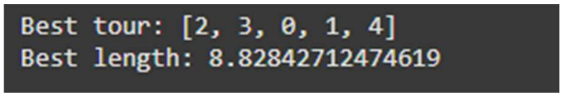
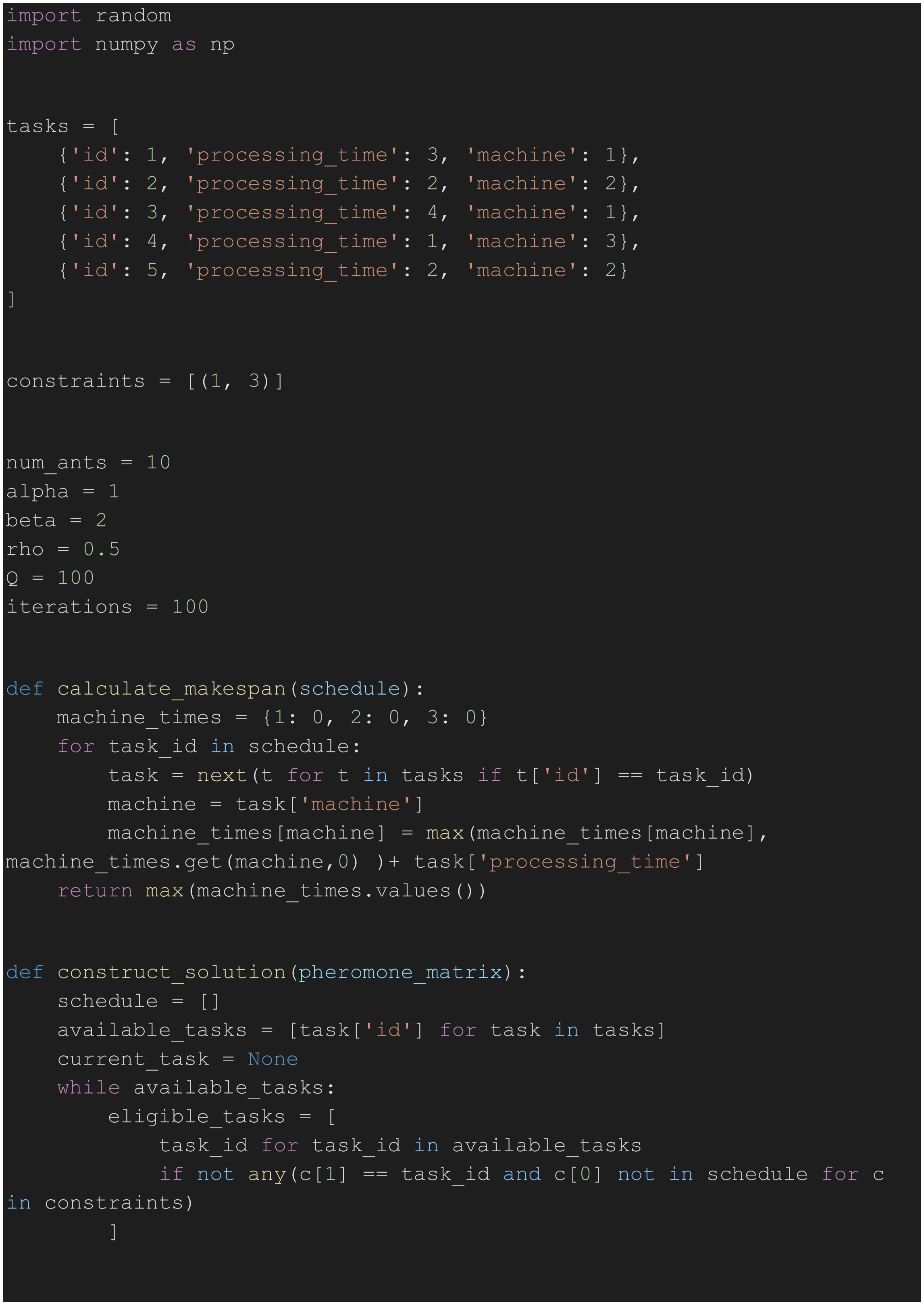
Lab Programmes
first Question

```
import numpy as np
import random


def distance(city1, city2):
  return np.sqrt((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)

def initialize_pheromones(num_cities, initial_pheromone):

  return np.full((num_cities, num_cities), initial_pheromone)

def calculate_probabilities(current_city, unvisited, pheromones,
distances, alpha, beta):

  probabilities = []
  for city in unvisited:
    tau = pheromones[current_city][city] ** alpha
    eta = (1 / distances[current_city][city]) ** beta
    probabilities.append(tau * eta)
  probabilities = np.array(probabilities)
  return probabilities / probabilities.sum()

def construct_solution(num_cities, pheromones, distances, alpha, beta):

  unvisited = list(range(num_cities))
  current_city = random.choice(unvisited)
  unvisited.remove(current_city)
  tour = [current_city]
  while unvisited:
    probabilities = calculate_probabilities(current_city, unvisited,
pheromones, distances, alpha, beta)
    next_city = random.choices(unvisited, weights=probabilities)[0]
    tour.append(next_city)
    unvisited.remove(next_city)
    current_city = next_city
  return tour

def update_pheromones(pheromones, all_tours, distances, rho, Q):

  pheromones *= (1 - rho)  # Evaporation
  for tour, tour_length in all_tours:
    pheromone_increase = Q / tour_length
    for i in range(len(tour)):
      from_city = tour[i]
      to_city = tour[(i + 1) % len(tour)]  # Circular tour
      pheromones[from_city][to_city] += pheromone_increase
      pheromones[to_city][from_city] += pheromone_increase

def calculate_tour_length(tour, distances):

  return sum(distances[tour[i]][tour[(i + 1) % len(tour)]] for i in
range(len(tour)))

def ant_colony_optimization(cities, num_ants, alpha, beta, rho, Q,
iterations, initial_pheromone):

  num_cities = len(cities)
  distances = np.array([[distance(cities[i], cities[j]) for j in
range(num_cities)] for i in range(num_cities)])
  pheromones = initialize_pheromones(num_cities, initial_pheromone)
  best_tour = None
  best_length = float('inf')
  for _ in range(iterations):
    all_tours = []
    for _ in range(num_ants):
      tour = construct_solution(num_cities, pheromones, distances,
alpha, beta)
      tour_length = calculate_tour_length(tour, distances)
      all_tours.append((tour, tour_length))
    if tour_length < best_length:
      best_tour = tour
      best_length = tour_length
    update_pheromones(pheromones, all_tours, distances, rho, Q)
  return best_tour, best_length


if __name__ == "__main__":
  cities = [(0, 0), (2, 0), (2, 2), (0, 2), (1, 1)]  # Define city
coordinates
  num_ants = 10
  alpha = 1.0
  beta = 2.0
  rho = 0.5
  Q = 100
  iterations = 100
  initial_pheromone = 1.0
  best_tour, best_length = ant_colony_optimization(cities, num_ants,
alpha, beta, rho, Q, iterations, initial_pheromone)
  print("Best tour:", best_tour)
  print("Best length:", best_length)
```

output:

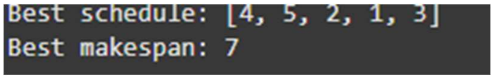


```
Best tour: [2, 3, 0, 1, 4]
Best length: 8.82842712474619
```



```
import random
import numpy as np


tasks = [
    {'id': 1, 'processing_time': 3, 'machine': 1},
    {'id': 2, 'processing_time': 2, 'machine': 2},
    {'id': 3, 'processing_time': 4, 'machine': 1},
    {'id': 4, 'processing_time': 1, 'machine': 3},
    {'id': 5, 'processing_time': 2, 'machine': 2}
]


constraints = [(1, 3)]


num_ants = 10
alpha = 1
beta = 2
rho = 0.5
Q = 100
iterations = 100


def calculate_makespan(schedule):
    machine_times = {1: 0, 2: 0, 3: 0}
    for task_id in schedule:
        task = next(t for t in tasks if t['id'] == task_id)
        machine = task['machine']
        machine_times[machine] = max(machine_times[machine],
machine_times.get(machine,0) )+ task['processing_time']
    return max(machine_times.values())


def construct_solution(pheromone_matrix):
    schedule = []
    available_tasks = [task['id'] for task in tasks]
    current_task = None
    while available_tasks:
        eligible_tasks = [
            task_id for task_id in available_tasks
            if not any(c[1] == task_id and c[0] not in schedule for c
in constraints)
        ]
        if current_task is None:
            current_task = random.choice(eligible_tasks)
        else:

            probabilities = []
            for next_task in eligible_tasks:
                pheromone = pheromone_matrix[current_task -
1][next_task - 1]
                heuristic = 1 / (next(t for t in tasks if t['id'] ==
next_task)['processing_time'] + 1e-6)
                probability = (pheromone ** alpha) * (heuristic **
beta)
                probabilities.append(probability)


            probabilities = np.array(probabilities) /
np.sum(probabilities)


            current_task = np.random.choice(eligible_tasks,
p=probabilities)

        schedule.append(current_task)
        available_tasks.remove(current_task)
    return schedule


num_tasks = len(tasks)
pheromone_matrix = np.ones((num_tasks, num_tasks)) * 0.1

best_schedule = None
best_makespan = float('inf')

for _ in range(iterations):

    all_schedules = [construct_solution(pheromone_matrix) for _ in
range(num_ants)]


    makespans = [calculate_makespan(schedule) for schedule in
all_schedules]


    min_makespan = min(makespans)
    if min_makespan < best_makespan:
        best_makespan = min_makespan
        best_schedule = all_schedules[makespans.index(min_makespan)]
    delta_pheromone_matrix = np.zeros((num_tasks, num_tasks))
    for schedule, makespan in zip(all_schedules, makespans):
        for i in range(len(schedule) - 1):
            task1, task2 = schedule[i], schedule[i + 1]
            delta_pheromone_matrix[task1 - 1][task2 - 1] += Q /
makespan

    pheromone_matrix = (1 - rho) * pheromone_matrix +
delta_pheromone_matrix

print("Best schedule:", best_schedule)
print("Best makespan:", best_makespan)
```

output



```
Best schedule: [4, 5, 2, 1, 3]
Best makespan: 7
```