site. When the bots are free to
go the shop owner will press OK on prompt.
    return_path = get_path(loc, initial_location)
    first_bot.move(path)
    # All the other bots are going to just follow the bot just infront of it so all the bots
will reach the destination.
    for i in range(1, bot_count):
        # Bot 'i' will stop following Bot 'i-1'
        list_bots[i].follow(NONE)
```

part (f)

Task: Bot performance and health can be checked using communication protocol.

Description of code:

1. We check the various parameters of the bot
2. If they are within threshold we report bot being healthy
3. Otherwise we report the issues

```
def Check_bot_health(bot):
    healthy = True
    avg_start_time = bot.Last_working_stats.start_time.avg()
    if avg_start_time>max_threshold_start_time:
        healthy = False
        print("Check bots starting time")
    avg_battery_perf_time = bot.Last_working_stats.(battery_drop_per_hour).avg()
    if avg_battery_perf_time> min_threshold_perf_time:
        healthy = False
        print("Replace Battery")
    avg_charging_time = bot.Last_working_stats.battery_increase_per_hour_per_watt.avg()
    if avg_charging_time > max_threshold_charging_time:
        healthy = False
        print("Replace Charging jack")
    Number_of_times_stuck = bot.num_times_stuck_per_month
    if Number_of_times_stuck>max_allowable_stuck_threshold:
        healthy = False
        print("Replace sensing related parts")
    if healthy:
        print("Health performace check done. Bot is healthy")
        print("The average starting time is ", avg_start_time)
        print("The average battery performance is ",avg_battery_perf_time)
        print("The average charhing time is ",avg_charging_time)
        print("The number of times the bot got stuck during past 30 days
is",Number_of_times_stuck)
    print("Performance check done")
```

part (g)

Task:

Some customers can leave (some materials which they initially thought of buying but later on lost interest). This can be redirected to a docking station where there is a clearance

1. In this code we are checking if the bot is empty after use.
2. If it is not empty, then we redirect it to unloading station

```
def Redirect():
    redirect = False
    if len(self.List)!=0:
        redirect = True
    if redirect:
        #Goto unloading station.
```

part (h)

Task: If more than “m” bots are present in a radius of R, it will generate a signal and unused bots will be redirected to other places where there are less number of bots to avoid congestion. This will happen only during working hours.

Description of the code:

1. The function gets a list of all the clusters of the bots
2. In each cluster if the number of bots around the centre of cluster within radius R exceeds m, then we redirect these bots to loc and num or preassigned places which ever place has less number of bots
3. Assignment is done one-by-one so that no place gets crowded
4. If no assignment can be done(Too many free bots), then the bots are sent to charging station

```
def Relocate():
    loc = [chg_station_loc]
    num = [threshold]
    List = get_location_of_all_working_bots()
    clusters = k_means_cluter(List,k)#k can be tuned
    for cluster in clusters:
        centre = cluster.centre#centre of cluster
        count = num_of_bots_around(centre)
        if count >threshold:
            redirect_free_bots_to_new_location(loc,num)
        else:
            loc.append(centre)
            num.append(threshold-count)
 def redirect_free_bot(loc, num):
    sort(num,lic) in dict order
    assigned_centres = get_assigned_centres()# preassigned list where bots can form a cluster
    for centre in assigned_centre:
        count = num_of_bots_around(centre)
        if count <threshold and threshold-count>num[0]:
            send_bot_to_this cluster()
 def redirect_free_bots(loc,num):
    while(all_bots_not_redirected):
        redirect_free_bot(loc,num)
```

part (i)

Task: If there are two bots, one bot with a user and the other bot is within 2m of its position and a signal will be transmitted and these bots will be kept informed about each other to avoid collisions

```
# If there are two bots, one bot with a user and the other bot is within 2m of its position and
a signal will be transmitted and these bots will be kept informed about each other to avoid
collisions

def CloseEnough(bot_with_customer):
  Minimum_distance=2
  for bot in bots:
    if bot!=bot_with_customer and Distance(bot, bot_with_customer) <= Minimum_distance:
      bot_with_customer.send_signal(bot,"Too close")
      while ( Distance(bot,bot_with_customer) <= Minimum_distance):
        bot.move_away(bot_with_customer)
        if(bot.battery=='critical')
          bot.sleep();
```

```
          break;
      if(Distance(bot, bot_with_customer) > Minimum_distance):
        print("Bot with id %d has moved away from bot with id %d" % bot.id() %
bot_with_customer.id())
        print("Task succesful")
      else:
        print("The bot with id %d has low battery, send replacement" % bot.id())
        print("Task unsuccessful")
```

part (j)

Task: If the number of working bots drops below a threshold the communication protocol can be used to deploy reserve bots to fill the gap and the security will be alerted

Description of code:

1. Let Threshold be the minimum number of bots which are required to be in functioning state at any given time
2. Then we can maintain a counter (initialized to zero) and check for all working bots if they are in critical state (not in working condition) or not and if we found a bot in critical situation increase the counter value by 1
3. Then compare the counter value with threshold and if it less inform the staff that they need to deploy another batch of bots and the bots in critical state will be set for charging
4. Here in the code "bots" are the currently working bots, We remove a bot which is in critical state from "bots" and later when a batch is released we add the corresponding batch to the "bots" again.

```
#Code
Threshold = Min_Working_Bot
No_of_bots_in_critical_situtation = []
Need = False
for bot in bots:
    if(bot.battery==critical_limit):
        No_of_bots_in_critical_situtatio.append(bot)
    Bots.remove(Bot)


if(len(No_of_bots_in_critical_situtation) < Threshold):
    this.signal_staff("Need_to_deploy_more_bots")
    Need = True;

Charge(No_of_bots_in_critical_situtation)

# if singnal is recived, a batch of bot will be released

if(Need == True):
    #release a batch of bot
    relased_batch_of_bots = Relase()
    Bots.add(relased_batch_of_bots)
    Need = False
```

part (k):

Task: Sometimes customers need different baskets for different purchases. Communication protocol can be used to allot those bots to the customer which have the requested basket attached to them(If any such bot is already free)

```
// first of all keep an int that would monitor the state of the bot
//state = 0 means bot is being used
//state = 1 means bot is free
```

```
// the size of the basket with the bot is stored in the 'size' attribute of Bot object.
//Different basket sizes are stored in a sorted list named basket_sizes


// Helper function to find bots nearby
// Assuming all bot objects are stored in a list named bot_list
BOT get_nearby_bot(req_size = size_value){

    for bot_obj in bot_list{
        if(distance bw 2 bots is less than a threshold && bot_obj.state == 1){
            if (bot_obj.size=size_value)
            //that means a nearby bot is found and it is free
            return bot_obj;
        }
        //else keep looking
    }
    //if the code comes here, that means no nearby bot is found
    return NULL;
}

// Listener Function
@listen
def bot_requested(req_size = k):
    // We want to return the nearby free bot with the desired size, if not available, we want to
return a free bot with a basket marginally biggen than requested.
    size_fit_bot = get_nearby_bot(req_size=k) ## Bot, if available, of the required size.
    if size_fit_bot is not NULL:
        return size_fit_bot #If found, return that bot

    // If a bot of the require size is not found, find bigger bots, starting with marginally
bigger ones.
    for size_value in basket_sizes:
        // skip sizes lesser than requested.
        if size_value <= k:
            continue
        bigger_bot = get_nearby_bot(req_size = size_value)
        if bigger_bot is not NULL:
            return bigger_bot // return a bot as soon as it is found.

    // If the execution reaches this point, it means that no bot bigger than or equal to
requested size is available.
    // In that case, we can ask the user if a smaller basket will work and if there is no
otherway, we can combine two smaller bots as mentioned in part(c).
```

part (l)

Task: If a family/group has come to a mall, the corresponding bots assigned to the family/group members can be put in communication with each other to let family/group members know of each other's purchases and inform them of multiple purchases of the same item by different members of the family.

Description of the code:

1. List is the list of items currently in the bot
2. Attached_bots is the list of bots which have been attached to the current bots
3. We search for all items inside each of these bots
4. If the current item to be inserted is already present we ask user for confirmation
5. If user confirms, the item is added

6. If the item is not present in any of the attached bot lists the we search in the current bots list
7. If the item is present, we ask the user for confirmation
8. If user confirms, the item is added
9. If the item is not present in current bots list also, the item is added without any prompt.

```
self.List = []
Attatched_bots = get_attached_bots()# This returns the attatched bots
def insert(item):
    ins = True
    for bot in Attached_bots:
        if item in bot.List:
            disp("This item is already present in the catalogue of user:", bot.user_name,"Do you
want to add it")
            ins = prompt_user()
            break
    if ins and item in self.List:
        disp("This item is already present in the your catalogue. Do you want to add it")
        ins = prompt_user()
    if ins:
        List.append(item)
        update_to_main_server(self, item)
def Display():
    for bot in Attached_bots:
        disp(bot.user_name)
        for item in bot.List:
            disp(item)
            wait(1s)
        return;
```

part (m):

Task: Communication protocol can be used to put the Bot in **standby mode**(in secure mode-No item can be pulled out/put in) when the customer is in the lavatory. The bots will wait for the customer in a specified waiting area specified by the mall close to the lavatory to avoid clustering in the lavatory

```
def standby(BOT mybot):
    toloc = waitarea.location
    mybot.lockmotion() # bot stays there
    # Also update in communication protocaol database
    user_response = mybot.user_option("Come to user", "Stay in standby")
    if user_response == "Come to user" :
        myloc = mybot.location
        toloc = tofollow.location
        FOLLOW_DISTANCE = 1 metre
        while ( Distance(myloc,toloc) < FOLLOW_DISTANCE):
            path = PathFindingAlgorithm(myloc,toloc)
            newbot.move(path, 5 sec) # go on that path for 5 sec, then update the path
            myloc = newbot.location
            toloc = tofollow.location
```

other part

Description of code :

1. First it is creating a heap which contains same new location bots as current bot will jump to in next move
2. Then it is searching for bots which is coming from different directions

3. And then allowing current bot to move and new_bot has to wait for some t time interval till current bot release its new location
4. And then similarly second bot has to wait till previous bot move from new location and so on;

```
void encounter_of_bot(bot current_bot){
    heap = create_heap(); //initialize heap which contain same new location bots as current_bot
    while(heap.is_notempty){
        new_bot = heap.head();
        if(coming from different directions){
            current_bot = allow_move;
            new_bot = allow_after_a_while(till previous bot release new location);
        }
    }
}
```