Group B OWIS Session 7: Python

Here is a 60‑minute, space‑themed Session 7 lesson plan to introduce Python basics while challenging prior coders with a complex task, ensuring both groups are engaged and learning meaningfully.

Quick outcome

By the end, students build small Python programs using variables, input, print, and operators around a Mars mission scenario, while experienced coders tackle an extended "Mars Supply Planner" challenge with functions and simple validation.

Objectives

- Explain why Python is widely used and how it will power future club projects (data handling, simulations, simple back‑ends).
- Use variables, input, print, and basic operators to compute simple space‑mission calculations (fuel, travel time, crew supplies).
- Differentiate tasks: beginners complete guided exercises; experienced students complete an open‑ended mini‑project with extra features.

Materials

- Laptops with Python (or Replit) and projector; starter templates for both tracks; printed challenge cards for the "Mars Supply Planner".
- Role stickers or labels (Beginner/Experienced) only to route tasks, not to spotlight ability; timer and whiteboard for prompts.

0–5 min: Space hook

- Prompt: "Mission Control needs a quick Python script to estimate water and food for a 30‑day Mars surface stay—can a few lines of code help a crew survive?" Briefly show a one‑screen pseudocode snippet to anchor purpose.

5–10 min: Why Python

- Python strengths: easy syntax, huge libraries, used in robotics, data science, automation, and space mission simulations; perfect bridge from app building to back‑end logic and data processing in future sessions.

- Today's focus: variables, input, print, operators—enough to compute real mission numbers and text-based interactions.

10–15 min: Diagnostic split and brief

- Quick self-sort: students who have coded in Python take the "Mars Supply Planner – Pro" card; others take the "Beginner Flight Check" card.
- Explain parallel goals: both groups work on the same mission theme with different depth; peer mentors can help but should not take over typing.

15–30 min: Beginners—guided build

- Mini-lesson with live code:
    - Variables and print:
    - python

```python
mission = "Artemis Mars Surface Ops"
days = 30
print("Mission:", mission)
print("Surface days:", days)
```

    -
    - Input and simple math:
    - python

```python
crew = int(input("Enter crew size: "))
days = int(input("Enter surface days: "))
water_per_day = 3  # liters per astronaut
total_water = crew * days * water_per_day
print("Total water needed (L):", total_water)
```

    -
    - Operators and formatted output:
    - python

```python
food_per_day = 0.8  # kg per astronaut
total_food = crew * days * food_per_day
print(f"Food mass (kg): {total_food}")
print("Average per astronaut (kg):", total_food / crew)
```

    -
- Practice prompts on slides:
    - Ask for habitat battery capacity and daily usage; compute days of backup power.
    - Ask for rover speed and distance to site; compute travel time in hours.
- Pair check: swap laptops and run each other's script with different inputs to confirm correct calculations.

15–30 min: Experienced—Mars Supply Planner (independent)

- Brief:
    - Build a text-mode planner that asks for crew, days, and safety margin (percent), computes water and food needs, and prints a summary report.
    - Stretch goals:
        - Add an oxygen canister count given per-canister hours.
        - Add a function per resource, e.g., `def water_needed(crew, days, liters_per_day, margin):`.
        - Input validation: reject negative or zero values; round up canisters.
- Suggested scaffold:
- `python`

```
def with_margin(value, margin_percent):
    return value * (1 + margin_percent / 100)

crew = int(input("Crew size: "))
days = int(input("Surface days: "))
margin = float(input("Safety margin %: "))

water_per_day = 3
food_per_day = 0.8

water = with_margin(crew * days * water_per_day, margin)
food = with_margin(crew * days * food_per_day, margin)

print(f"Water (L): {round(water, 1)}")
print(f"Food (kg): {round(food, 1)}")
```

-

30–40 min: Interleaved coaching

- Rotate support: first pass with beginners checking types and operator use; second pass with experienced students reviewing function design and input checks.
- Micro-challenges on the board:
    - Beginners: add “crew roles” input and echo back a mission greeting using concatenation.
    - Experienced: add “dust storm day factor” that increases power usage; recompute backup days.

40–50 min: Partner tests and edge cases

- Mixed pairs run each other’s programs:
    - Try crew = 1 and crew = 0; ensure validation or sensible output.
    - Change days from 30 to 90; observe impact and discuss margins.
    - Non-numeric input: how does the program behave? Experienced coders note the exception and propose a simple guard or message.

50–55 min: Lightning share

- 3–4 quick demos:
    - Clean variable names and readable prints
    - A solution that adds a function
    - A solution that handles invalid inputs gracefully.
- Ask the audience to call out one improvement for clarity in each demo (e.g., units in outputs).

55–60 min: Reflection and next steps

- Exit ticket options:
    - “One place Python helped make a mission decision today”
    - “One operator, one input, and one variable used correctly—show a code line”.
- Bridge forward: next sessions can use Python to simulate comms delays, compute launch windows, or parse chat logs from the Private Chat App to analyze crew protocols.

Differentiation and supports

- Beginners:

    - Provide a starter file with TODO comments and example prints; emphasize `int()` vs `float()` for input.
    - Require at least two inputs and three computed outputs tied to mission needs.
- Experienced:
    - Require at least two functions, validation, and a final formatted report block; optional `math.ceil` for canisters.
    - Bonus: parameterize per-astronaut consumption via inputs and compare scenarios (crew=4 vs crew=6).

Assessment rubric (quick)

- Proficient: Correct use of variables, input, print, and basic operators to produce mission-relevant outputs.
- Strong: Clear prompts, units in outputs, and at least one mini-refactor (function or formatted string).
- Advanced: Validation for inputs, modular functions, and a concise mission summary report reflecting margins and edge cases.

Classroom management and pacing

- Timer checkpoints on projector: first working input/print by 20 minutes; beginner script complete by 35; planner MVP by 40; testing by 50.
- Seat experienced coders near beginners to enable quick peer support without overshadowing; circulate with a “just in time” hint list.

Closure line

“Today Python turned mission questions into numbers—food, water, time, and safety margins—exactly how crews justify Go/No-Go calls on Mars operations”