

Session 13

Working Railway Reservation System in Python built from the Session 1 to 6 specifications

This session turns the Railway Reservation System (RRS) documents from Sessions 1 to 6 into a program that runs. The scope statement fixed what the system does, the SRS numbered the requirements, the DFDs and ERD fixed the data, the modular design named the modules, and the screen designs fixed the prompts. The program here, `rrs.py`, implements modules 2 to 5 as a console application with JSON persistence, and every function in it traces back to a requirement id. The manual places this session under Software Change Management because the file you write now is the baseline that Session 14 changes under change control. A baseline you cannot trace is a baseline you cannot change safely.

Objectives

✗ Do not copy. Read for understanding and the viva

- Implement Train and Schedule Management, Search and Availability, Booking, and Cancellation and Refund from the Session 3 SRS
- Keep one function per menu action so each requirement maps to one named piece of code
- Enforce the business rules in code: 10-digit PNR, at most 6 passengers, no booking after departure, refund slabs, waitlist promotion in order
- Persist data in a JSON file so the demonstration in Session 14 survives a restart
- Record exactly what was left out of the SRS and why, so the limitations are a decision and not an accident

Problem Statement

■ Write in lab record

Develop "Railway Reservation System" as per specifications given in Sessions 1, 3, 4, 5 and 6.

Concept

✗ Do not copy. Read for understanding and the viva

Specification to code is a mapping, not a rewrite

Do not design again. Open the Session 3 SRS and the Session 5 structure chart and give each functional requirement a function name before writing any code. The traceability matrix below is written first and filled in as the code is written. In the viva the examiner will pick an FR id and ask where it lives; the answer must be a function name.

Why a console program and a JSON file

The SRS describes a web system with a payment gateway and SMS. A lab session has about three hours. A console menu removes the UI layer, and a JSON file removes the database server, while keeping every entity from the Session 4 ERD as a dictionary with the same attribute names. `rrs_data.json` holds three top-level maps: `trains`, `schedules`, and `bookings`. Route, Coach, Fare and Passenger records nest inside them, which is the same one-to-many structure the ERD draws.

Business rules belong in one place each

Each rule from the SRS appears in exactly one function: the refund slabs in `cancel_ticket`, the departure check in `book_ticket` and `find_schedules`, the 6-passenger limit as the constant `MAX_PASSENGERS`, PNR uniqueness in `new_pnr`, waitlist order in `promote_waitlist`. When Session 14 changes a rule, the diff touches one function.

Seat allocation model

A Coach record is `[coach count, seats per coach]` and seat labels are generated on demand as `3A1-17` (class, coach number, seat number). A schedule stores only which labels are taken and by which PNR, so a cancellation frees a label and the next waitlisted PNR takes it. Availability is per class for the whole run, not per segment, and that is listed as a limitation below.

Traceability Matrix

■ Write in lab record

FR ids follow the numbering of the Session 3 SRS. Module numbers follow Session 5. User Management (FR-1.1 to FR-1.6), Payment, Notification and Reports are not implemented; see Known Limitations.

Module (Session 5)	FR id (Session 3)	Requirement	Function in rrs.py	DFD process (Session 4)	Screen (Session 6)
2 Train and Schedule Management	FR-2.2	Admin adds a train with type and coaches per class	<code>add_train</code>	2.1 Maintain Train	Add Train form
2 Train and Schedule Management	FR-2.3	Admin defines the route: stations in sequence with times and distance	<code>add_train</code> (route loop)	2.2 Maintain Route	Add Train form
2 Train and Schedule Management	FR-2.5	Admin sets a per-km fare for each class	<code>add_train</code> (fare loop)	2.3 Maintain Fare	Add Train form
2 Train and Schedule Management	FR-2.6	Admin creates a schedule (run) of a train on a date	<code>add_schedule</code> , <code>new_schedule</code>	2.4 Create Schedule	Add Schedule form
3 Search and Availability	FR-3.1	Search trains by source, destination and date	<code>search_trains</code> , <code>find_schedules</code>	3.1 Search Trains	Search screen
3 Search and Availability	FR-3.4	Show seat availability per class, or waitlist position	<code>check_availability</code> , <code>print_availability</code> , <code>free_seats</code>	3.2 Check Availability	Availability screen
4 Booking	FR-4.2	Capture up to 6 passengers with age, gender, berth preference	<code>book_ticket</code> (passenger loop)	4.1 Capture Passengers	Booking form

Module (Session 5)	FR id (Session 3)	Requirement	Function in rrs.py	DFD process (Session 4)	Screen (Session 6)
4 Booking	FR-4.5	Allocate seats in order, or waitlist when none are free	<code>book_ticket</code> , <code>free_seats</code> , <code>seat_labels</code>	4.2 Allocate Seats	Booking form
4 Booking	FR-4.4	Compute fare as rate per km x distance x passengers	<code>book_ticket</code> (fare line)	4.3 Compute Fare	Booking form
4 Booking	FR-4.8	Generate a unique 10-digit PNR and store the booking	<code>new_pnr</code> , <code>book_ticket</code>	4.4 Generate PNR	Booking confirmation
4 Booking	FR-4.3	Refuse booking after the schedule has departed	<code>book_ticket</code> , <code>departure</code>	4.1 Capture Passengers	Booking form
5 Cancellation and Refund	FR-5.1	Cancel a booking by PNR	<code>cancel_ticket</code>	5.1 Cancel Booking	Cancel screen
5 Cancellation and Refund	FR-5.3	Compute refund by hours before departure	<code>cancel_ticket</code> (refund slabs)	5.2 Compute Refund	Cancel screen
5 Cancellation and Refund	FR-5.5	Promote the waitlist in order when seats are freed	<code>promote_waitlist</code>	5.3 Promote Waitlist	Cancel screen
4 Booking	FR-4.9	Look up a booking by PNR	<code>pnr_lookup</code> , <code>print_booking</code>	4.5 PNR Enquiry	PNR status screen

Module (Session 5)	FR id (Session 3)	Requirement	Function in rrs.py	DFD process (Session 4)	Screen (Session 6)
All	NFR-R3	Data survives program restart	<code>load_data</code> , <code>save_data</code>	D1 to D4 data stores	none
All	NFR-U2	Every input validated, no crash on bad input	<code>ask</code> , <code>ask_int</code> , <code>ask_date</code>	none	all forms

Program

■ Write in lab record

rrs.py

```

1  #!/usr/bin/env python3
2  """Railway Reservation System (RRS) - MCS-217 Session 13.
3
4  Console implementation of modules 2 to 5 of the RRS specified in
5  Sessions 1, 3, 4, 5 and 6:
6      2. Train and Schedule Management   (Administrator)
7      3. Search and Availability         (Passenger)
8      4. Booking                         (Passenger)
9      5. Cancellation and Refund        (Passenger)
10
11  Standard library only. All data lives in rrs_data.json next to this file.
12  """
13  import json
14  import os
15  import random
16  from datetime import date, datetime, timedelta
17
18  DATA_FILE = os.path.join(os.path.dirname(os.path.abspath(__file__)), "rrs_data.json")
19  CLASSES = ("SL", "3A", "2A", "1A")
20  BERTHS = ("LB", "MB", "UB", "SL", "SU")
21  CLERKAGE = 60          # flat clerkage in rupees (business rule, Session 3)
22  MAX_PASSENGERS = 6     # per PNR (business rule, Session 3)
23
24  # ----- persistence
25
26
27  def load_data():
28      """Read rrs_data.json; seed two trains and two schedules on first run."""
29      if os.path.exists(DATA_FILE):
30          with open(DATA_FILE) as fh:
31              return json.load(fh)
32      data = {"trains": {}, "schedules": {}, "bookings": {}}
33      data["trains"]["12951"] = {
34          "name": "Mumbai Rajdhani", "type": "Rajdhani",
35          "route": [
36              {"station_code": "BCT", "name": "Mumbai Central", "sequence": 1,
37               "arrival_time": "--", "departure_time": "17:00", "distance_km": 0},
38              {"station_code": "KOTA", "name": "Kota Jn", "sequence": 2,
39               "arrival_time": "01:35", "departure_time": "01:40", "distance_km": 869},
40              {"station_code": "NDLS", "name": "New Delhi", "sequence": 3,
41               "arrival_time": "08:35", "departure_time": "--", "distance_km": 1384},
42          ],
43          "coaches": {"3A": [2, 64], "2A": [1, 48], "1A": [1, 18]},
44          "fares": {"3A": 2.10, "2A": 2.90, "1A": 4.80},
45      }
46      data["trains"]["12137"] = {
47          "name": "Punjab Mail", "type": "Mail",
48          "route": [
49              {"station_code": "CSMT", "name": "Mumbai CSMT", "sequence": 1,
50               "arrival_time": "--", "departure_time": "19:35", "distance_km": 0},
51              {"station_code": "BPL", "name": "Bhopal Jn", "sequence": 2,
52               "arrival_time": "09:15", "departure_time": "09:25", "distance_km": 837},
53              {"station_code": "NDLS", "name": "New Delhi", "sequence": 3,
54               "arrival_time": "20:35", "departure_time": "--", "distance_km": 1541},
55          ],

```

```

56     "coaches": {"SL": [3, 72], "3A": [1, 64]},
57     "fares": {"SL": 0.60, "3A": 1.60},
58 }
59 in_10_days = (date.today() + timedelta(days=10)).isoformat()
60 for sid, tno in (("S001", "12951"), ("S002", "12137")):
61     data["schedules"][sid] = new_schedule(tno, in_10_days, data["trains"][tno])
62     save_data(data)
63     return data
64
65
66 def save_data(data):
67     """Write the whole data dictionary back to rrs_data.json."""
68     with open(DATA_FILE, "w") as fh:
69         json.dump(data, fh, indent=1)
70
71
72 def new_schedule(train_no, run_date, train):
73     """Build a Schedule record with empty seat maps and waitlists per class."""
74     return {"train_no": train_no, "run_date": run_date, "status": "SCHEDULED",
75             "allocated": {c: {} for c in train["coaches"]},
76             "waitlist": {c: [] for c in train["coaches"]}}
77
78 # ----- input helpers
79
80
81 def ask(prompt, check=lambda s: True, error="Invalid value, try again."):
82     """Prompt until check(value) is true; returns the stripped string."""
83     while True:
84         value = input(prompt).strip()
85         if value and check(value):
86             return value
87         print(error)
88
89
90 def ask_int(prompt, lo, hi):
91     """Prompt for an integer in the closed range lo..hi."""
92     return int(ask(prompt, lambda s: s.isdigit() and lo ≤ int(s) ≤ hi,
93                  "Enter a number from %d to %d." % (lo, hi)))
94
95
96 def ask_date(prompt):
97     """Prompt for a date in YYYY-MM-DD form."""
98     def ok(s):
99         try:
100             date.fromisoformat(s)
101
102             return True
103
104         except ValueError:
105
106             return False
107
108     return ask(prompt, ok, "Use the form YYYY-MM-DD.")
109
110
111

```

```

10
6 # ----- domain helpers
10
7
10
8
10
9 def seat_labels(train, cls):
11
10     """All seat labels of a class in allocation order, e.g. 3A1-17."""
11
1     coaches, per_coach = train["coaches"][cls]
11
2     return ["%s%d-%d" % (cls, c, s) for c in range(1, coaches + 1)
11
3             for s in range(1, per_coach + 1)]
11
4
11
5
11
6 def free_seats(data, sched, cls):
11
7     """Seat labels of this class not yet allocated on this schedule."""
11
8     train = data["trains"][sched["train_no"]]
11
9     return [s for s in seat_labels(train, cls) if s not in sched["allocated"][cls]]
12
10
12
1
12
2 def departure(data, sched):
12
3     """Datetime at which the schedule leaves its first station."""
12
4     first = data["trains"][sched["train_no"]]["route"][0]
12
5     return datetime.fromisoformat(sched["run_date"] + " " + first["departure_time"])
12
6
12
7
12
8 def stop_index(train, code):
12
9     """Index of a station code in the train route, or -1."""
13
10     for i, stop in enumerate(train["route"]):
13
11         if stop["station_code"] == code:
13
12             return i
13
13
3     return -1

```

```
13
14
13
5
13
6 def new_pnr(data):
13
7     """Unique 10-digit PNR."""
13
8     while True:
13
9         pnr = str(random.randint(10 ** 9, 10 ** 10 - 1))
14
10         if pnr not in data["bookings"]:
14
11             return pnr
14
12
14
2
14
3
14
4 def find_schedules(data, src, dst, run_date):
14
5     """Schedules on run_date whose route has src before dst and not yet departed."""
14
6     found = []
14
7     for sid, sched in sorted(data["schedules"].items()):
14
8         train = data["trains"][sched["train_no"]]
14
9         i, j = stop_index(train, src), stop_index(train, dst)
15
10         if sched["run_date"] == run_date and 0 ≤ i < j and departure(data, sched) > datetime.now():
15
11             found.append((sid, sched, train, i, j))
15
12     return found
15
13
15
4
15
5 def print_availability(data, sched):
15
6     """One line per class: free seats or waitlist length."""
15
7     for cls in sched["allocated"]:
15
8         free, wl = len(free_seats(data, sched, cls)), len(sched["waitlist"][cls])
15
9         print("    %-3s %s" % (cls, "AVAILABLE %d" % free if free else "WL %d" % (wl + 1)))
16
10
16
1
```

```

16
2 def promote_waitlist(data, sched, cls):
16
3     """Move waitlisted bookings of this class to CONFIRMED while seats allow."""
16
4     wl = sched["waitlist"][cls]
16
5     while wl:
16
6         booking = data["bookings"][wl[0]]
16
7         free = free_seats(data, sched, cls)
16
8         if len(free) < len(booking["passengers"]):
16
9             return
17
10         for p, seat in zip(booking["passengers"], free):
17
11             p["seat"] = seat
17
12             sched["allocated"][cls][seat] = wl[0]
17
13             booking["status"] = "CONFIRMED"
17
14             print("Waitlisted PNR %s promoted to CONFIRMED." % wl.pop(0))
17
15
16 # ----- admin actions
17
17
17
17
18
9 def add_train(data):
18
10     """Admin: add a train with its route stations, coaches and per-km fares."""
18
11     train_no = ask("Train number (5 digits): ", lambda s: s.isdigit() and len(s) == 5)
18
12     if train_no in data["trains"]:
18
13         print("Train %s already exists." % train_no)
18
14         return
18
15     train = {"name": ask("Train name: "), "type": ask("Type (Rajdhani/Mail/Express): "),
18
16             "route": [], "coaches": {}, "fares": {}}
18
17     stops = ask_int("Number of stations on the route (2-20): ", 2, 20)
18
19     for seq in range(1, stops + 1):
18
20         print("Station %d of %d" % (seq, stops))

```

```

19
0     train["route"].append({
19
1         "station_code": ask("  code: ").upper(), "name": ask("  name: "), "sequence": seq,
19
2         "arrival_time": "--" if seq == 1 else ask("  arrival HH:MM: "),
19
3         "departure_time": "--" if seq == stops else ask("  departure HH:MM: "),
19
4         "distance_km": 0 if seq == 1 else ask_int("  distance from origin (km): ", 1, 5000)})
19
5 for cls in CLASSES:
19
6     n = ask_int("Coaches of class %s (0-10): " % cls, 0, 10)
19
7     if n:
19
8         train["coaches"][cls] = [n, ask_int("  seats per %s coach (1-80): " % cls, 1, 80)]
19
9         train["fares"][cls] = float(ask("  fare per km for %s (rupees): " % cls,
20
0                                     lambda s: s.replace(".", "", 1).isdigit()))
20
1     data["trains"][train_no] = train
20
2     save_data(data)
20
3     print("Train %s %s added with %d stations." % (train_no, train["name"], stops))
20
4
20
5
20
6 def add_schedule(data):
20
7     """Admin: create a run of an existing train on a given date."""
20
8     list_trains(data)
20
9     train_no = ask("Train number: ", lambda s: s in data["trains"], "No such train.")
21
0     run_date = ask_date("Run date (YYYY-MM-DD): ")
21
1     if any(s["train_no"] == train_no and s["run_date"] == run_date for s in data["schedules"].values):
21
2         print("That train already runs on %s." % run_date)
21
3         return
21
4     sid = "%03d" % (len(data["schedules"]) + 1)
21
5     data["schedules"][sid] = new_schedule(train_no, run_date, data["trains"][train_no])
21
6     save_data(data)
21
7     print("Schedule %s created: %s on %s." % (sid, train_no, run_date))

```

```

21
8
21
9
22
0 def list_trains(data):
22
1     """Admin: print every train with its route and classes."""
22
2     for train_no, t in sorted(data["trains"].items()):
22
3         stations = " > ".join(s["station_code"] for s in t["route"])
22
4         print("%s %-16s %s classes: %s" % (train_no, t["name"], stations, ", ".join(t["coaches"])))
22
5
22
6 # ----- passenger actions
22
7
22
8
22
9 def search_trains(data):
23
0     """Passenger: list trains between two stations on a date with availability."""
23
1     src, dst = ask("From station code: ").upper(), ask("To station code: ").upper()
23
2     run_date = ask_date("Journey date (YYYY-MM-DD): ")
23
3     found = find_schedules(data, src, dst, run_date)
23
4     if not found:
23
5         print("No trains found for %s to %s on %s." % (src, dst, run_date))
23
6         return
23
7     for sid, sched, train, i, j in found:
23
8         km = train["route"][j]["distance_km"] - train["route"][i]["distance_km"]
23
9         print("%s %s %s dep %s arr %s %d km" % (sid, sched["train_no"], train["name"],
24
0             train["route"][i]["departure_time"], train["route"][j]["arrival_time"], km))
24
1         print_availability(data, sched)
24
2
24
3
24
4 def check_availability(data):
24
5     """Passenger: seat availability per class for one schedule."""

```

```

24
6     sid = ask("Schedule id: ", lambda s: s in data["schedules"], "No such schedule.")
24
7     sched = data["schedules"][sid]
24
8     print("%s %s on %s" % (sched["train_no"], data["trains"][sched["train_no"]]["name"], sched["run
24
9     print_availability(data, sched)
25
0
25
1
25
2 def book_ticket(data):
25
3     """Passenger: book up to six passengers, allocate seats, generate PNR."""
25
4     sid = ask("Schedule id: ", lambda s: s in data["schedules"], "No such schedule.")
25
5     sched = data["schedules"][sid]
25
6     train = data["trains"][sched["train_no"]]
25
7     if departure(data, sched) ≤ datetime.now():
25
8         print("This train has already departed. Booking refused.")
25
9         return
26
0     src = ask("From station code: ", lambda s: stop_index(train, s.upper()) ≥ 0, "Not on route.")
26
1     dst = ask("To station code: ",
26
2         lambda s: stop_index(train, s.upper()) > stop_index(train, src), "Must be after %s."
26
3     cls = ask("Class (%s): " % "/".join(train["coaches"]), lambda s: s.upper() in train["coaches"])
26
4     n = ask_int("Number of passengers (1-%d): " % MAX_PASSENGERS, 1, MAX_PASSENGERS)
26
5     passengers = []
26
6     for k in range(1, n + 1):
26
7         passengers.append({"name": ask("Passenger %d name: " % k),
26
8                             "age": ask_int("Passenger %d age: " % k, 1, 120),
26
9                             "gender": ask("Passenger %d gender (M/F/O): " % k, lambda s: s.upper() :
27
0                             "berth_preference": ask("Passenger %d berth preference (LB/MB/UB/SL/SU)
27
1                             lambda s: s.upper() in BERTHS).upper(), "seat":
27
2     km = train["route"][stop_index(train, dst)]["distance_km"] - train["route"][stop_index(train, s
27
3     fare = round(train["fares"][cls] * km * n, 2)

```

```

27
4    free = free_seats(data, sched, cls)
27
5    status = "CONFIRMED" if len(free) ≥ n else "WAITLISTED"
27
6    print("Fare: %d km x Rs %.2f/km x %d passengers = Rs %.2f [%s]" % (km, train["fares"][cls], n,
27
7    if ask("Confirm booking (Y/N): ", lambda s: s.upper() in "YN").upper() ≠ "Y":
27
8        print("Booking abandoned.")
27
9        return
28
0    pnr = new_pnr(data)
28
1    if status == "CONFIRMED":
28
2        for p, seat in zip(passengers, free):
28
3            p["seat"] = seat
28
4            sched["allocated"][cls][seat] = pnr
28
5    else:
28
6        sched["waitlist"][cls].append(pnr)
28
7    data["bookings"][pnr] = {"pnr": pnr, "schedule_id": sid, "from_station": src, "to_station": dst,
28
8                                "class": cls, "booking_time": datetime.now().isoformat(timespec="seconds"),
28
9                                "status": status, "total_fare": fare, "passengers": passengers}
29
0    save_data(data)
29
1    print("Booked. PNR %s status %s" % (pnr, status))
29
2    print_booking(data, pnr)
29
3
29
4
29
5 def cancel_ticket(data):
29
6     """Passenger: cancel by PNR, compute refund, promote waitlist."""
29
7     pnr = ask("PNR: ", lambda s: s in data["bookings"], "No such PNR.")
29
8     booking = data["bookings"][pnr]
29
9     if booking["status"] == "CANCELLED":
30
0         print("PNR %s is already cancelled." % pnr)
30
1         return

```

```
30
2    sched = data["schedules"][booking["schedule_id"]]
30
3    hours = (departure(data, sched) - datetime.now()).total_seconds() / 3600
30
4    if booking["status"] == "WAITLISTED":
30
5        refund, rule = booking["total_fare"] - CLERKAGE, "waitlisted, fare minus clerkage"
30
6        sched["waitlist"][booking["class"]].remove(pnr)
30
7    elif hours > 48:
30
8        refund, rule = booking["total_fare"] - CLERKAGE, "more than 48 h before departure"
30
9    elif hours ≥ 12:
31
10        refund, rule = booking["total_fare"] * 0.5, "between 48 h and 12 h before departure"
31
11    else:
31
12        refund, rule = 0.0, "less than 12 h before departure"
31
13    refund = round(max(refund, 0.0), 2)
31
14    print("Refund: Rs %.2f (%s)" % (refund, rule))
31
15    if ask("Confirm cancellation (Y/N): ", lambda s: s.upper() in "YN").upper() ≠ "Y":
31
16        print("Cancellation abandoned.")
31
17        return
31
18    if booking["status"] == "CONFIRMED":
31
19        for p in booking["passengers"]:
32
20            del sched["allocated"][booking["class"]][p["seat"]]
32
21            p["seat"] = None
32
22    booking["status"] = "CANCELLED"
32
23    booking["refund"] = {"amount": refund, "reason": rule,
32
24                        "processed_at": datetime.now().isoformat(timespec="seconds")}
32
25    promote_waitlist(data, sched, booking["class"])
32
26    save_data(data)
32
27    print("PNR %s cancelled." % pnr)
32
28
32
29
```

```

33
0 def pnr_lookup(data):
33
1     """Passenger: show a booking by PNR."""
33
2     print_booking(data, ask("PNR: ", lambda s: s in data["bookings"], "No such PNR.))
33
3
33
4
33
5 def print_booking(data, pnr):
33
6     """Print one booking with its passengers and seats."""
33
7     b = data["bookings"][pnr]
33
8     sched = data["schedules"][b["schedule_id"]]
33
9     print("PNR %s %s %s %s %s > %s class %s fare Rs %.2f %s" % (
34
10         pnr, sched["train_no"], data["trains"][sched["train_no"]]["name"], sched["run_date"],
34
11         b["from_station"], b["to_station"], b["class"], b["total_fare"], b["status"]))
34
12     for p in b["passengers"]:
34
13         print(" %-12s %3d %s pref %-2s seat %s" % (p["name"], p["age"], p["gender"],
34
14                                                     p["berth_preference"], p["seat"] or "--"))
34
15
34
16 # ----- menus
34
17
34
18
34
19 def run_menu(title, actions, data):
35
20     """Print a numbered menu and dispatch until the user chooses 0."""
35
21     while True:
35
22         print("\n= %s =" % title)
35
23         for k, (label, _) in enumerate(actions, 1):
35
24             print(" %d. %s" % (k, label))
35
25         print(" 0. Back")
35
26         choice = ask_int("Choice: ", 0, len(actions))
35
27         if choice == 0:

```

```

35
8         return
35
9         actions[choice - 1][1](data)
36
0
36
1
36
2 def main():
36
3     """Entry point: role selection."""
36
4     data = load_data()
36
5     admin = [("Add train", add_train), ("Add schedule", add_schedule), ("List trains", list_trains)]
36
6     passenger = [("Search trains", search_trains), ("Check availability", check_availability),
36
7                  ("Book ticket", book_ticket), ("Cancel ticket", cancel_ticket), ("PNR lookup", pnr_lookup)]
36
8     roles = [("Administrator", lambda d: run_menu("Administrator", admin, d)),
36
9              ("Passenger", lambda d: run_menu("Passenger", passenger, d))]
37
0     print("Railway Reservation System (Session 13 build)")
37
1     try:
37
2         run_menu("Main menu", roles, data)
37
3     except EOFError:
37
4         pass
37
5     print("Bye.")
37
6
37
7
37
8 if __name__ == "__main__":
37
9     main()

```

How to Run

Python 3.8 or later, no packages to install. The first run creates `rrs_data.json` beside the script with two trains (12951 Mumbai Rajdhani and 12137 Punjab Mail) and one schedule each, dated 10 days from today, so a demonstration works immediately. Delete the JSON file to start again.

```
1 python3 rrs.py
```

Menu tree: Main menu offers Administrator and Passenger. Administrator: Add train, Add schedule, List trains. Passenger: Search trains, Check availability, Book ticket, Cancel ticket, PNR lookup. Enter 0 to go back.

Sample Output

Run on 2026-09-26 with Python 3.13 on a fresh data file. Repeated menu listings are removed; each `Choice:` line shows the option typed. The run adds a train with one 1A seat so the waitlist can be shown in a few steps.

```
1 Railway Reservation System (Session 13 build)
2 Choice: 1
3 Choice: 1
4 Train number (5 digits): 22222
5 Train name: Demo Express
6 Type (Rajdhani/Mail/Express): Express
7 Number of stations on the route (2-20): 2
8 Station 1 of 2
9   code: AGC
10  name: Agra Cantt
11  departure HH:MM: 06:10
12 Station 2 of 2
13   code: NDLS
14   name: New Delhi
15   arrival HH:MM: 09:00
16   distance from origin (km): 195
17 Coaches of class SL (0-10): 0
18 Coaches of class 3A (0-10): 0
19 Coaches of class 2A (0-10): 0
20 Coaches of class 1A (0-10): 1
21   seats per 1A coach (1-80): 1
22   fare per km for 1A (rupees): 2.50
23 Train 22222 Demo Express added with 2 stations.
24 Choice: 2
25 12137 Punjab Mail      CSMT > BPL > NDLS  classes: SL, 3A
26 12951 Mumbai Rajdhani BCT > KOTA > NDLS  classes: 3A, 2A, 1A
27 22222 Demo Express     AGC > NDLS  classes: 1A
28 Train number: 22222
29 Run date (YYYY-MM-DD): 2026-10-08
30 Schedule S003 created: 22222 on 2026-10-08.
31 Choice: 0
32 Choice: 2
33 Choice: 1
34 From station code: BCT
35 To station code: NDLS
36 Journey date (YYYY-MM-DD): 2026-10-06
37 S001 12951 Mumbai Rajdhani dep 17:00 arr 08:35 1384 km
38   3A AVAILABLE 128
39   2A AVAILABLE 48
40   1A AVAILABLE 18
41 Choice: 3
42 Schedule id: S001
43 From station code: BCT
44 To station code: NDLS
45 Class (3A/2A/1A): 2A
46 Number of passengers (1-6): 2
47 Passenger 1 name: Asha Verma
48 Passenger 1 age: 34
49 Passenger 1 gender (M/F/O): F
50 Passenger 1 berth preference (LB/MB/UB/SL/SU): LB
51 Passenger 2 name: Rohan Verma
52 Passenger 2 age: 8
53 Passenger 2 gender (M/F/O): M
54 Passenger 2 berth preference (LB/MB/UB/SL/SU): UB
55 Fare: 1384 km x Rs 2.90/km x 2 passengers = Rs 8027.20 [CONFIRMED]
```

```
56 Confirm booking (Y/N): Y
57 Booked. PNR 2030868043 status CONFIRMED
58 PNR 2030868043 12951 Mumbai Rajdhani 2026-10-06 BCT > NDLS class 2A fare Rs 8027.20 CONFIRMED
59 Asha Verma 34 F pref LB seat 2A1-1
60 Rohan Verma 8 M pref UB seat 2A1-2
61 Choice: 3
62 Schedule id: S003
63 From station code: AGC
64 To station code: NDLS
65 Class (1A): 1A
66 Number of passengers (1-6): 1
67 Passenger 1 name: Kiran Rao
68 Passenger 1 age: 41
69 Passenger 1 gender (M/F/O): M
70 Passenger 1 berth preference (LB/MB/UB/SL/SU): LB
71 Fare: 195 km x Rs 2.50/km x 1 passengers = Rs 487.50 [CONFIRMED]
72 Confirm booking (Y/N): Y
73 Booked. PNR 9478409106 status CONFIRMED
74 PNR 9478409106 22222 Demo Express 2026-10-08 AGC > NDLS class 1A fare Rs 487.50 CONFIRMED
75 Kiran Rao 41 M pref LB seat 1A1-1
76 Choice: 3
77 Schedule id: S003
78 From station code: AGC
79 To station code: NDLS
80 Class (1A): 1A
81 Number of passengers (1-6): 1
82 Passenger 1 name: Meera Nair
83 Passenger 1 age: 29
84 Passenger 1 gender (M/F/O): F
85 Passenger 1 berth preference (LB/MB/UB/SL/SU): LB
86 Fare: 195 km x Rs 2.50/km x 1 passengers = Rs 487.50 [WAITLISTED]
87 Confirm booking (Y/N): Y
88 Booked. PNR 3409290220 status WAITLISTED
89 PNR 3409290220 22222 Demo Express 2026-10-08 AGC > NDLS class 1A fare Rs 487.50 WAITLISTED
90 Meera Nair 29 F pref LB seat --
91 Choice: 5
92 PNR: 9478409106
93 PNR 9478409106 22222 Demo Express 2026-10-08 AGC > NDLS class 1A fare Rs 487.50 CONFIRMED
94 Kiran Rao 41 M pref LB seat 1A1-1
95 Choice: 4
96 PNR: 9478409106
97 Refund: Rs 427.50 (more than 48 h before departure)
98 Confirm cancellation (Y/N): Y
99 Waitlisted PNR 3409290220 promoted to CONFIRMED.
10
0 PNR 9478409106 cancelled.
10
1 Choice: 5
10
2 PNR: 3409290220
10
3 PNR 3409290220 22222 Demo Express 2026-10-08 AGC > NDLS class 1A fare Rs 487.50 CONFIRMED
10
4 Meera Nair 29 F pref LB seat 1A1-1
10
5 Choice: 4
```

```

10
6  PNR: 9478409106
10
7  PNR 9478409106 is already cancelled.
10
8  Choice: 0
10
9  Choice: 0
11
0  Bye.

```

Check by hand: fare $1384 \times 2.90 \times 2 = 8027.20$; refund $487.50 - 60 = 427.50$ because departure is 12 days away. Bad input is rejected and the prompt repeats, for example `Schedule id: S999` prints `No such schedule.` and asks again.

Known Limitations

■ Write in lab record

SRS item not implemented	Why	Effect on the demonstration
Module 1 User Management (FR-1.1 to FR-1.6)	Role is chosen from a menu; no registration, login or password hash	Anyone can act as Administrator. Acceptable in a single-user lab build
Module 6 Payment	Needs an external gateway; the SRS marks it as an external actor	Fare is computed and stored in <code>total_fare</code> ; no Payment record with mode or txn_ref
Module 7 Notification	Needs SMS and email services	The console line printed after booking stands in for the confirmation message
Module 8 Reports	Admin-only reads over the same data; no new rule to prove	Occupancy or revenue can be read from <code>rrs_data.json</code> by hand
Segment-wise availability	A seat is held for the whole run, even for a BCT to KOTA booking	Availability can be under-reported on long routes
Berth preference	Stored on the Passenger record but not used in allocation	Seat labels are allocated in order
Schedule status changes and delay notices	Out of scope for one user role per module	<code>status</code> is always SCHEDULED
Concurrency	Single process, file rewritten after each action	Two clerks running two copies could double-book

Validation Checklist

■ Write in lab record

Each row is a business rule from the Session 3 SRS with the test performed on the run above.

Rule	Test	Expected	Observed	Pass
PNR is a 10-digit unique number	Book three tickets	Three distinct 10-digit values	2030868043, 9478409106, 3409290220	Yes
Max 6 passengers per PNR	Enter 7 at the passenger count prompt	Prompt repeats with range message	Enter a number from 1 to 6.	Yes
Fare = rate per km x distance x passengers	2A BCT to NDLS, 2 passengers	$2.90 \times 1384 \times 2 = 8027.20$	Rs 8027.20	Yes
Seats allocated in order	First 2A booking on S001	2A1-1 and 2A1-2	2A1-1, 2A1-2	Yes
Waitlist when no seats	Second booking on a 1-seat class	Status WAITLISTED, seat shown as --	WAITLISTED	Yes
Refund 100% minus clerkage over 48 h	Cancel 12 days before departure	$487.50 - 60 = 427.50$	Rs 427.50	Yes
Refund 50% between 48 h and 12 h	Demo Express scheduled tomorrow, departs 06:10; cancelled at 09:09 today (21 h before)	$50\% \text{ of } 487.50 = 243.75$	Rs 243.75	Yes
No refund under 12 h	12951 scheduled today, departs 17:00; cancelled at 09:09 (8 h before), fare 4013.60	Rs 0.00	Rs 0.00	Yes
Waitlist promoted in order	Cancel the confirmed 1A ticket	First waitlisted PNR gets the freed seat	PNR 3409290220 promoted, seat 1A1-1	Yes
No booking after departure	Admin adds a 12951 run dated yesterday, passenger tries to book it	Refused	This train has already departed. Booking refused.	Yes
Cancelled PNR cannot be cancelled again	Cancel 9478409106 twice	Second attempt refused	is already cancelled.	Yes
Data survives restart	Book in one run, look up in the next	Booking found	Found with same seats	Yes

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** Where is the 48-hour refund rule in the code? **A:** In `cancel_ticket`; hours before departure come from `departure(data, sched) minus datetime.now()`.
- **Q:** How is PNR uniqueness guaranteed? **A:** `new_pnr` draws a random 10-digit number and retries while it already exists in `bookings`.
- **Q:** Why is the schedule the unit of booking and not the train? **A:** A train runs on many dates; seats belong to one run. That is the Schedule entity from the Session 4 ERD.
- **Q:** What happens to a waitlisted booking when a confirmed one is cancelled? **A:** `promote_waitlist` takes the first PNR in the class waitlist and allocates freed seats if enough are free for all its passengers.
- **Q:** Why is there no login? **A:** Module 1 was cut for the lab build; role is picked from the menu. It is recorded in Known Limitations, not hidden.
- **Q:** How does the program stop a booking after departure? **A:** `book_ticket` compares `departure()` with the current time and refuses; `find_schedules` also hides departed runs from search.
- **Q:** What is stored in `rrs_data.json` and how does it map to the ERD? **A:** Three maps: trains (with nested Route, Coach, Fare), schedules (with allocated seats and waitlists), bookings (with nested Passenger and Refund).
- **Q:** Why compute fare from a per-km rate and not a fixed fare per pair of stations? **A:** The Session 4 data dictionary defines Fare as `(train_no, class, rate_per_km)`, so the code follows the data model.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Building a different system from the one specified. The examiner checks the code against your own SRS; new modules that are not in it earn nothing.
- No traceability. A program without the FR-to-function table cannot be defended in a viva and cannot be changed in Session 14.
- Hard-coding the demonstration data in the program instead of seeding a data file, so the second run cannot show persistence.
- Letting bad input crash the program. Every `input()` goes through `ask`, `ask_int` or `ask_date`.
- Hiding the gaps. Write the Known Limitations table; an honest gap is a design decision, an undiscovered one is a defect.
- Refund computed from booking time instead of departure time. The rule is hours before departure.

Session Summary

■ Write in lab record

- Traceability matrix from Session 3 FR ids and Session 5 modules to function names
- Full listing of `rrs.py` with the module docstring
- How to run, and the sample session transcript showing add schedule, search, book, waitlist, lookup and cancel with promotion
- Known Limitations table stating what from the SRS is not built and why
- Validation checklist with observed values for every business rule

[Edit this page](#)

< Previous
Session 12

Next >
Session 14