PYTHON FOUNDATIONS / PROJECT BRIEF

Print the Repair Desk Sign

THE ASSIGNMENT

A community repair cafe is opening a desk for broken household items. Its first program makes a sign from a desk name, a room label, and an opening hour. A string is a piece of text enclosed in quotes. A variable lets later lines refer to a value by name. Here you will clean incoming text before formatting it with an f-string. The expressions inside braces supply the values for the printed sign. Start with fixed inputs, then change one value at a time to see exactly which output line it affects.

ACCEPTANCE TARGET

The heading is Bicycle Repairs Desk and the underline has 20 equals signs.

START HERE

Install Python 3.12 with its standard library. Extract the study bundle to a folder you can edit. In a terminal, open that folder and run python --version (or py -3.12 --version on Windows). Use a plain-text code editor and UTF-8.

```
python projects/day_001.py
# Windows alternative: py -3.12 projects/day_001.py
```

Each day spans five pages: frame the task, build it, inspect the two solution sections, and test an extension. The PDF outline links to every day. The code bundle includes a searchable project index.

JUMP TO A TRACK

Day 001 Start writing Python
Day 013 Automate practical work
Day 025 Build with Flask
Day 037 Ship Storyboard Studio

PYTHON FOUNDATIONS / GUIDED BUILD

Print the Repair Desk Sign

Create a working copy of day_001.py. Use the project contract and implement each step before comparing the reference.

STEP 01

Create sign.py in an empty lab folder. Store the desk name and room as strings and the hour as an integer.

STEP 02

Call strip on the name to remove edge spaces. Split the room text into words and join them with one space.

STEP 03

Use :02d inside an f-string to show the hour with at least two digits. Append :00 outside the braces.

STEP 04

Build a heading and an underline whose length matches the heading. Print the room and opening time below them.

STEP 05

Run python sign.py. Edit the room and confirm that only the room line changes.

FIRST EXPERIMENT

Run the smallest valid example. Record the exact result you observed.

PYTHON FOUNDATIONS / REFERENCE SOLUTION

Print the Repair Desk Sign

day_001.py | Part 1 of 2

```
raw_desk = "  bicycle repairs  "
raw_room = "  North    Hall "
opening_hour = 9

desk = raw_desk.strip().title()
room = " ".join(raw_room.split())
heading = f"{desk} Desk"
opening = f"{opening_hour:02d}:00"

```

READ THE DESIGN

Follow the assignments in order. Write down a value after each transformation and compare it to the acceptance target.

PYTHON FOUNDATIONS / RUN AND INSPECT

Print the Repair Desk Sign

day_001.py | Part 2 of 2

```
10  assert heading == "Bicycle Repairs Desk"
11  assert room == "North Hall"
12  assert opening == "09:00"
13
14  print(heading)
15  print("=" * len(heading))
16  print(f"Room: {room}")
17  print(f"Opens: {opening}")
```

RUN FROM THE EXTRACTED BUNDLE

```
python projects/day_001.py
```

OBSERVED CONSOLE OUTPUT

```
Bicycle Repairs Desk
====================
Room: North Hall
Opens: 09:00
```

PYTHON FOUNDATIONS / CHECK AND EXTEND

Print the Repair Desk Sign

VERIFY THE BEHAVIOR

01 The heading is Bicycle Repairs Desk and the underline has 20 equals signs.

02 The room prints as North Hall with one internal space.

03 The opening line reads Opens: 09:00. Set opening_hour to 14 and adjust the check to expect 14:00.

DEBUG A LIKELY FAILURE

If the padded hour raises ValueError, inspect type(opening_hour). The d format requires an integer, so remove quotes around the hour.

TAKE IT FURTHER

Add a closing hour. Print both times on one line and verify that 9 and 16 display as 09:00 and 16:00.

LAB NOTES

What did the checks prove? What remains untested?

PYTHON FOUNDATIONS / PROJECT BRIEF

Budget the Repair Slots

THE ASSIGNMENT

An organizer has a three-hour room booking, but setup and cleanup use part of it. Compute how many full appointments fit in the remaining minutes. Floor division with // keeps only whole slots for nonnegative minutes, while % yields the unused minutes. divmod returns both values in one call. Recombine the quotient and remainder to check your arithmetic. The result should account for every minute and keep a short leftover interval out of the appointment count.

ACCEPTANCE TARGET

Usable time is 155 minutes.

PLAN THE BOUNDARY

Store the room booking, setup, cleanup, and appointment length in integer minutes.

BEFORE YOU CODE

Describe the input and result you will use for your first run.

Predict one case the program should reject or handle separately.

PYTHON FOUNDATIONS / GUIDED BUILD

Budget the Repair Slots

Create a working copy of day_002.py. Use the project contract and implement each step before comparing the reference.

STEP 01

Store the room booking, setup, cleanup, and appointment length in integer minutes.

STEP 02

Subtract both overhead intervals from the booking to get usable_minutes.

STEP 03

Pass usable_minutes and slot_minutes to divmod. Assign the two results to slots and spare_minutes.

STEP 04

Rebuild the usable minutes from those two results. Add an assertion for the original booking total.

STEP 05

Format a short schedule report showing appointments and spare time separately.

FIRST EXPERIMENT

Run the smallest valid example. Record the exact result you observed.

PYTHON FOUNDATIONS / REFERENCE SOLUTION

Budget the Repair Slots

day_002.py | Part 1 of 2

```python
room_minutes = 180
setup_minutes = 15
cleanup_minutes = 10
slot_minutes = 20

usable_minutes = room_minutes - setup_minutes - cleanup_minutes
slots, spare_minutes = divmod(usable_minutes, slot_minutes)
scheduled_minutes = slots * slot_minutes + spare_minutes
total_minutes = setup_minutes + scheduled_minutes + cleanup_minutes
```

READ THE DESIGN

Follow the assignments in order. Write down a value after each transformation and compare it to the acceptance target.

PYTHON FOUNDATIONS / RUN AND INSPECT

Budget the Repair Slots

day_002.py | Part 2 of 2

```
10
11  assert slots == 7
12  assert spare_minutes == 15
13  assert total_minutes == room_minutes
14
15  print(f"Usable time: {usable_minutes} minutes")
16  print(f"Full appointments: {slots}")
17  print(f"Unscheduled time: {spare_minutes} minutes")
```

RUN FROM THE EXTRACTED BUNDLE

```
python projects/day_002.py
```

OBSERVED CONSOLE OUTPUT

```
Usable time: 155 minutes
Full appointments: 7
Unscheduled time: 15 minutes
```

PYTHON FOUNDATIONS / CHECK AND EXTEND

Budget the Repair Slots

VERIFY THE BEHAVIOR

01 Usable time is 155 minutes.

02 The booking fits seven appointments with 15 minutes to spare.

03 Set slot_minutes to 31 and update the checks. The same booking fits five appointments with zero spare minutes.

DEBUG A LIKELY FAILURE

If slots prints as 7.75, check the operator. The report needs the integer quotient from divmod rather than the decimal result of /.

TAKE IT FURTHER

Reserve a five-minute break after the third appointment. Subtract that break before allocating slots and check that the report still accounts for 180 minutes.

LAB NOTES

What did the checks prove? What remains untested?

PYTHON FOUNDATIONS / PROJECT BRIEF

Triage a Repair Request

THE ASSIGNMENT

Intake volunteers sort repair requests by conditions that must be evaluated in order. Items with signs of electrical damage go to the safety lead. Other items go to a staffed desk when one is available. The remaining items go to the waiting table. Python runs the first matching branch in an if, elif, else chain. A branch can therefore represent a policy priority. Use a boolean value for each observation and keep the routing decision separate from the item label.

ACCEPTANCE TARGET

With damaged and desk_ready both True, the route is Safety review.

PLAN THE BOUNDARY

Give the item a readable label and store damaged and desk_ready as boolean values.

BEFORE YOU CODE

Describe the input and result you will use for your first run.

Predict one case the program should reject or handle separately.

PYTHON FOUNDATIONS / GUIDED BUILD

Triage a Repair Request

Create a working copy of day_003.py. Use the project contract and implement each step before comparing the reference.

STEP 01

Give the item a readable label and store damaged and desk_ready as boolean values.

STEP 02

Put the electrical damage condition in the first branch because it takes priority over desk availability.

STEP 03

Use elif for an available desk and else for the waiting table. Assign route exactly once.

STEP 04

Print the label and the selected route in a single line.

STEP 05

Change both inputs to True and verify that the safety route still wins.

FIRST EXPERIMENT

Run the smallest valid example. Record the exact result you observed.

PYTHON FOUNDATIONS / REFERENCE SOLUTION

Triage a Repair Request

day_003.py | Part 1 of 2

```
item = "Table lamp"
damaged = True
desk_ready = True

if damaged:
    route = "Safety review"
elif desk_ready:
```

READ THE DESIGN

Follow the assignments in order. Write down a value after each transformation and compare it to the acceptance target.

PYTHON FOUNDATIONS / RUN AND INSPECT

Triage a Repair Request

day_003.py | Part 2 of 2

```
    route = "Repair desk"
else:
    route = "Waiting table"

assert route == "Safety review"
print(f"{item} -> {route}")
```

RUN FROM THE EXTRACTED BUNDLE

python projects/day_003.py

OBSERVED CONSOLE OUTPUT

```
Table lamp -> Safety review
```

PYTHON FOUNDATIONS / CHECK AND EXTEND

Triage a Repair Request

VERIFY THE BEHAVIOR

01 With damaged and desk_ready both True, the route is Safety review.

02 Set damaged to False and update the assertion. A ready desk yields Repair desk.

03 Set both values to False and update the assertion. The route is Waiting table.

DEBUG A LIKELY FAILURE

If a damaged item goes to the desk, inspect the first branch and the input types. The string 'False' is nonempty and evaluates as true.

TAKE IT FURTHER

Introduce a second safety condition for a leaking battery. Combine both safety observations with or and test a request that matches both.

LAB NOTES

What did the checks prove? What remains untested?