

NEXT STEP COURSE

Python

From Zero

Volume 2

Build Useful Python Tools

Digital Course Edition

Flow charts, worked examples, practical programs, workbook tasks, and code files.

```
python console

>>> tools = ['menus', 'files', 'reports']
>>> for tool in tools:
...     print('Build:', tool)
Build: menus
Build: files
Build: reports
>>> print('Useful programs.')
Useful programs.
```

How To Use This Course

Volume 2 continues from the first beginner course

1. Review

Skim the Volume 1 idea if the topic feels rusty.

2. Read

Read the plain-English idea before the code.

3. Trace

Follow the diagram and say what each step does.

4. Type

Type the program yourself. Copying teaches less.

5. Run

Run the program and read any error calmly.

6. Change

Make one small improvement and run it again.

Copyright Notice

Aldermere Systems publication

Aldermere Systems publication

© 2026 Aldermere Systems. All rights reserved. This publication and its supporting code examples are produced for Aldermere Systems. You may use the course for learning, but the publication design, text, diagrams, structure, and packaged materials remain the property of Aldermere Systems.

Course Contents

Screen-friendly chapter map

Module 1**Quick Refresh**

Skills checklist app

Module 2**Better Menus**

Multi-tool command menu

Module 3**Validating User Input**

Safe input helpers

Module 4**Dates and Times**

Daily journal logger

Module 5**Random Choices and Simple Games**

Number guessing game

Module 6**CSV Files**

Expense tracker saved to CSV

Module 7**JSON Files**

Profile saver

Module 8**Searching and Filtering Data**

Contact search tool

Module 9**Sorting and Reports**

Mini sales report generator

Module 10**Cleaner Functions**

Refactored to-do app

Module 11**Planning a Small Project**

Project planner worksheet

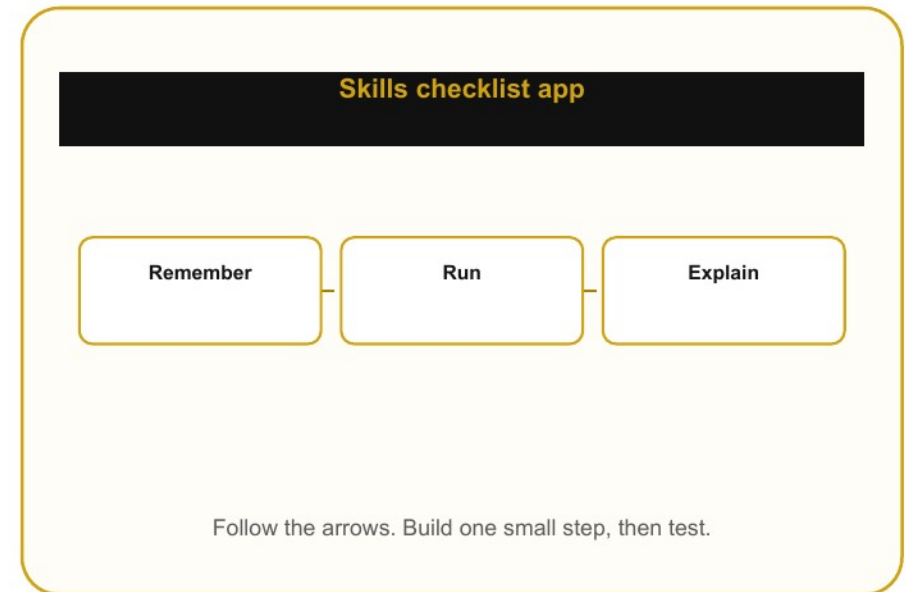
Module 12**Final Project**

Personal productivity dashboard

Module 1: Quick Refresh

What this module teaches

- Review variables, input, if statements, loops, lists, dictionaries, functions, files, and try/except.
- Understand how small skills combine into useful tools.
- Use a checklist program to see progress clearly.



Plain English idea

Reconnect Volume 1 skills so the learner starts confident, not cold.

Module 1: Program (1 of 2)

Useful working program

This program builds the skills checklist app. Type it, run it, then change one small part.

FILE: 01_skills_checklist.py

```
skills = [  
    "variables",  
    "input",  
    "if statements",  
    "loops",  
    "lists",  
    "dictionaries",  
    "functions",  
    "files",  
    "error handling",  
]  
  
print("PYTHON SKILLS CHECKLIST")  
  
for skill in skills:
```

Try it next Run it once, change one value, then run it again.

Module 1: Program (2 of 2)

Useful working program

This program builds the skills checklist app. Type it, run it, then change one small part.

FILE: 01_skills_checklist.py

```
answer = input(f"Do you remember {skill}? yes/no: ").lower()

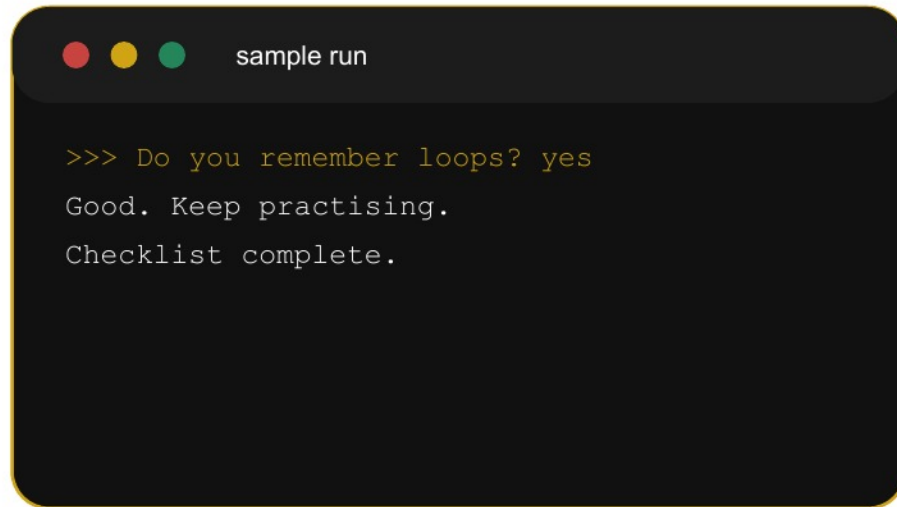
if answer == "yes":
    print("Good. Keep practising.")
else:
    print("No problem. Review it before moving on.")

print("Checklist complete.")
```

Try it next Run it once, change one value, then run it again.

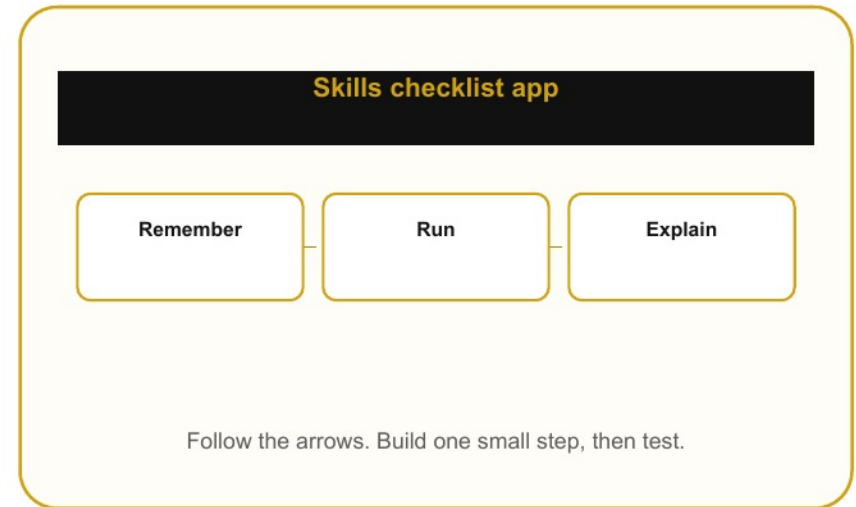
Module 1: Example

See the idea running



```
sample run

>>> Do you remember loops? yes
Good. Keep practising.
Checklist complete.
```



What happened

The learner gives input, Python follows the skills checklist app steps, and the result appears on screen or in a local file.

Module 1: Workbook

Write, predict, change, test

Task 1

Add three more skills to the list.

Task 2

Count how many times the learner answered yes.

Task 3

Print a final confidence score.

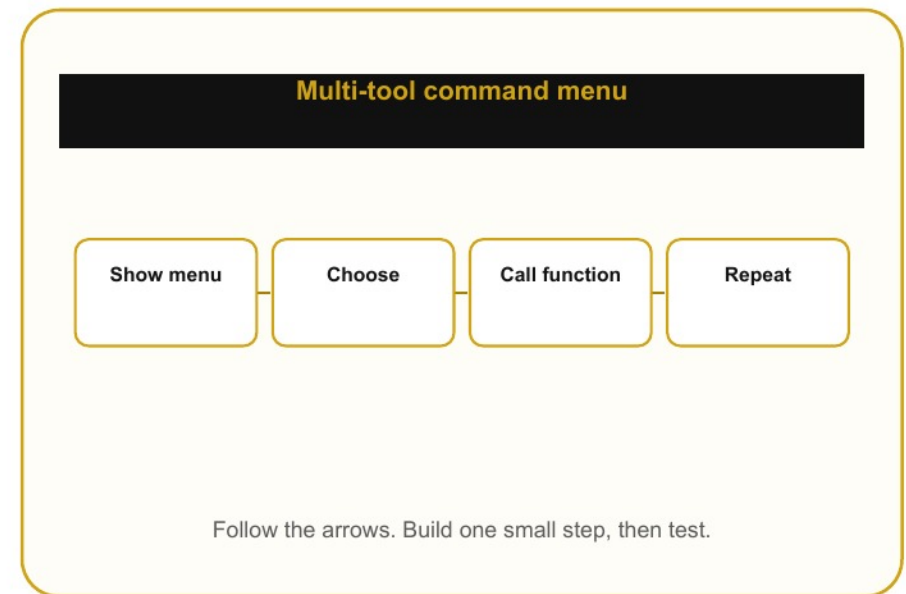
Answer guide hint

The important part is making one small working change before moving to the next task.

Module 2: Better Menus

What this module teaches

- Use one main loop for the menu.
- Use functions to keep each menu action tidy.
- Avoid copying the same code again and again.



Plain English idea

Build menu programs that feel organised and easy to expand.

Module 2: Program (1 of 2)

Useful working program

This program builds the multi-tool command menu. Type it, run it, then change one small part.

FILE: 02_multi_tool_menu.py

```
def show_greeting():
    name = input("Name: ")
    print("Hello", name)

def quick_calculator():
    first = float(input("First number: "))
    second = float(input("Second number: "))
    print("Total:", first + second)

while True:
    print("\nMAIN MENU")
    print("1. Greeting")
    print("2. Add two numbers")
```

Try it next Run it once, change one value, then run it again.

Module 2: Program (2 of 2)

Useful working program

This program builds the multi-tool command menu. Type it, run it, then change one small part.

FILE: 02_multi_tool_menu.py

```
print("3. Exit")

choice = input("Choose: ")

if choice == "1":
    show_greeting()
elif choice == "2":
    quick_calculator()
elif choice == "3":
    print("Goodbye.")
    break
else:
    print("Please choose 1, 2, or 3.")
```

Try it next Run it once, change one value, then run it again.

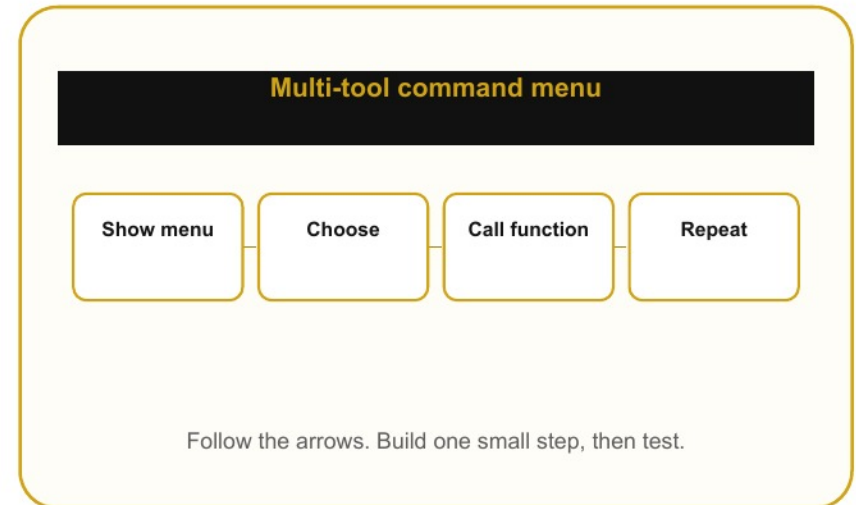
Module 2: Example

See the idea running



```
sample run

>>> Choose: 2
First number: 10
Second number: 5
Total: 15.0
```



What happened

The learner gives input, Python follows the multi-tool command menu steps, and the result appears on screen or in a local file.

Module 2: Workbook

Write, predict, change, test

Task 1

Add a fourth menu option.

Task 2

Move the menu printing into its own function.

Task 3

Make the program keep running after invalid input.

Answer guide hint

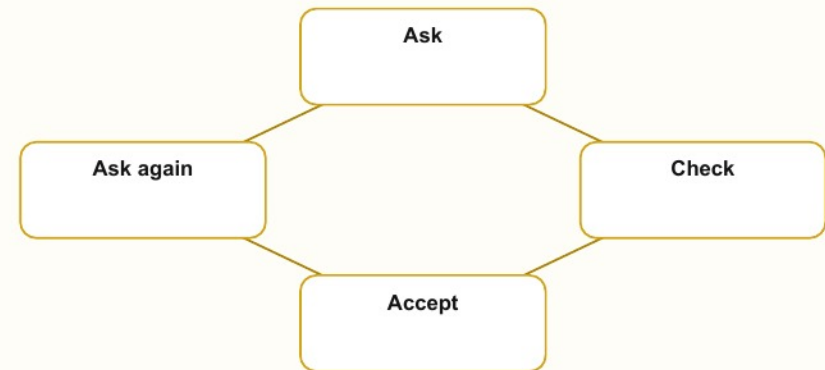
The important part is making one small working change before moving to the next task.

Module 3: Validating User Input

What this module teaches

- Ask again when the user enters bad input.
- Create reusable input helper functions.
- Use validation for numbers, menu choices, and empty text.

Safe input helpers



Plain English idea

Stop simple typing mistakes from breaking beginner programs.

Module 3: Program (1 of 2)

Useful working program

This program builds the safe input helpers. Type it, run it, then change one small part.

FILE: 03_safe_input_helpers.py

```
def get_number(prompt):
    while True:
        try:
            return float(input(prompt))
        except ValueError:
            print("Please enter a number.")

def get_choice(prompt, allowed):
    while True:
        choice = input(prompt)
        if choice in allowed:
            return choice
        print("Choose one of:", ", ".join(allowed))
```

Try it next Run it once, change one value, then run it again.

Module 3: Program (2 of 2)

Useful working program

This program builds the safe input helpers. Type it, run it, then change one small part.

FILE: 03_safe_input_helpers.py

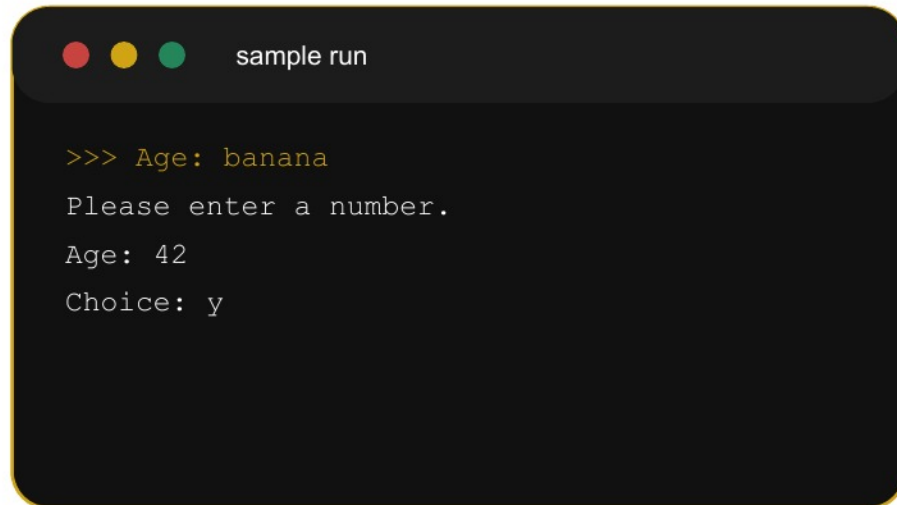
```
age = get_number("Age: ")
choice = get_choice("Continue? y/n: ", ["y", "n"])

print("Age:", age)
print("Choice:", choice)
```

Try it next Run it once, change one value, then run it again.

Module 3: Example

See the idea running



```
sample run

>>> Age: banana
Please enter a number.
Age: 42
Choice: y
```



What happened

The learner gives input, Python follows the safe input helpers steps, and the result appears on screen or in a local file.

Module 3: Workbook

Write, predict, change, test

Task 1

Create a `get_text` function that rejects blank input.

Task 2

Use `get_number` in a budget program.

Task 3

Use `get_choice` in a menu program.

Answer guide hint

The important part is making one small working change before moving to the next task.

Module 4: Dates and Times

What this module teaches

- Import datetime from the standard library.
- Create today strings for filenames and records.
- Save notes with dates so they are useful later.

Daily journal logger

Get time

Format

Write file

Follow the arrows. Build one small step, then test.

Plain English idea

Use timestamps for logs, journals, and saved records.

Module 4: Program

Useful working program

This program builds the daily journal logger. Type it, run it, then change one small part.

FILE: 04_daily_journal_logger.py

```
from datetime import datetime

now = datetime.now()
stamp = now.strftime("%Y-%m-%d %H:%M")

entry = input("Write today's journal entry: ")

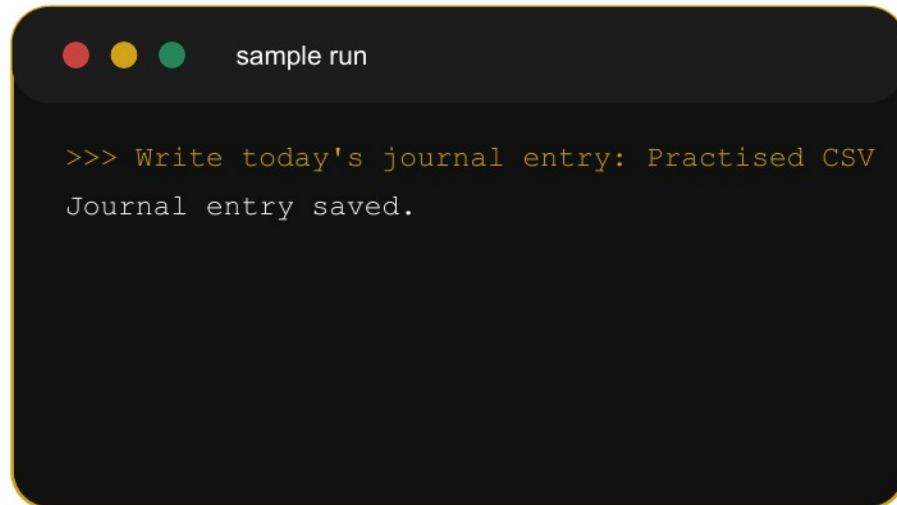
with open("journal.txt", "a", encoding="utf-8") as file:
    file.write(f"[{stamp}] {entry}\n")

print("Journal entry saved.")
```

Try it next Run it once, change one value, then run it again.

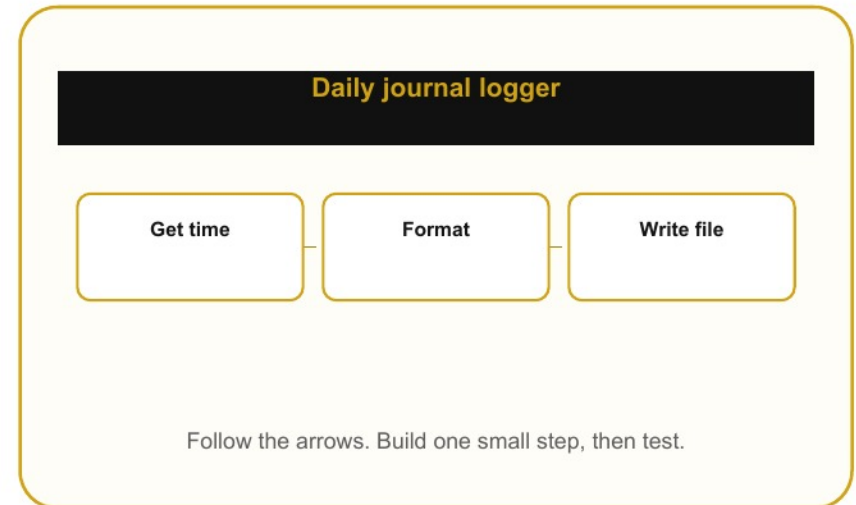
Module 4: Example

See the idea running



```
sample run

>>> Write today's journal entry: Practised CSV
Journal entry saved.
```



What happened

The learner gives input, Python follows the daily journal logger steps, and the result appears on screen or in a local file.

Module 4: Workbook

Write, predict, change, test

Task 1

Change the date format to day/month/year.

Task 2

Add a menu option to read the journal.

Task 3

Save entries to a file named after the current month.

Answer guide hint

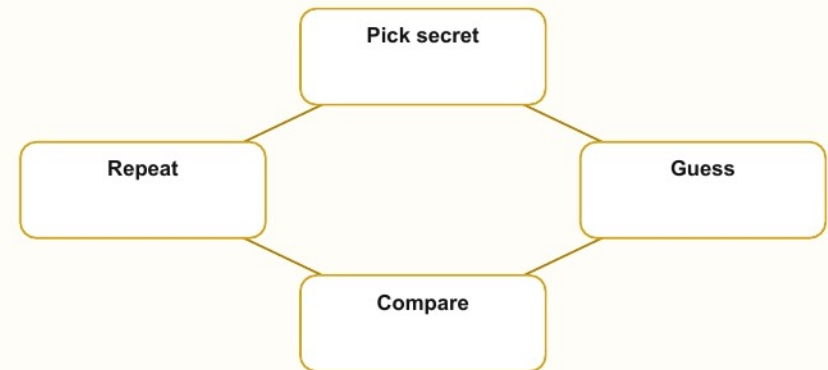
The important part is making one small working change before moving to the next task.

Module 5: Random Choices and Simple Games

What this module teaches

- Use `random.randint` for number games.
- Use `random.choice` for picking from a list.
- Keep the game loop simple and readable.

Number guessing game



Plain English idea

Use randomness for games, quizzes, and decision tools.

Module 5: Program (1 of 2)

Useful working program

This program builds the number guessing game. Type it, run it, then change one small part.

FILE: 05_number_guessing_game.py

```
import random

def get_number(prompt):
    while True:
        try:
            return int(input(prompt))
        except ValueError:
            print("Please enter a whole number.")

secret = random.randint(1, 20)
tries = 0

print("Guess the number from 1 to 20.")
```

Try it next Run it once, change one value, then run it again.

Module 5: Program (2 of 2)

Useful working program

This program builds the number guessing game. Type it, run it, then change one small part.

FILE: 05_number_guessing_game.py

```
while True:
    guess = get_number("Your guess: ")
    tries += 1

    if guess < secret:
        print("Too low.")
    elif guess > secret:
        print("Too high.")
    else:
        print("Correct.")
        print("Tries:", tries)
        break
```

Try it next Run it once, change one value, then run it again.

Module 5: Example

See the idea running



```
sample run

>>> Your guess: 8
Too low.
Your guess: 14
Correct.
```



What happened

The learner gives input, Python follows the number guessing game steps, and the result appears on screen or in a local file.

Module 5: Workbook

Write, predict, change, test

Task 1

Limit the game to five tries.

Task 2

Let the learner choose the maximum number.

Task 3

Print a different message for first-try success.

Answer guide hint

The important part is making one small working change before moving to the next task.

Module 6: CSV Files

What this module teaches

- Understand CSV as rows and columns.
- Use the csv module instead of building rows by hand.
- Append new records safely.

Expense tracker saved to CSV

Input

Row

CSV file

Spreadsheet

Follow the arrows. Build one small step, then test.

Plain English idea

Store table-style records that spreadsheets can open.

Module 6: Program (1 of 2)

Useful working program

This program builds the expense tracker saved to csv. Type it, run it, then change one small part.

FILE: 06_csv_expense_tracker.py

```
import csv
from datetime import date

def get_amount(prompt):
    while True:
        try:
            return float(input(prompt))
        except ValueError:
            print("Please enter a number.")

item = input("Expense item: ")
amount = get_amount("Amount: GBP ")
category = input("Category: ")
```

Try it next Run it once, change one value, then run it again.

Module 6: Program (2 of 2)

Useful working program

This program builds the expense tracker saved to csv. Type it, run it, then change one small part.

FILE: 06_csv_expense_tracker.py

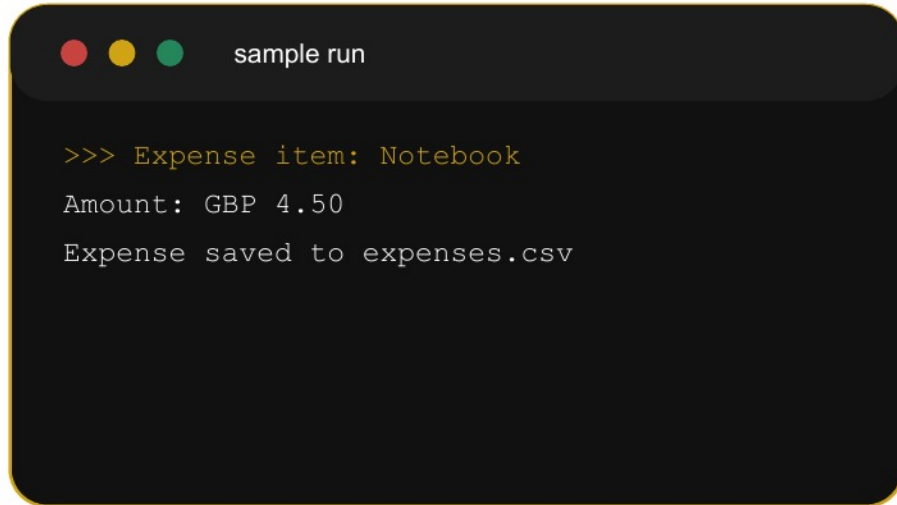
```
with open("expenses.csv", "a", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerow([date.today(), item, category, amount])

print("Expense saved to expenses.csv")
```

Try it next Run it once, change one value, then run it again.

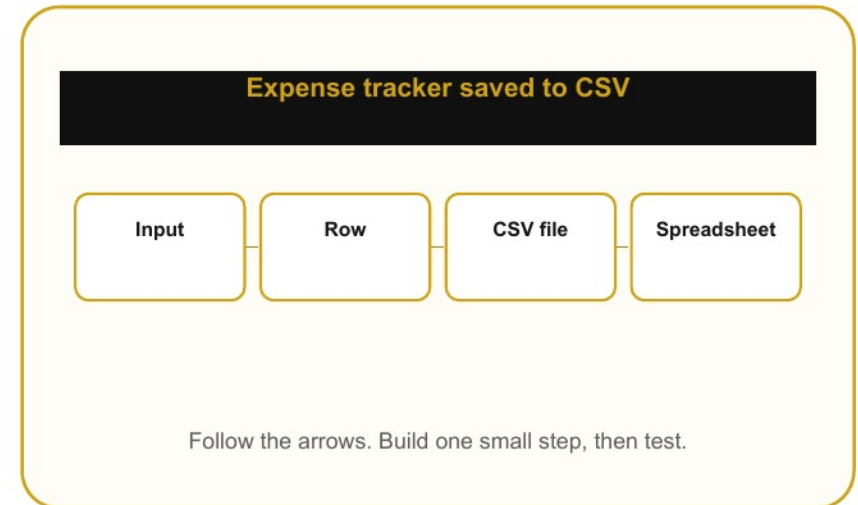
Module 6: Example

See the idea running



```
sample run

>>> Expense item: Notebook
Amount: GBP 4.50
Expense saved to expenses.csv
```



What happened

The learner gives input, Python follows the expense tracker saved to csv steps, and the result appears on screen or in a local file.

Module 6: Workbook

Write, predict, change, test

Task 1

Add a notes column.

Task 2

Read the CSV and print all expenses.

Task 3

Print the total spent.

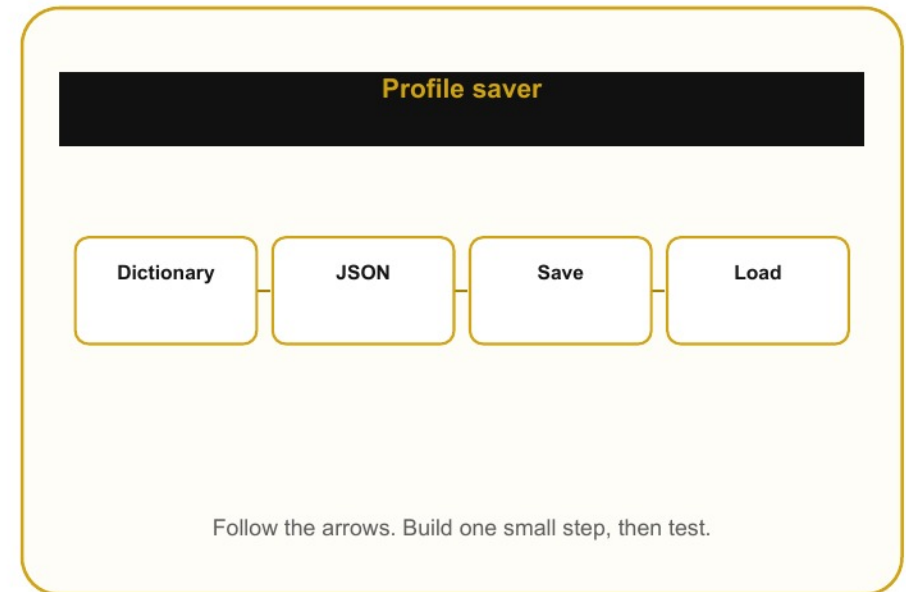
Answer guide hint

The important part is making one small working change before moving to the next task.

Module 7: JSON Files

What this module teaches

- Use dictionaries and lists with `json.dump`.
- Use `json.load` to bring saved information back.
- Choose JSON when the data has labels and structure.



Plain English idea

Save structured information such as settings, profiles, and lists.

Module 7: Program

Useful working program

This program builds the profile saver. Type it, run it, then change one small part.

FILE: 07_json_profile_saver.py

```
import json

profile = {
    "name": input("Name: "),
    "goal": input("Learning goal: "),
    "favourite_tool": input("Favourite Python tool so far: "),
}

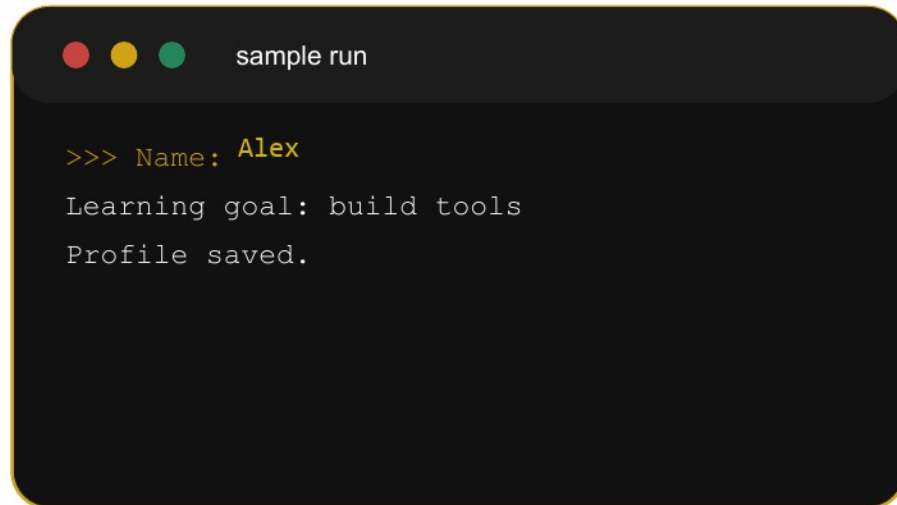
with open("profile.json", "w", encoding="utf-8") as file:
    json.dump(profile, file, indent=4)

print("Profile saved.")
```

Try it next Run it once, change one value, then run it again.

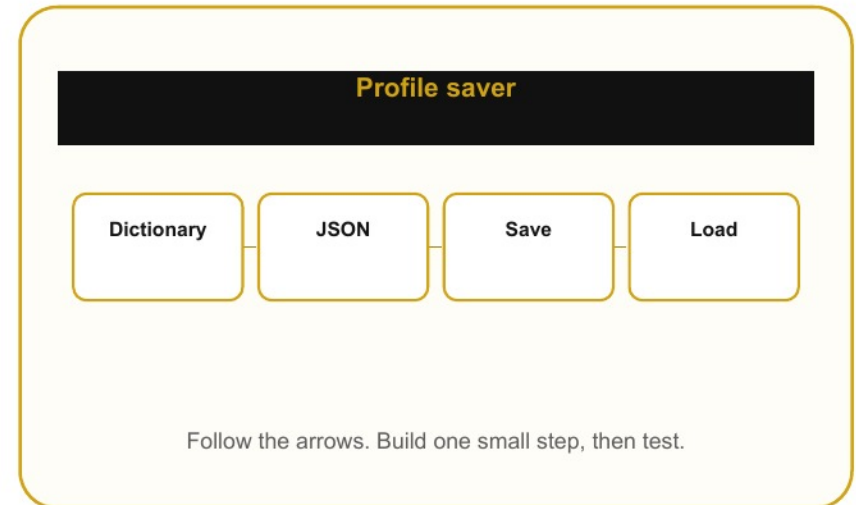
Module 7: Example

See the idea running



```
sample run

>>> Name: Alex
Learning goal: build tools
Profile saved.
```



What happened

The learner gives input, Python follows the profile saver steps, and the result appears on screen or in a local file.

Module 7: Workbook

Write, predict, change, test

Task 1

Add a list of three learning goals.

Task 2

Read the saved profile back and print it.

Task 3

Add a menu with save profile and view profile.

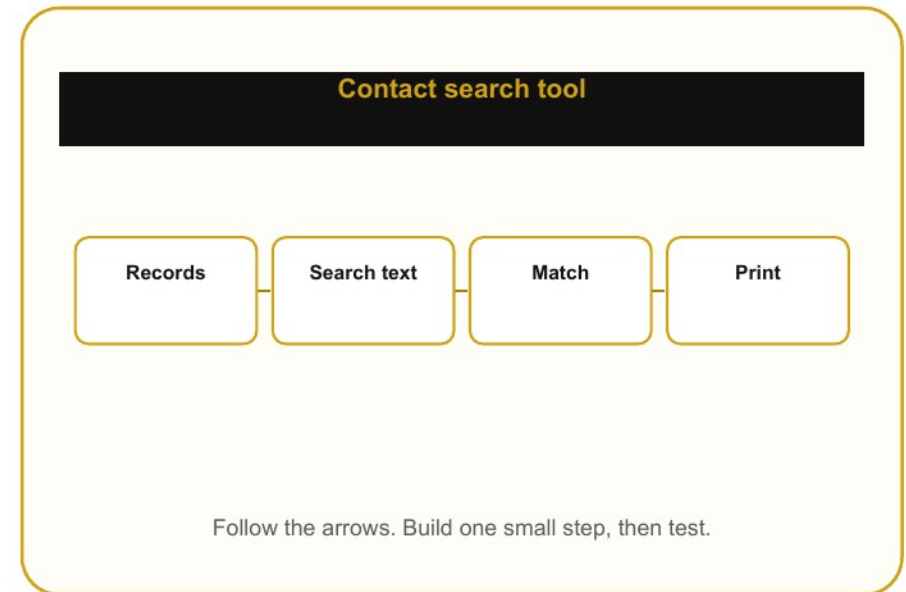
Answer guide hint

The important part is making one small working change before moving to the next task.

Module 8: Searching and Filtering Data

What this module teaches

- Loop through dictionaries in a list.
- Compare lower-case text for friendly searching.
- Print only records that match.



Plain English idea

Find useful records inside lists and saved data.

Module 8: Program

Useful working program

This program builds the contact search tool. Type it, run it, then change one small part.

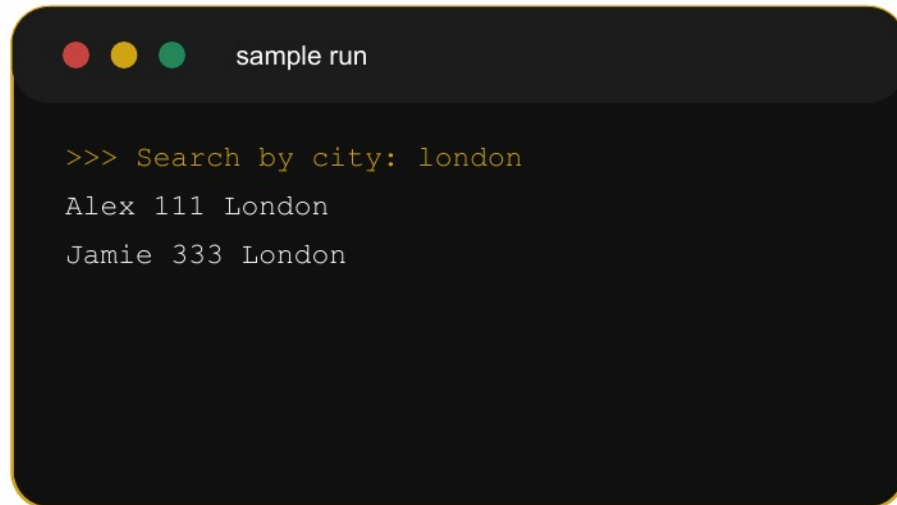
FILE: 08_contact_search_tool.py

```
contacts = [  
    {"name": "Alex", "phone": "111", "city": "London"},  
    {"name": "Sam", "phone": "222", "city": "Bristol"},  
    {"name": "Jamie", "phone": "333", "city": "London"},  
]  
  
search = input("Search by city: ").lower()  
  
for contact in contacts:  
    if contact["city"].lower() == search:  
        print(contact["name"], contact["phone"], contact["city"])
```

Try it next Run it once, change one value, then run it again.

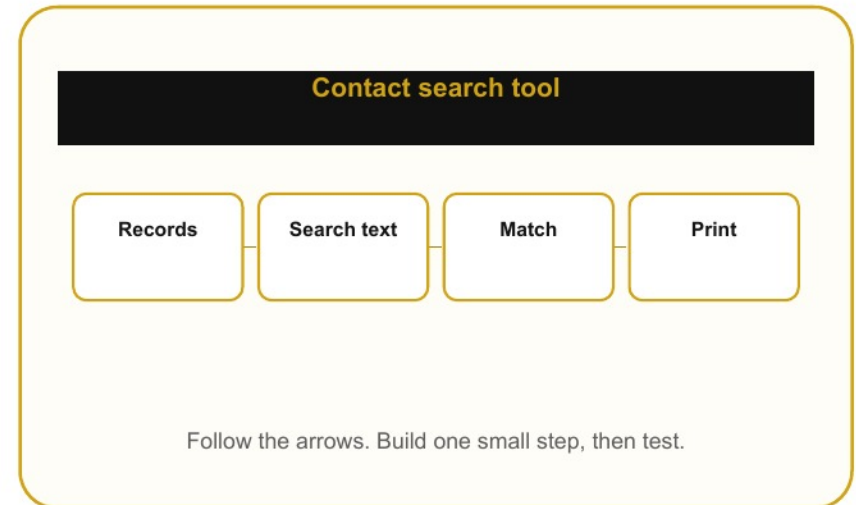
Module 8: Example

See the idea running



```
sample run

>>> Search by city: london
Alex 111 London
Jamie 333 London
```



What happened

The learner gives input, Python follows the contact search tool steps, and the result appears on screen or in a local file.

Module 8: Workbook

Write, predict, change, test

Task 1

Search by name instead of city.

Task 2

Allow partial matches, such as 'lon'.

Task 3

Show a message if no records are found.

Answer guide hint

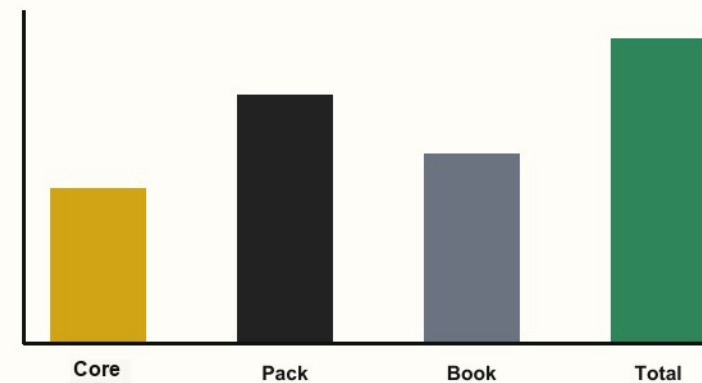
The important part is making one small working change before moving to the next task.

Module 9: Sorting and Reports

What this module teaches

- Use sorted to organise data.
- Use sum to total simple numbers.
- Print reports that answer a real question.

Mini sales report generator



Plain English idea

Turn saved records into useful summaries.

Module 9: Program

Useful working program

This program builds the mini sales report generator. Type it, run it, then change one small part.

FILE: 09_sales_report_generator.py

```
sales = [
    {"product": "Starter Course", "amount": 0},
    {"product": "Template Pack", "amount": 19},
    {"product": "Workbook", "amount": 9},
]

total = sum(item["amount"] for item in sales)
ordered = sorted(sales, key=lambda item: item["amount"], reverse=True)

print("SALES REPORT")
print("Total sales: GBP", total)

for item in ordered:
    print(item["product"], "GBP", item["amount"])
```

Try it next Run it once, change one value, then run it again.

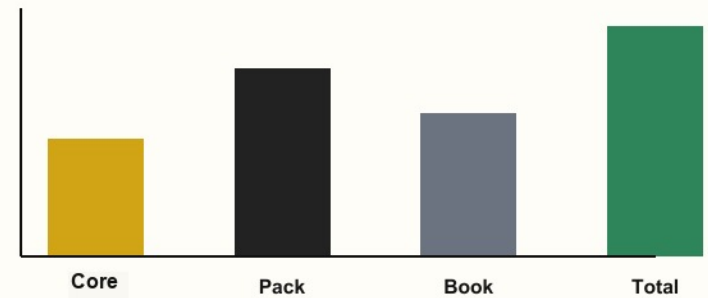
Module 9: Example

See the idea running

sample run

```
>>> SALES REPORT  
Total sales: GBP 28  
Template Pack GBP 19
```

Mini sales report generator



What happened

The learner gives input, Python follows the mini sales report generator steps, and the result appears on screen or in a local file.

Module 9: Workbook

Write, predict, change, test

Task 1

Add more sales records.

Task 2

Print the highest sale only.

Task 3

Add a count of how many sales were made.

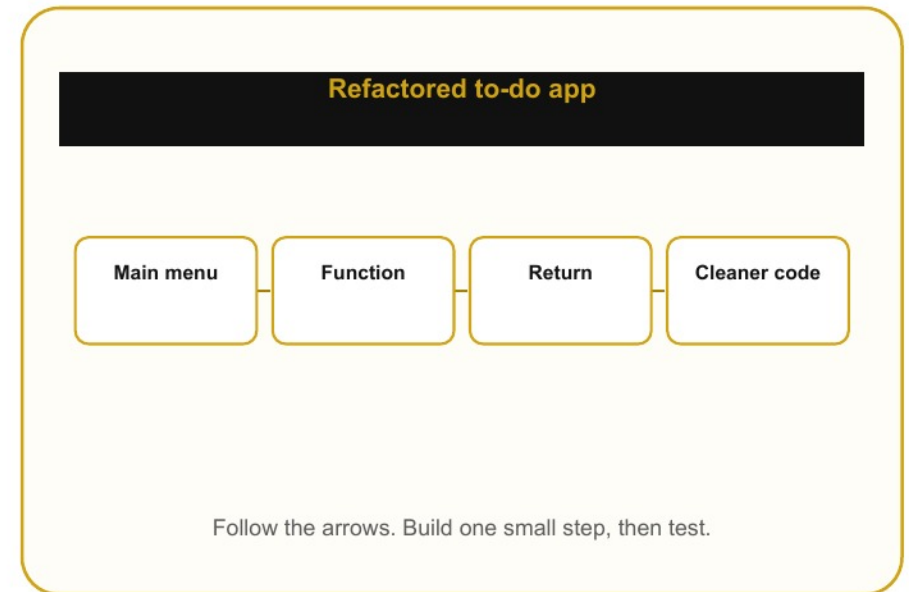
Answer guide hint

The important part is making one small working change before moving to the next task.

Module 10: Cleaner Functions

What this module teaches

- Give each function one clear job.
- Use names that explain what the function does.
- Keep the main loop short.



Plain English idea

Make bigger programs easier to understand and repair.

Module 10: Program (1 of 3)

Useful working program

This program builds the refactored to-do app. Type it, run it, then change one small part.

FILE: 10_refactored_todo_app.py

```
tasks = []

def add_task():
    task = input("Task: ")
    tasks.append(task)
    print("Task added.")

def view_tasks():
    if not tasks:
        print("No tasks yet.")
        return

    for number, task in enumerate(tasks, start=1):
```

Try it next Run it once, change one value, then run it again.

Module 10: Program (2 of 3)

Useful working program

This program builds the refactored to-do app. Type it, run it, then change one small part.

FILE: 10_refactored_todo_app.py

```
        print(number, task)

while True:
    print("\n1. Add task")
    print("2. View tasks")
    print("3. Exit")

    choice = input("Choose: ")

    if choice == "1":
        add_task()
    elif choice == "2":
        view_tasks()
    elif choice == "3":
```

Try it next Run it once, change one value, then run it again.

Module 10: Program (3 of 3)

Useful working program

This program builds the refactored to-do app. Type it, run it, then change one small part.

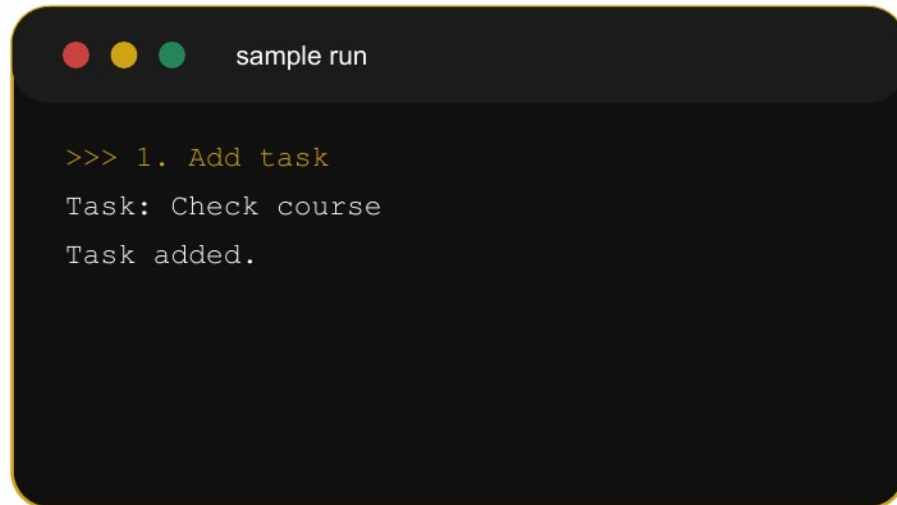
FILE: 10_refactored_todo_app.py

```
break
```

Try it next Run it once, change one value, then run it again.

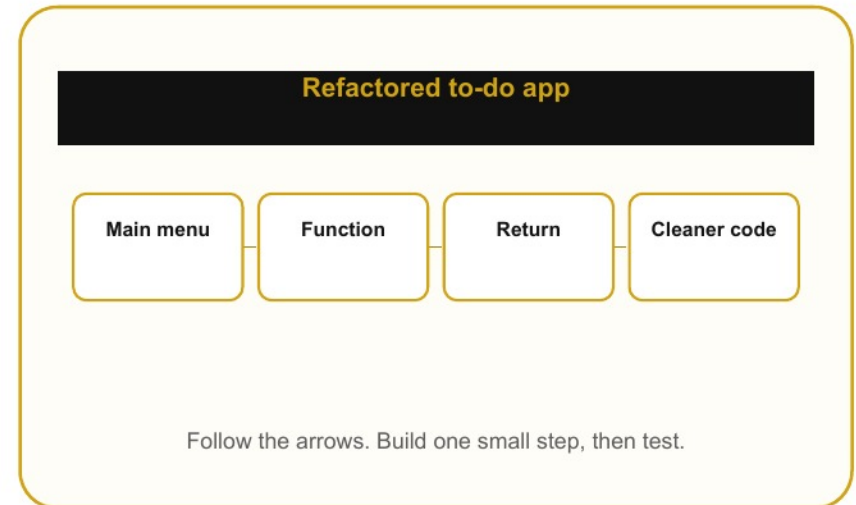
Module 10: Example

See the idea running



```
sample run

>>> 1. Add task
Task: Check course
Task added.
```



What happened

The learner gives input, Python follows the refactored to-do app steps, and the result appears on screen or in a local file.

Module 10: Workbook

Write, predict, change, test

Task 1

Add a `remove_task` function.

Task 2

Add a `save_tasks` function.

Task 3

Add a `load_tasks` function.

Answer guide hint

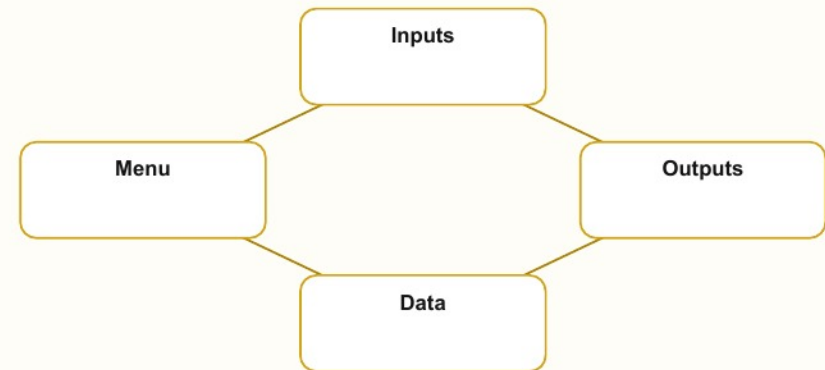
The important part is making one small working change before moving to the next task.

Module 11: Planning a Small Project

What this module teaches

- Write the inputs, outputs, saved data, menu options, and functions first.
- Build one feature at a time.
- Test each feature before adding the next one.

Project planner worksheet



Plain English idea

Teach learners to plan before coding so projects do not become tangled.

Module 11: Program

Useful working program

This program builds the project planner worksheet. Type it, run it, then change one small part.

FILE: 11_project_planner.py

```
project = {}

project["name"] = input("Project name: ")
project["problem"] = input("What problem does it solve? ")
project["inputs"] = input("What will the user type in? ")
project["outputs"] = input("What should the program show? ")
project["save_file"] = input("Will it save a file? ")

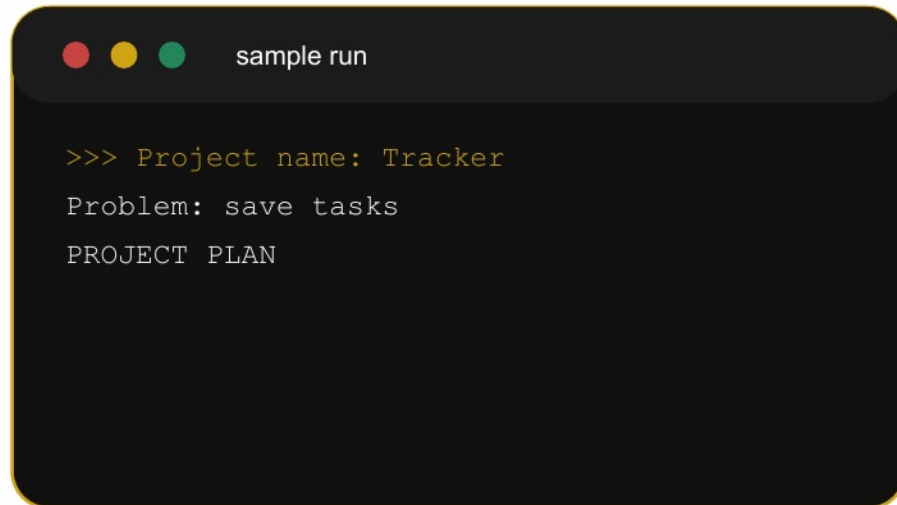
print("\nPROJECT PLAN")

for key, value in project.items():
    print(key.title() + ":", value)
```

Try it next Run it once, change one value, then run it again.

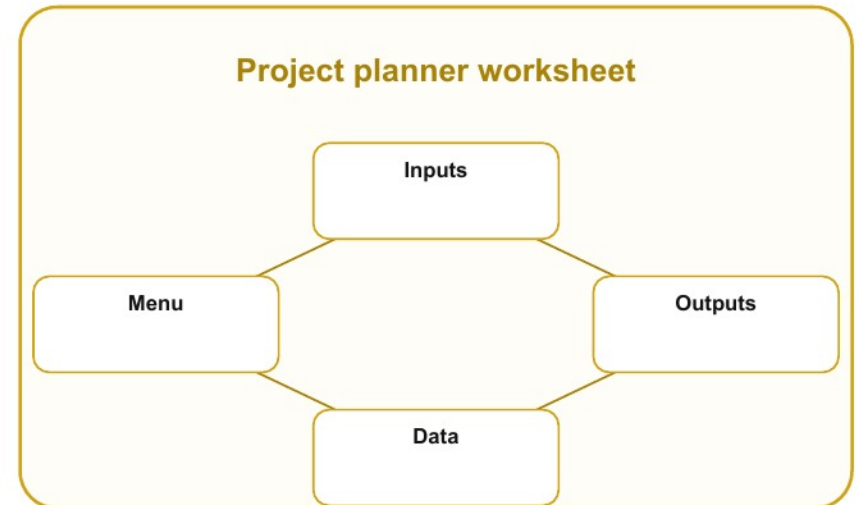
Module 11: Example

See the idea running



```
sample run

>>> Project name: Tracker
Problem: save tasks
PROJECT PLAN
```



What happened

The learner gives input, Python follows the project planner worksheet steps, and the result appears on screen or in a local file.

Module 11: Workbook

Write, predict, change, test

Task 1

Add a field for menu options.

Task 2

Add a field for functions needed.

Task 3

Save the plan to a text file.

Answer guide hint

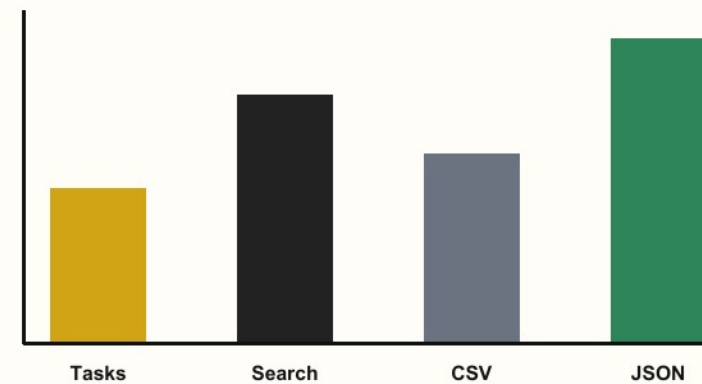
The important part is making one small working change before moving to the next task.

Module 12: Final Project

What this module teaches

- Build a personal productivity dashboard step by step.
- Keep data in simple local files.
- Use the Volume 2 skills in one practical program.

Personal productivity dashboard



Plain English idea

Combine menus, validation, dates, files, search, reports, CSV, JSON, and functions.

Module 12: Program (1 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
import csv
import json
from datetime import date

TASKS_FILE = "dashboard_tasks.txt"
EXPENSES_FILE = "dashboard_expenses.csv"
PROFILE_FILE = "dashboard_profile.json"

def get_number(prompt):
    while True:
        try:
            return float(input(prompt))
        except ValueError:
            print("Please enter a number.")
```

Try it next Run it once, change one value, then run it again.

Module 12: Program (2 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
def add_task():
    task = input("Task: ")
    with open(TASKS_FILE, "a", encoding="utf-8") as file:
        file.write(task + "\n")
    print("Task saved.")

def view_tasks():
    try:
        with open(TASKS_FILE, "r", encoding="utf-8") as file:
            tasks = file.readlines()
    except FileNotFoundError:
        print("No tasks yet.")
```

Try it next Run it once, change one value, then run it again.

Module 12: Program (3 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
        return

    for number, task in enumerate(tasks, start=1):
        print(number, task.strip())

def search_tasks():
    search = input("Search tasks: ").lower()

    try:
        with open(TASKS_FILE, "r", encoding="utf-8") as file:
            tasks = file.readlines()
    except FileNotFoundError:
        print("No tasks yet.")
    return
```

Try it next Run it once, change one value, then run it again.

Module 12: Program (4 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
found = False

for task in tasks:
    if search in task.lower():
        print(task.strip())
        found = True

if not found:
    print("No matching tasks found.")

def add_expense():
    item = input("Expense item: ")
    amount = get_number("Amount: GBP ")
```

Try it next Run it once, change one value, then run it again.

Module 12: Program (5 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
with open(EXPENSES_FILE, "a", newline="", encoding="utf-8") as file:
    writer = csv.writer(file)
    writer.writerow([date.today(), item, amount])

print("Expense saved.")

def expense_report():
    total = 0

    try:
        with open(EXPENSES_FILE, "r", encoding="utf-8") as file:
            reader = csv.reader(file)
            for row in reader:
```

Try it next Run it once, change one value, then run it again.

Module 12: Program (6 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
        print(row[0], row[1], "GBP " + row[2])
        total += float(row[2])
except FileNotFoundError:
    print("No expenses yet.")
    return

print("Total spent: GBP", round(total, 2))

def save_profile():
    profile = {
        "name": input("Name: "),
        "main_goal": input("Main goal: "),
        "updated": str(date.today()),
    }
```

Try it next Run it once, change one value, then run it again.

Module 12: Program (7 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
with open(PROFILE_FILE, "w", encoding="utf-8") as file:
    json.dump(profile, file, indent=4)

print("Profile saved.")

def view_profile():
    try:
        with open(PROFILE_FILE, "r", encoding="utf-8") as file:
            profile = json.load(file)
    except FileNotFoundError:
        print("No profile saved yet.")
    return
```

Try it next Run it once, change one value, then run it again.

Module 12: Program (8 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
for key, value in profile.items():
    print(key.title() + ":", value)

while True:
    print("\nPERSONAL PRODUCTIVITY DASHBOARD")
    print("1. Add task")
    print("2. View tasks")
    print("3. Search tasks")
    print("4. Add expense")
    print("5. Expense report")
    print("6. Save profile")
    print("7. View profile")
    print("8. Exit")
```

Try it next Run it once, change one value, then run it again.

Module 12: Program (9 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
choice = input("Choose: ")

if choice == "1":
    add_task()
elif choice == "2":
    view_tasks()
elif choice == "3":
    search_tasks()
elif choice == "4":
    add_expense()
elif choice == "5":
    expense_report()
elif choice == "6":
    save_profile()
elif choice == "7":
```

Try it next Run it once, change one value, then run it again.

Module 12: Program (10 of 10)

Useful working program

This program builds the personal productivity dashboard. Type it, run it, then change one small part.

FILE: 12_productivity_dashboard.py

```
        view_profile()
elif choice == "8":
    print("Goodbye.")
    break
else:
    print("Choose 1 to 8.")
```

Try it next Run it once, change one value, then run it again.

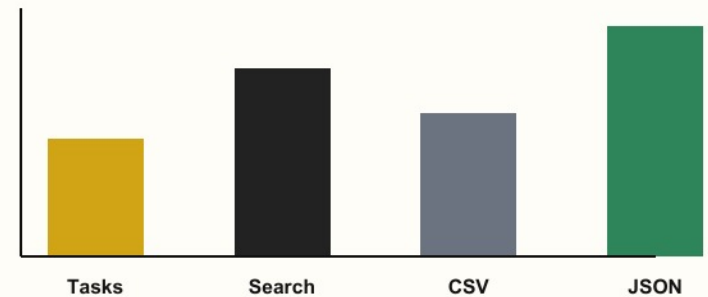
Module 12: Example

See the idea running

sample run

```
>>> 3. Search tasks  
Search tasks: invoice  
Send invoice
```

Personal productivity dashboard



What happened

The learner gives input, Python follows the personal productivity dashboard steps, and the result appears on screen or in a local file.

Module 12: Workbook

Write, predict, change, test

Task 1

Add a notes feature.

Task 2

Add search for tasks.

Task 3

Add safer number input for expenses.

Answer guide hint

The important part is making one small working change before moving to the next task.

Final Project Build Map

Personal productivity dashboard

Dashboard feature flow

Menu

Tasks

Search

CSV expenses

JSON profile

Report

Follow the arrows. Build one small step, then test.

Feature coverage



Answer Guide

Suggested checks

Module 1: Quick Refresh

- Expected result: learner can run 01_skills_checklist.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Module 2: Better Menus

- Expected result: learner can run 02_multi_tool_menu.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Module 3: Validating User Input

- Expected result: learner can run 03_safe_input_helpers.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Module 4: Dates and Times

- Expected result: learner can run 04_daily_journal_logger.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Answer Guide Continued

Suggested checks

Module 5: Random Choices and Simple Games

- Expected result: learner can run 05_number_guessing_game.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Module 6: CSV Files

- Expected result: learner can run 06_csv_expense_tracker.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Module 7: JSON Files

- Expected result: learner can run 07_json_profile_saver.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Module 8: Searching and Filtering Data

- Expected result: learner can run 08_contact_search_tool.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Answer Guide Continued

Suggested checks

Module 9: Sorting and Reports

- Expected result: learner can run 09_sales_report_generator.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Module 10: Cleaner Functions

- Expected result: learner can run 10_refactored_todo_app.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Module 11: Planning a Small Project

- Expected result: learner can run 11_project_planner.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Module 12: Final Project

- Expected result: learner can run 12_productivity_dashboard.py without errors.
- Good answer: one small change is made and tested.
- Stretch answer: the idea is reused in another mini-program.

Glossary

Plain-English terms

CSV

A simple table file. Spreadsheets can open it.

JSON

A structured file format for labelled data.

Validation

Checking input before the program trusts it.

Function

A named block of code that performs one job.

Report

A useful summary printed from stored data.

Refactor

Improve code shape without changing what it does.

Completion Page

Volume 2 checkpoint

Well done

You have moved from following beginner examples to building practical Python tools with menus, files, validation, search, reports, CSV, JSON, and functions.

- I can build menu-driven programs.
- I can validate user input.
- I can save and read simple data files.
- I can search, sort, and report on stored records.
- I can plan and build a small Python project step by step.