```
class order():
    def __init__(self, number):
        self.number = number
        self.a_time = 0
        self.sections = []
        self.qualified_group = []
    def set_group(self,group):
        self.qualified_group.append(group)
    def add_section(self,value):
        self.sections.append(value)
    def update_time(self, start, finish, interval):
        self.start = start
        self.finish = finish

class sub_order():

     def __init__(self, index ):
        #self.order_number = order_number
        #self.line = line
        self.index = index
        self.tasks = {}
        self.start = []
        self.finish = []
     def update_time(self,attr, task_type):
        self.tasks[attr] = task_type

f= open("output.csv","a")
f.write('Order, Assigned Group, Start Date, Start Time, Finish day, Finish time, 
Assembly Time, Status \n')


all_sections = []
map_order = {}


def map_oder_input(sub):
    map_order[sub.Order].add_section(sub)
    map_order[sub.Order].a_time += math.ceil(getattr(sub,'Real Time'))
def read_data_assembly():

#order_input = pd.read_csv('order_input.csv')
    fields = ['Order', 'Line', 'Status', 'Sched. Ship Date',
              'Real Status' , 'Real Time', 'Promised' , 'ISSUE','Missing Materials' ]
    #section_input = pd.read_csv("Axis-Assembly-Input.csv", skiprows = 1)
    assembly_input = pd.read_csv("Schedule-10.01.2019.csv", skipinitialspace=True, 
usecols=fields)
    priority_rank = {
```

```
    'High Priority': 1,
    'Priority' : 2,
    'Regular': 3

    }
    status_rank = {
        'Wiring Started': 1,
        'Machine Shop Finished' : 2,
        'Machine Shop Started': 3,
        'Scheduled/Released': 4

        }
    good_value = ['0', float('NaN')]
    for index, row in assembly_input.iterrows():
        if(row['Status'] in status_rank and row['ISSUE'] in good_value):
            if(row['Order'] not in map_order):
                ord = order(row['Order'])
                map_order[row['Order']] = ord
            sub = sub_order(index)

            for value in fields:
                setattr(sub, value, row[value])
            setattr(sub, 'priority', priority_rank[sub.Promised])

            if status_rank[sub.Status] == 1 :
                sub.Status = 1
            elif status_rank[sub.Status] == 2:
                if getattr(sub,'Missing Materials') in good_value:
                    sub.Status = 2
                elif getattr(sub,'Missing Materials') == '+Cartridge':
                    sub.Status = 3
                else :sub.Status = 7
            elif status_rank[sub.Status] == 3:
                if getattr(sub,'Missing Materials') == '+lens':
                    sub.Status = 4
                if getattr(sub,'Missing Materials') == 'Material+Cartridge':
                    sub.Status = 5
                if getattr(sub,'Missing Materials') ==
'Material+lens+Cartridge+Housing':
                    sub.Status = 6
                else :sub.Status = 7
            else :sub.Status = 7
            map_oder_input(sub)



#number_of_orders = len(order_input)
capacity = {
    1: 21,
    4: 21,
    7: 21,
    10: 14,
    12: 14


}

useage= {
    1: 0,
    4: 0,
    7: 0,
    10: 0,
    12: 0
    }
def assign_date(assembly_orders,f, status):
    assembly_orders.sort(key = attrgetter('group', 'start'), reverse=False)
    output = ['number', 'group', 'start_day', 'start', 'finish_day', 'finish',
'a_time', 'Status']
    line = []
    for o in assembly_orders:
        if(status > 1):
            o.start = o.start + useage[o.group]
            o.finish = o.start + o.a_time
        value = math.floor((o.start)/ (60*capacity[o.group]))
        setattr(o, 'start_day', value )
        value = math.floor((o.finish) / (60*capacity[o.group]))

        setattr(o, 'finish_day', value )
        useage[o.group] = useage[o.group] +  o.a_time
        for i in output:
            line += str(getattr(o,i))
            line += ","
        line += "\n"
        line = ''.join(line)
    f.write(line)

        #print (s.start[m], s.finish[m])


def generate_assembly_schedule():
    assembly_orders = []
    for i in range(1,7):
        for index, ord in map_order.items():
              num = []
              [num.append(s.Status) for s in ord.sections]
              setattr(ord, 'Status', max(num))
            if ord.Status == i :
                assembly_orders.append(ord)
            foo = [1,4,7,10,12]
            random.shuffle(foo)
            if(ord.Status == 1):
                ord.qualified_group = foo[0]
            else:
              ord.qualified_group = foo[: random.randint(1, 4)]
        print("Number of Order with assembly status {} is {} ".format(i, 
len(assembly_orders)))
        if len(assembly_orders) > 0:
            AssemblyScheduling(assembly_orders)
            assign_date(assembly_orders,f, i)
        assembly_orders.clear()
    print("fnish")



read_data_assembly()
generate_assembly_schedule()
print('finish')

def AssemblyScheduling(all_orders):

       # Instantiate a cp model.

    horizon = sum(o.a_time for o in all_orders);
    horizon =math.ceil(horizon )
    model = cp_model.CpModel()
    # Variables
    makespan = model.NewIntVar(0, horizon, 'makespan')
    x = {}
    start= {}
    y = {}
    inter = []
    for o in all_orders:
        t = []
        for p in all_orders :
            if p.number !=  o.number:
                try:
                    inter = intersection(o.qualified_group, p.qualified_group)
                    if(len(inter) > 0):
                        for r in inter:
                            y[(o.number, p.number, r)] = model.NewBoolVar('y[%i,%i,
%i]' % (o.number, p.number, r))
                except TypeError:
                    if o.qualified_group ==  p.qualified_group :
                        inter = o.qualified_group
                        y[(o.number, p.number, inter)] = model.NewBoolVar('y[%i,%i,
%i]' % (o.number, p.number, inter))
        try:
            for g in o.qualified_group:
                x[(o.number,g)] = model.NewBoolVar('x[%i,%i]' % (o.number, g))
            start[o.number] = model.NewIntVar(0,horizon,'start[%i]' % o.number)
        except TypeError:
            g = o.qualified_group
            x[(o.number,g)] = model.NewBoolVar('x[%i,%i]' % (o.number, g))
            start[o.number] = model.NewIntVar(0,horizon,'start[%i]' % o.number)


    ## Constraints

    # Each task is assigned one qualified group.
    for o in all_orders:
        try:
            model.Add(sum(x[(o.number, g)] for g in o.qualified_group) == 1)
        except TypeError:
            model.Add(x[(o.number, o.qualified_group)]  == 1)

    for o in all_orders:
        for p in all_orders :
            if p.number > o.number:
                try:
                    inter = intersection(o.qualified_group, p.qualified_group)
                    if(len(inter) > 0):
                        for r in inter:
                             model.Add(start[o.number] >= start[p.number] +
p.a_time).OnlyEnforceIf(y[(o.number, p.number, r)])
                             model.Add(start[p.number] >= start[o.number] +
o.a_time).OnlyEnforceIf(y[(p.number, o.number, r)])
                             model.Add(y[(o.number, p.number, r)] + y[(p.number,
o.number, r)] >= x[(o.number, r)] + x[(p.number, r)] - 1)
                             model.Add(y[(o.number, p.number, r)] + y[(p.number,
o.number, r)] <= x[(o.number, r)])
                             model.Add(y[(o.number, p.number, r)] + y[(p.number,
o.number, r)] <=  x[(p.number, r)])

                            #model.add(y[(o.number, p.number, r)] ==
0).OnlyEnforceIf(x[(o.number, r)].Not())
                            #model.add(y[(o.number, p.number, r)] ==
0).OnlyEnforceIf(x[(p.number, r)].Not())

                            #model.Add(y[(o.number, p.number, r)] + y[(o.number,
p.number, r)] == 1).OnlyEnforceIf(x[(o.number, r)] and x[(p.number, r)])
                except TypeError:
                    if o.qualified_group ==  p.qualified_group :
                        r = o.qualified_group
                        model.Add(start[o.number] >= start[p.number] + 
p.a_time).OnlyEnforceIf(x[(o.number, r)] and x[(p.number, r)] and y[(o.number, 
p.number, r)])
                        model.Add(start[p.number] >= start[o.number] + 
o.a_time).OnlyEnforceIf(x[(o.number, r)] and x[(p.number, r)] and y[(o.number, 
p.number, r)].Not())

    ## Total task size for each worker is at most total_size_max
    ##for i in all_workers:
    ##    model.Add(sum(sizes[j] * x[i][j] for j in all_tasks) <= total_size_max)

    ## Total cost
    [model.Add(makespan >= start[o.number] + o.a_time)  for o in all_orders]
    model.Minimize(makespan)

    solver = cp_model.CpSolver()
    solver.parameters.max_time_in_seconds = 5.0
    status = solver.Solve(model)
    print('Status is ', solver.StatusName(status))
    if status == cp_model.FEASIBLE:
        print('Feasible sol = %i' % solver.ObjectiveValue())
        assign_solution(solver, all_orders, start, x)
    if status == cp_model.OPTIMAL:
        print('Optimal sol = %i' % solver.ObjectiveValue())
        assign_solution(solver,all_orders, start, x)
```