Problem 1: Trace

Part A
1. Finds the minimum element between indices $i$ and $n-1$, inclusive and stores its index as $m$
2. Swaps the element at index $i$ with the minimum element between indices $i$ and $n-1$ if the minimum is not at index $i$
3. Sorts lst's elements in nondecreasing order

Part B
Line 1: [11, 21] 8 16
Line 2: [11, 21] 3 5
Line 3: [12, 23] 14 15
Line 4: [12, 23] 3 5

Problem 2: Data Structures - Join

```python
def join(a, b):
    joined_dict = {}
    for key in a:
        if key in b:
            values_tup = (a[key], b[key])
            joined_dict[key] = values_tup
    return joined_dict
```

Problem 3: Nested Data Structures - Flights Data

```python
def early_bird(filename):
    with open(filename, 'r') as f:
        earliest_flights = {}
        for line in f:
            airport_info = line.strip().split(',')[1]
            departure_city = airport_info[0].lower()
```
if departure_city not in earliest_flights:
    earliest_flights[departure_city] = {}
# now go through flights
for flight in airport_info[1:]:    # skip over depart city
    arrival_city, time = extract_arrival_info(flight)

    # the key moment
    inner = earliest_flights[departure_city]
    if arrival_city not in inner:
        inner[arrival_city] = time
    else: # can omit the else and this still works
        if inner[arrival_city] > time:
            inner[arrival_city] = time
return earliest_flights

def extract_arrival_info(flight):
    if flight[3].isdigit(): # 3 letter airport code
        return flight[:3].lower(), int(flight[3:])
    else:
        # 4 letter airport code
        return flight[:4].lower(), int(flight[4:])

Problem 4: Graphics Program - Changing Mindsets

def changing_mindsets(window):
    with open('2015.txt', 'r') as f:
        for line in f:
            data = line.split()
            wealth = float(data[1])
            health = float(data[2])
            population = float(data[3])
            x = wealth * window.width
            y = (1-health) * window.height
            radius = math.sqrt(population / math.pi)
            circle = GOval(2 * radius, 2 * radius)
            window.add(circle, x - radius, y - radius)
Problem 5: Classes - Building a Spaceship

Part A

class Spaceship:

    def __init__(self, initial_food):
        self.food_on_board = initial_food
        self.crew_map = {}
        self.planets_visited = []

    def board(self, crew_member_name, food_per_day):
        ""
        Boards a crew member with the given food intake.
        This crew member will now consume food during trips.
        Input:
            crew_member_name (str): Name of the crew member to board
            food_per_day (int): Daily food consumption in pounds
        Output: This function should not return any value
        ""
        self.crew_map[crew_member_name] = food_per_day

    def get_planets_visited(self):
        ""
        Returns a list of visited planets, in order of visit.
        Output:
            visited_planets (list): A list of visited planets
        ""
        return self.planets_visited

    def fly_to(self, planet_name, days_required):
        ""
        Attempts to fly to a planet, which takes the given number of days.
        Input:
            planet_name (str): name of the planet to fly to
            days_required (int): number of days required to fly to planet
        Output:
            flight_status (boolean): Whether or not flight was successful
        ""
        food_remaining = self.food_on_board
        for crew_member in self.crew_map:
            total_food_per_member = self.crew_map[crew_member] *
            days_required
            food_remaining -= total_food_per_member
        if food_remaining < 0:
return False

self.planets_visited.append(planet_name)
self.food_on_board = food_remaining
return True

Part B

SENTINEL = "DONE"
def populate_ship(ship):
    """
    Given an empty spaceship, prompt the user for information about
    the crew members and populate the spaceship.
    Input:
        ship (Spaceship): The spaceship to which new crew members should
        be added
    Output:
        This function does not return a value but directly modified the
        provided ship object
    """
    print("Welcome to the CS106AP Spaceport! Please enter information about
    one crew member at a time. When you're ready to take off, enter the string " +
    SENTINEL + ")
    while True:
        crew_data = input("Please enter crew member name and daily food
        consumption: ")
        if crew_data == SENTINEL:
            break
        crew_data_parts = crew_data.split()
        name = crew_data_parts[0]
        consumption = int(crew_data_parts[1])
        ship.board(name, consumption)

    print("Ready for liftoff!")

Part C
def visitable_planets(starting_food, planet_dict):
    """
    This function prints a list of visitable planets
    given an initial amount of starting food.
    Input:
        starting_food (int): Number of pounds of food initially available
        planet_dict (dict): Associates destination planet with number of
days to arrive there
    Output: Prints a list of all visitable planets, in the order that they
    were visited.
    """
# Create a new spaceship
karels_ship = Spaceship(starting_food)

# Assume your function from Part B works as intended
populate_ship(karels_ship)

# Visit each planet until we run out of food
for planet, travel_time in planet_dict.items():
    success = karels_ship.fly_to(planet, travel_time)
    if not success:
        break

# Return the list of planets we had enough food to visit
print(karels_ship.get_planets_visited())

Problem 6: One-Liners

Part A: Code Reading

```python
>>> list(range(10, 5, -1))
[10, 9, 8, 7, 6]

>>> nums = list(range(5))
>>> [n * n for n in nums if n < 3]
[0, 1, 4]

>>> pairs = [('donut', 10), ('lemon', 5), ('apple', 6)]
>>> sorted(pairs, key=lambda pair: pair[1])
[('lemon', 5), ('apple', 6), ('donut', 10)]

>>> max(pairs, key=lambda pair: pair[1])
('donut', 10)

>>> d = {'ca': 'california', 'ak': 'alaska', 'tx': 'texas'}
>>> sorted(d.keys())
['ak', 'ca', 'tx']

>>> [s[1] for s in sorted(d.values())]
['a', 'e']

>>> pairs = [(3, 1), (4, 5), (6, 3)]
>>> sorted(pairs, key=lambda pair: pair[1])
[(3, 1), (6, 3), (4, 5)]
```
Part B: Code Writing

# a.
[s[0].upper() for s in strs if len(s) > 0]

# b.
[pt[0] for pt in points]

# c.
sorted(points, key=lambda pt: pt[1], reverse=True)