

PAID COURSE EDITION

Python From Zero

Volume 1

Learn Python From Absolute Zero

Digital Course Edition

Clear lessons, diagrams, examples, workbook tasks,
and matching Python files.

A screenshot of a Python console window titled "python console". The window has a dark background and a light-colored title bar with three colored buttons (red, yellow, green). The console shows the following code and output:

```
>>> print('Hello, world!')
Hello, world!
>>> name = input('Name: ')
Name: Alex
>>> print('Hello', name)
Hello Alex
```

Start Here

How to use this course

Read

Start with the plain-English explanation.

Look

Study the diagram before typing code.

Type

Type the program yourself so your fingers learn it.

Run

Run the program and read any error calmly.

Change

Make one small change and run it again.

Build

Use the projects to turn lessons into useful tools.

Course promise

By the end, a complete beginner can understand core Python basics and build useful small programs.

Copyright Notice

Aldermere Systems publication

Copyright

© 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.

Digital edition

This is a screen-friendly digital course. It is not prepared as a physical KDP paperback interior.

Course Contents

Screen-friendly chapter map

Module 1**First Steps**

Personal greeting

Module 2**Variables**

Profile card

Module 3**Numbers and Maths**

VAT calculator

Module 4**Text and Strings**

Listing title cleaner

Module 5**Decisions**

Password strength checker

Module 6**Loops**

Countdown timer

Module 7**Lists**

Shopping list maker

Module 8**Dictionaries**

Contact card

Module 9**Functions**

Tip calculator

Module 10**Files**

Simple notes app

Module 11**Errors**

Safe calculator

Module 12**Final Build**

Beginner Command Centre

Module 1: First Steps

Programming as step-by-step instructions

What this module teaches

- Understand what programming means
- Run `print()` to display a message
- Use `input()` to ask a question

Programming is a recipe

Ask

Store

Print

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

Plain English idea

Programming means giving Python clear instructions in the right order.

Module 1: Program

Useful working program

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

FILE: hello_greeting.py

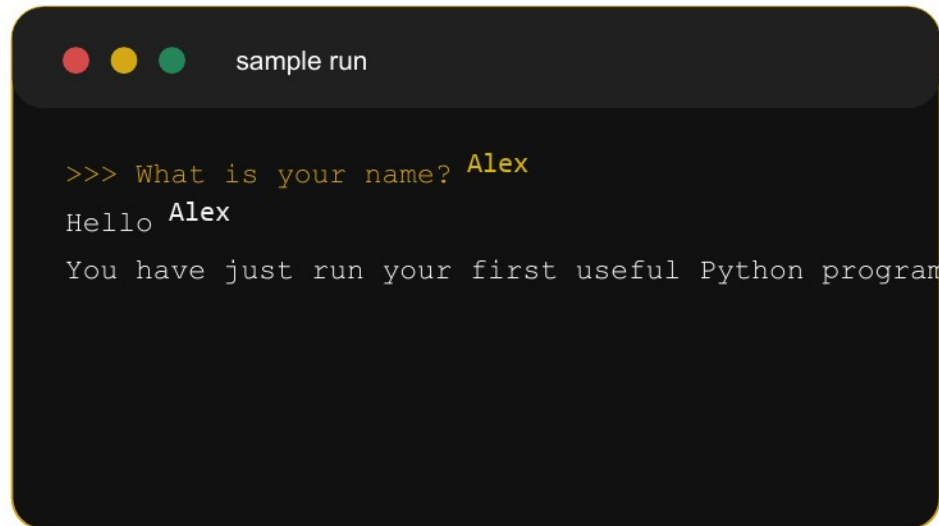
```
print("Hello, world!")
print("Welcome to Python From Zero.")

name = input("What is your name? ")
print("Hello", name)
print("You have just run your first useful Python program.")
```

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

Module 1: Example

See the idea running



```
sample run

>>> What is your name? Alex
Hello Alex
You have just run your first useful Python program
```



What happened

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

Module 1: Workbook

Write, predict, change, test

Task 1

Change the greeting message.

Task 2

Ask for a favourite hobby.

Task 3

Print the name and hobby together.

Answer guide hint

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

Module 2: Variables

Labelled boxes for information

What this module teaches

- Create simple variables
- Understand text, numbers and true or false values
- Use variables in a printed message

Variables are labelled boxes

Name

Age

Hobby

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

Plain English idea

A variable is a labelled box that stores one useful piece of information.

Module 2: Program

Useful working program

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

FILE: profile_card.py

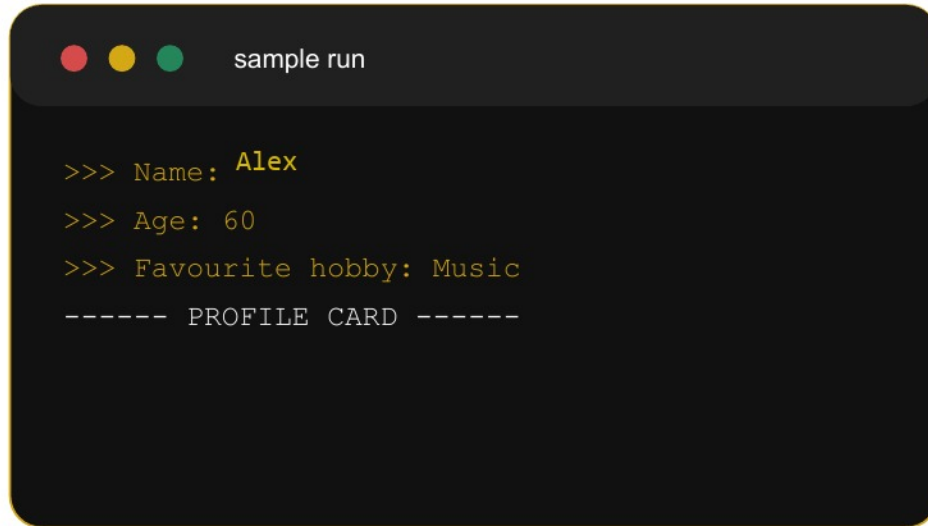
```
name = input("Name: ")
age = input("Age: ")
hobby = input("Favourite hobby: ")

print("----- PROFILE CARD -----")
print("Name:", name)
print("Age:", age)
print("Hobby:", hobby)
print("-----")
```

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

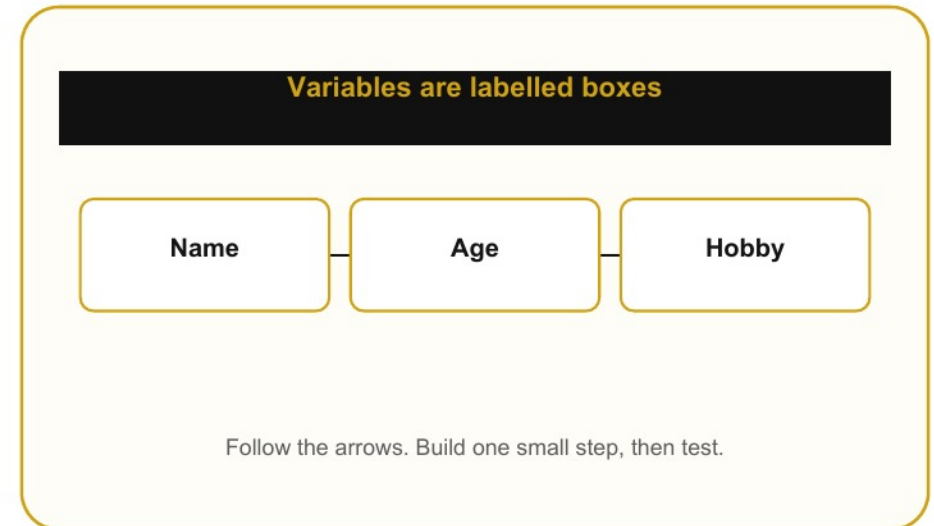
Module 2: Example

See the idea running



```
sample run

>>> Name: Alex
>>> Age: 60
>>> Favourite hobby: Music
----- PROFILE CARD -----
```



What happened

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

Module 2: Workbook

Write, predict, change, test

Task 1

Add a country question.

Task 2

Add a favourite food question.

Task 3

Print one full sentence from the answers.

Answer guide hint

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

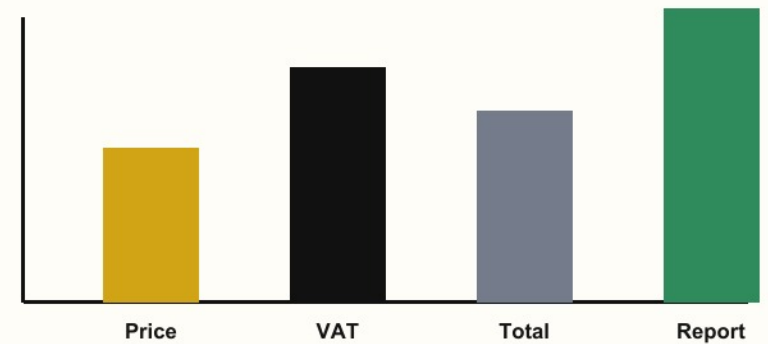
Module 3: Numbers and Maths

Python as a useful calculator

What this module teaches

- Use integers and decimals
- Convert typed input into numbers
- Calculate totals and percentages

Python can calculate



Plain English idea

Python can calculate totals, tax, tips, scores and measurements.

Module 3: Program

Useful working program

This program builds the vat calculator. Type it, run it, then change one small part.

FILE: vat_calculator.py

```
price = float(input("Enter price before VAT: £"))
vat_rate = 0.20

vat = price * vat_rate
total = price + vat

print("VAT: £", round(vat, 2))
print("Total price: £", round(total, 2))
```

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

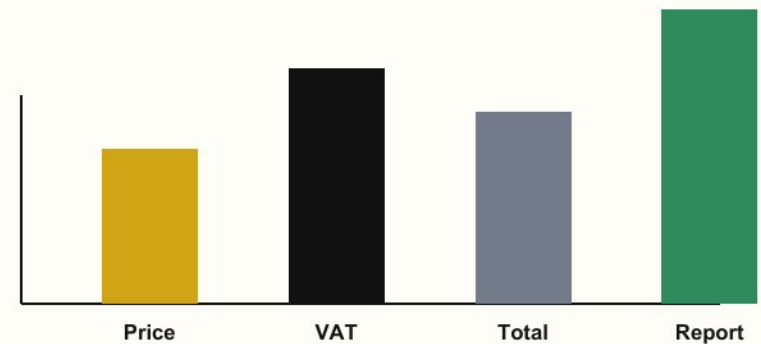
Module 3: Example

See the idea running

sample run

```
>>> Enter price before VAT: £10  
VAT: £ 2.0  
Total price: £ 12.0
```

Python can calculate



What happened

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

Module 3: Workbook

Write, predict, change, test

Task 1

Change the VAT rate.

Task 2

Ask for two prices and add them.

Task 3

Round the result to two decimal places.

Answer guide hint

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

Module 4: Text and Strings

Working with words

What this module teaches

- Use `strip()` to remove extra spaces
- Use `title()` to tidy a heading
- Use `len()` to count characters

Text goes through machines

Raw text

Strip

Title case

Count

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

Plain English idea

A string is text that Python can clean, count and format.

Module 4: Program

Useful working program

This program builds the listing title cleaner. Type it, run it, then change one small part.

FILE: listing_title_cleaner.py

```
title = input("Enter your product title: ")

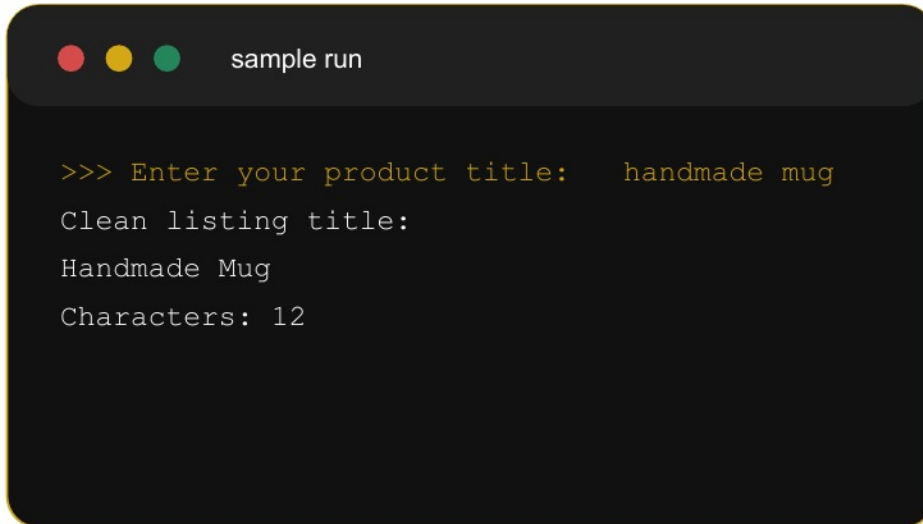
clean_title = title.strip().title()

print("Clean listing title:")
print(clean_title)
print("Characters:", len(clean_title))
```

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

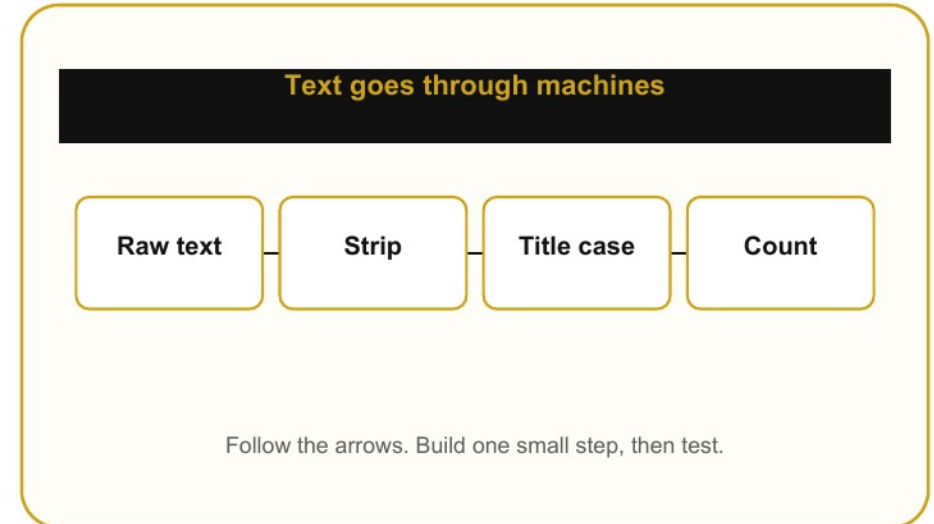
Module 4: Example

See the idea running



```
sample run

>>> Enter your product title:  handmade mug
Clean listing title:
Handmade Mug
Characters: 12
```



What happened

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

Module 4: Workbook

Write, predict, change, test

Task 1

Print the title in uppercase.

Task 2

Print the title in lowercase.

Task 3

Ask for a short description too.

Answer guide hint

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

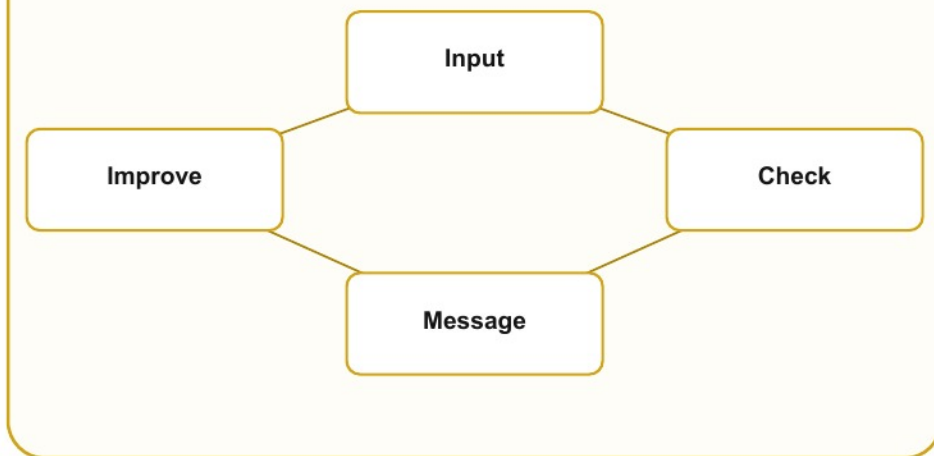
Module 5: Decisions

Programs that choose a path

What this module teaches

- Use if, elif and else
- Compare values
- Give different messages for different conditions

If statements make decisions



Plain English idea

An if statement lets a program choose what to do next.

Module 5: Program

Useful working program

This program builds the password strength checker. Type it, run it, then change one small part.

FILE: password_strength_checker.py

```
password = input("Enter a password: ")

if len(password) < 8:
    print("Weak password: too short.")
elif password.islower() or password.isupper():
    print("Medium password: mix upper and lower case.")
else:
    print("Stronger password.")
```

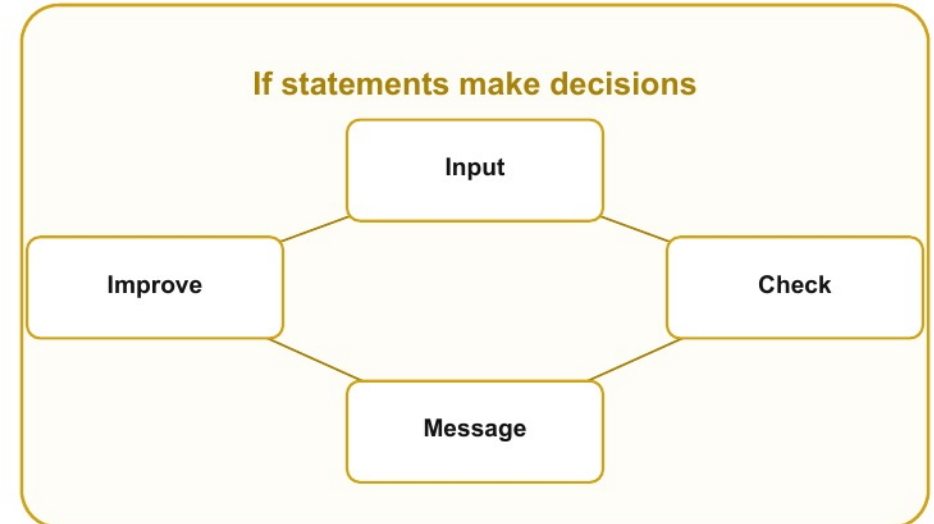
Try it next Run it once, change one value, then run it again.

Module 5: Example

See the idea running

```
sample run

>>> Enter a password: python
Weak password: too short.
>>> Enter a password: PythonCourse
Stronger password.
```



What happened

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

Module 5: Workbook

Write, predict, change, test

Task 1

Change the minimum password length.

Task 2

Add a message for empty passwords.

Task 3

Test three different passwords.

Answer guide hint

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

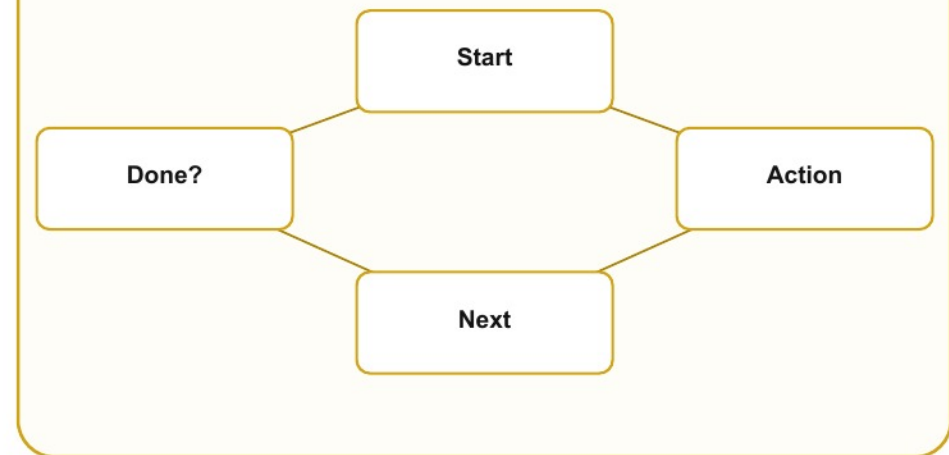
Module 6: Loops

Repeating useful work

What this module teaches

- Use for loops
- Use while loops
- Repeat until a condition is met

Loops repeat work



Plain English idea

A loop repeats work automatically.

Module 6: Program

Useful working program

This program builds the countdown timer. Type it, run it, then change one small part.

FILE: countdown_timer.py

```
import time

seconds = int(input("How many seconds? "))

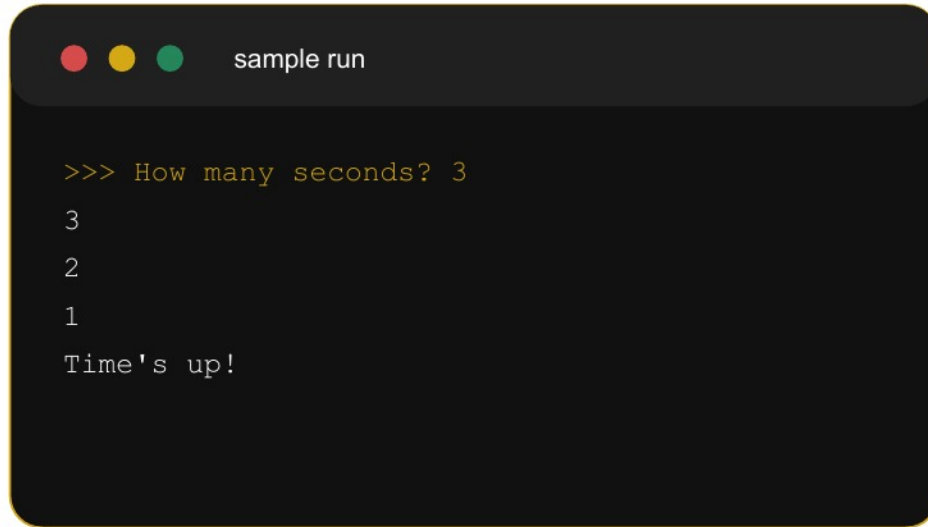
for number in range(seconds, 0, -1):
    print(number)
    time.sleep(1)

print("Time's up!")
```

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

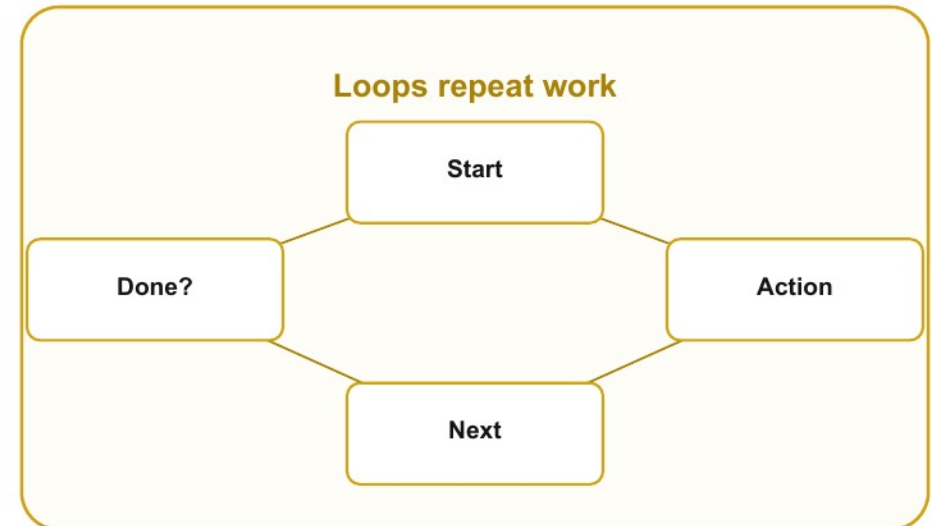
Module 6: Example

See the idea running



```
sample run

>>> How many seconds? 3
3
2
1
Time's up!
```



What happened

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

Module 6: Workbook

Write, predict, change, test

Task 1

Change the final message.

Task 2

Count from 1 to 10.

Task 3

Print only even numbers.

Answer guide hint

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

Module 7: Lists

Storing many items

What this module teaches

- Create a list
- Add items with `append()`
- Loop through list items

Lists store many things

Items

Index

Loop

Print

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

Plain English idea

A list holds many items in order.

Module 7: Program

Useful working program

This program builds the shopping list maker. Type it, run it, then change one small part.

FILE: shopping_list_maker.py

```
shopping = []

while True:
    item = input("Add item, or type done: ")

    if item.lower() == "done":
        break

    shopping.append(item)

print("Your shopping list:")
for item in shopping:
    print("-", item)
```

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

Module 7: Example

See the idea running

sample run

```
>>> Add item, or type done: milk
>>> Add item, or type done: bread
Your shopping list:
- milk
- bread
```

Lists store many things

Items

Index

Loop

Print

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

What happened

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

Module 7: Workbook

Write, predict, change, test

Task 1

Add three items.

Task 2

Print item numbers.

Task 3

Change done to stop.

Answer guide hint

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

Module 8: Dictionaries

Labelled records

What this module teaches

- Create key and value pairs
- Read values by label
- Store structured information

Dictionaries store labelled facts

Name

Phone

Email

Print

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

Plain English idea

A dictionary stores labelled information.

Module 8: Program

Useful working program

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

FILE: contact_card.py

```
contact = {}

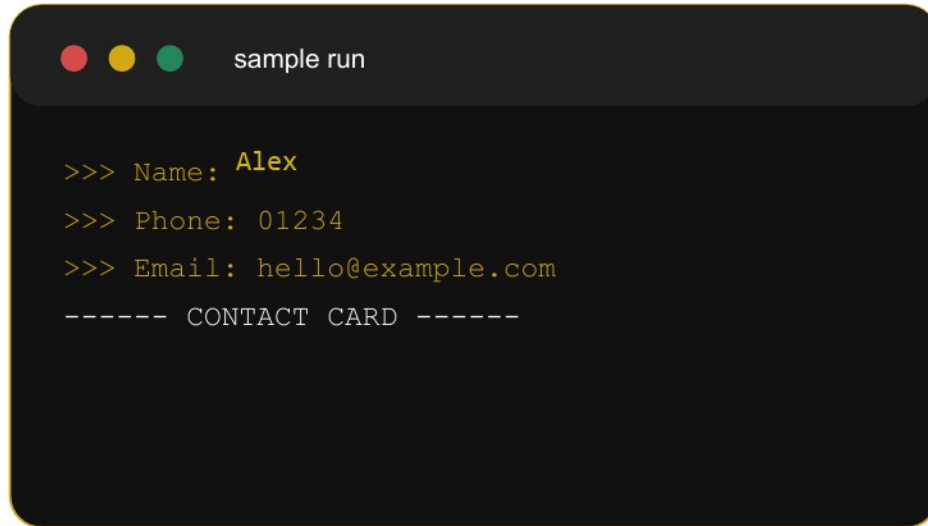
contact["name"] = input("Name: ")
contact["phone"] = input("Phone: ")
contact["email"] = input("Email: ")

print("----- CONTACT CARD -----")
print("Name:", contact["name"])
print("Phone:", contact["phone"])
print("Email:", contact["email"])
```

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

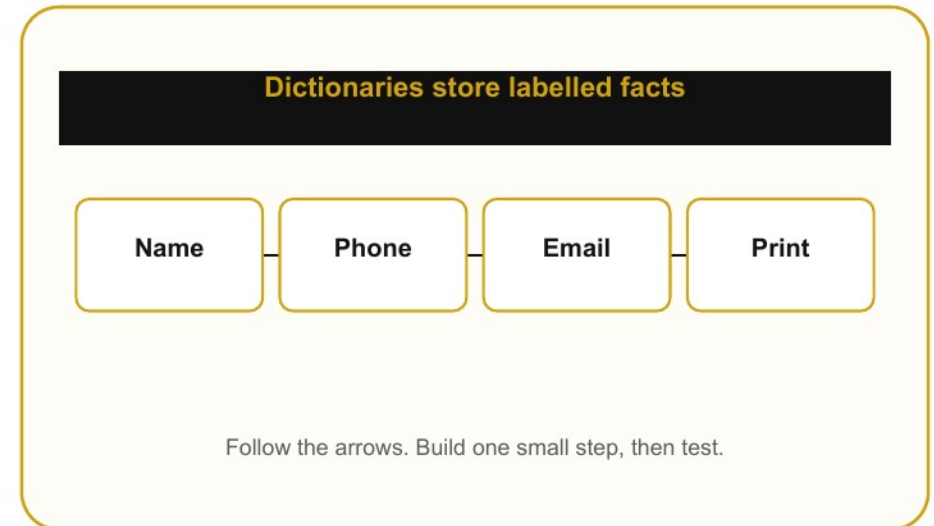
Module 8: Example

See the idea running



```
sample run

>>> Name: Alex
>>> Phone: 01234
>>> Email: hello@example.com
----- CONTACT CARD -----
```



What happened

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

Module 8: Workbook

Write, predict, change, test

Task 1

Add an address field.

Task 2

Add a notes field.

Task 3

Print one summary sentence.

Answer guide hint

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

Module 9: Functions

Reusable blocks

What this module teaches

- Define a function
- Send information into a function
- Return a result

Functions are machines

Input

Function

Return

Use

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

Plain English idea

A function is a named block of code that performs one clear job.

Module 9: Program

Useful working program

This program builds the tip calculator. Type it, run it, then change one small part.

FILE: tip_calculator.py

```
def calculate_tip(amount, percentage):  
    return amount * percentage / 100  
  
bill = float(input("Bill amount: £"))  
tip_percent = float(input("Tip percentage: "))  
  
tip = calculate_tip(bill, tip_percent)  
total = bill + tip  
  
print("Tip: £", round(tip, 2))  
print("Total: £", round(total, 2))
```

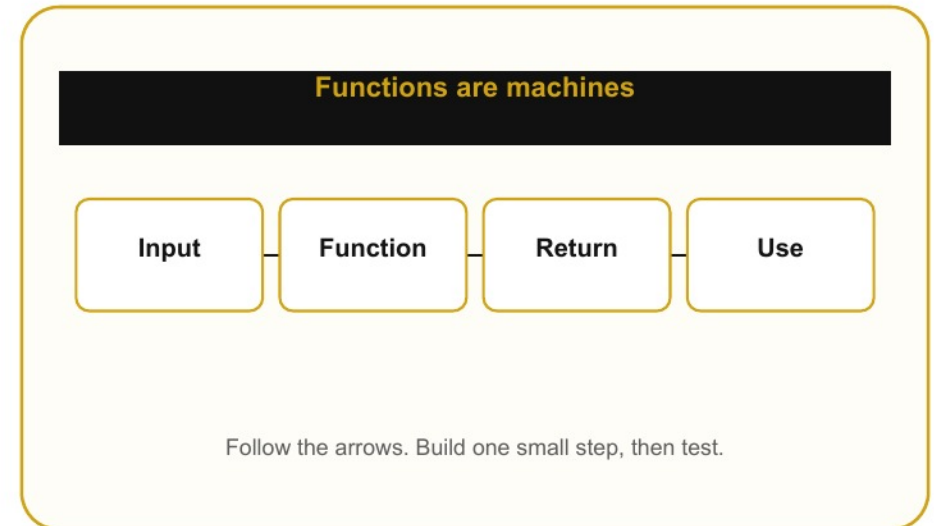
Try it next Run it once, change one value, then run it again.

Module 9: Example

See the idea running



```
>>> Bill amount: £20
>>> Tip percentage: 10
Tip: £ 2.0
Total: £ 22.0
```



What happened

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

Module 9: Workbook

Write, predict, change, test

Task 1

Rename the function.

Task 2

Change the percentage.

Task 3

Create a function that doubles a number.

Answer guide hint

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

Module 10: Files

Saving information

What this module teaches

- Open files safely with with
- Write notes to a file
- Read saved notes

Python writes to files

Input

Write

Save

Read

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

Plain English idea

Files let a program remember information after it closes.

Module 10: Program (1 of 2)

Useful working program

This program builds the simple notes app. Type it, run it, then change one small part.

FILE: simple_notes_app.py

```
while True:
    print("1. Add note")
    print("2. Read notes")
    print("3. Exit")

    choice = input("Choose an option: ")

    if choice == "1":
        note = input("Write your note: ")
        with open("my_notes.txt", "a") as file:
            file.write(note + "\n")
        print("Note saved.")

    elif choice == "2":
        try:
```

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

Module 10: Program (2 of 2)

Useful working program

This program builds the simple notes app. Type it, run it, then change one small part.

FILE: simple_notes_app.py

```
        with open("my_notes.txt", "r") as file:
            print(file.read())
    except FileNotFoundError:
        print("No notes found yet.")

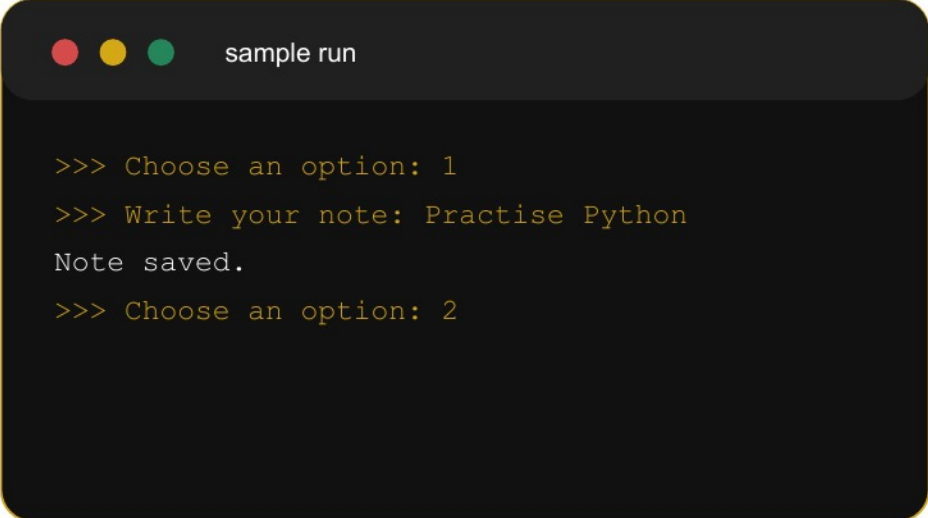
    elif choice == "3":
        print("Goodbye.")
        break

    else:
        print("Invalid choice.")
```

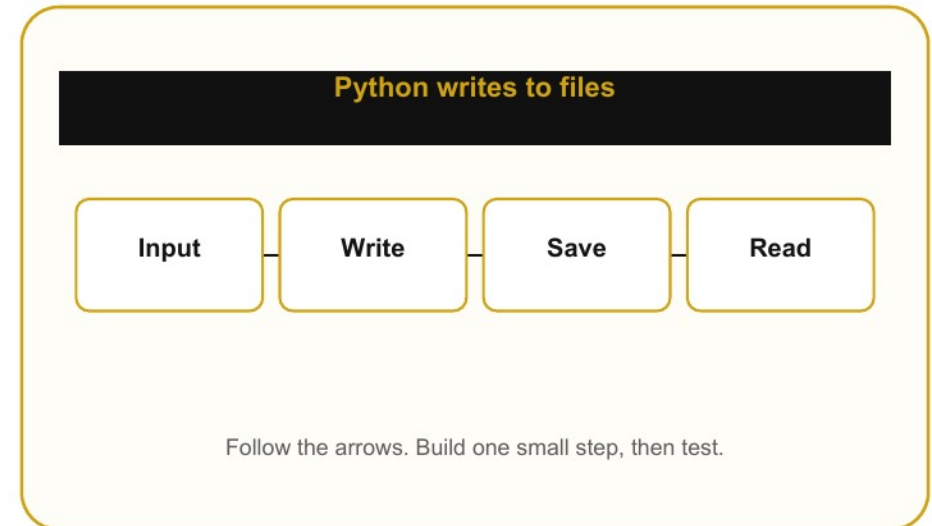
Try it next Run it once, change one value, then run it again.

Module 10: Example

See the idea running



```
>>> Choose an option: 1
>>> Write your note: Practise Python
Note saved.
>>> Choose an option: 2
```



What happened

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

Module 10: Workbook

Write, predict, change, test

Task 1

Change the file name.

Task 2

Add a date to each note.

Task 3

Change the menu text.

Answer guide hint

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

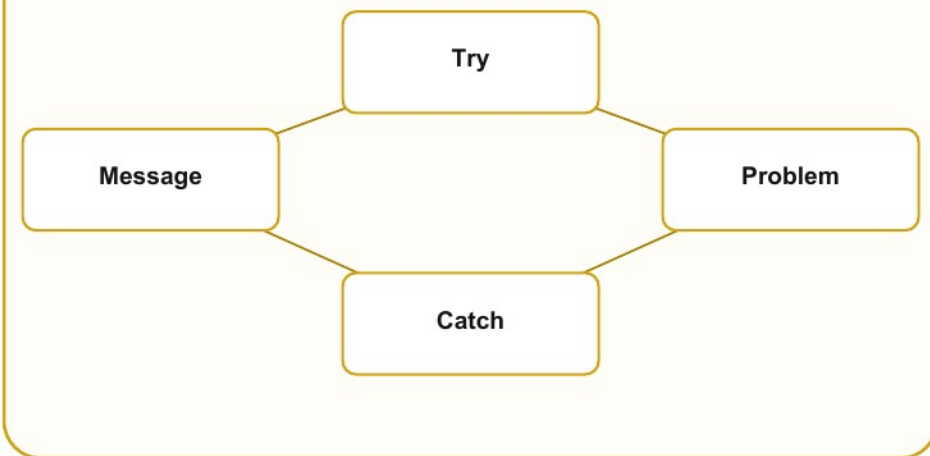
Module 11: Errors

Handling mistakes calmly

What this module teaches

- Use try and except
- Catch wrong number input
- Handle division by zero

Errors can be caught



Plain English idea

Good programs handle mistakes with helpful messages.

Module 11: Program (1 of 2)

Useful working program

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

FILE: safe_calculator.py

```
try:
    first = float(input("First number: "))
    second = float(input("Second number: "))
    operator = input("Choose +, -, *, /: ")

    if operator == "+":
        print(first + second)
    elif operator == "-":
        print(first - second)
    elif operator == "*":
        print(first * second)
    elif operator == "/":
        print(first / second)
    else:
        print("Unknown operator.")
```

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

Module 11: Program (2 of 2)

Useful working program

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

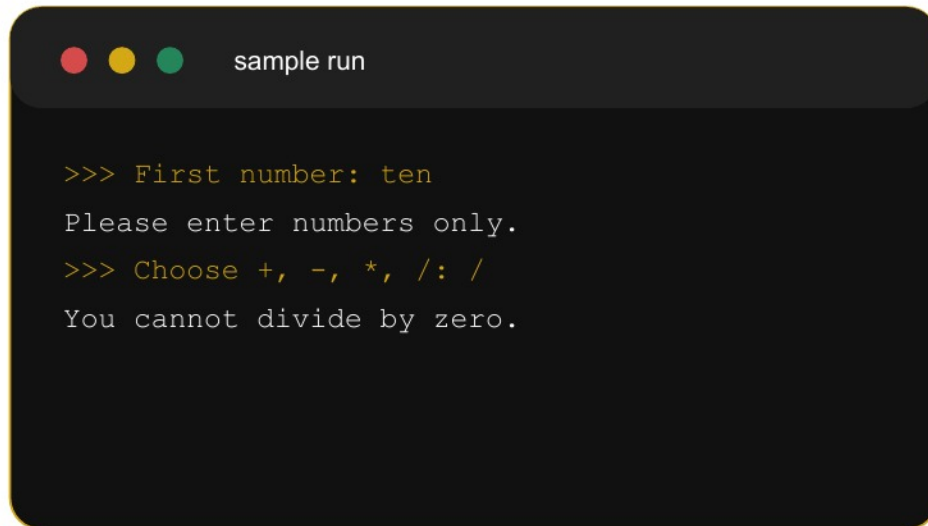
FILE: safe_calculator.py

```
except ValueError:
    print("Please enter numbers only.")
except ZeroDivisionError:
    print("You cannot divide by zero.")
```

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

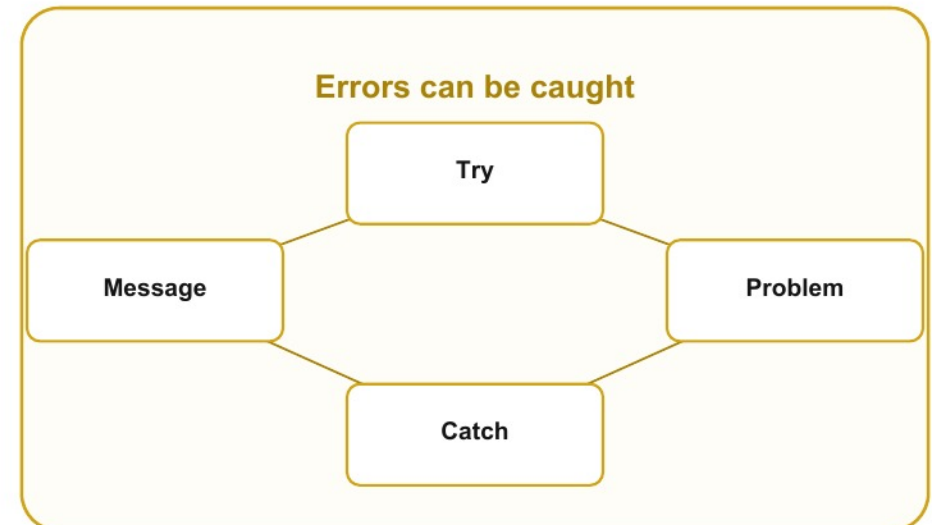
Module 11: Example

See the idea running



```
sample run

>>> First number: ten
Please enter numbers only.
>>> Choose +, -, *, /: +
You cannot divide by zero.
```



What happened

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

Module 11: Workbook

Write, predict, change, test

Task 1

Add a message for unknown operators.

Task 2

Test division by zero.

Task 3

Add another operation.

Answer guide hint

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

Module 12: Final Build

Putting beginner skills together

What this module teaches

- Use functions for structure
- Create a menu program
- Combine notes and calculator tools

Final Project: Command Centre

Menu

Notes

Calculator

Exit

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

Plain English idea

A real beginner project combines menus, functions, choices, files and helpful messages.

Module 12: Program (1 of 4)

Useful working program

This program builds the beginner command centre. Type it, run it, then change one small part.

FILE: beginner_command_centre.py

```
def add_note():
    note = input("Write note: ")
    with open("command_notes.txt", "a") as file:
        file.write(note + "\n")
    print("Note saved.")

def read_notes():
    try:
        with open("command_notes.txt", "r") as file:
            print(file.read())
    except FileNotFoundError:
        print("No notes found.")
```

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

Module 12: Program (2 of 4)

Useful working program

This program builds the beginner command centre. Type it, run it, then change one small part.

FILE: beginner_command_centre.py

```
def calculator():
    try:
        a = float(input("First number: "))
        b = float(input("Second number: "))
        op = input("Operator + - * /: ")

        if op == "+":
            print(a + b)
        elif op == "-":
            print(a - b)
        elif op == "*":
            print(a * b)
        elif op == "/":
            print(a / b)
        else:
```

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

Module 12: Program (3 of 4)

Useful working program

This program builds the beginner command centre. Type it, run it, then change one small part.

FILE: beginner_command_centre.py

```
        print("Unknown operator.")
except ValueError:
    print("Please enter valid numbers.")
except ZeroDivisionError:
    print("Cannot divide by zero.")

while True:
    print("\n=== BEGINNER COMMAND CENTRE ===")
    print("1. Add note")
    print("2. Read notes")
    print("3. Calculator")
    print("4. Exit")

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

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

Module 12: Program (4 of 4)

Useful working program

This program builds the beginner command centre. Type it, run it, then change one small part.

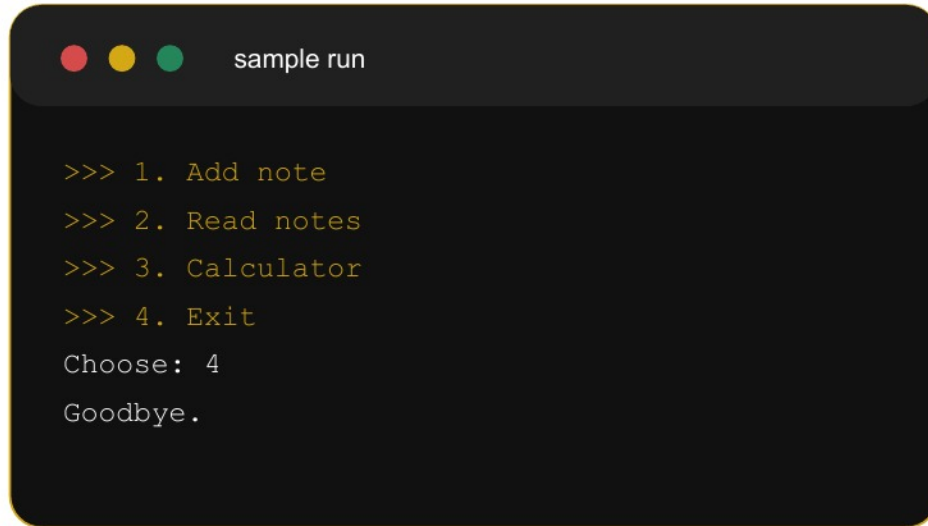
FILE: beginner_command_centre.py

```
if choice == "1":
    add_note()
elif choice == "2":
    read_notes()
elif choice == "3":
    calculator()
elif choice == "4":
    print("Goodbye.")
    break
else:
    print("Invalid choice.")
```

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

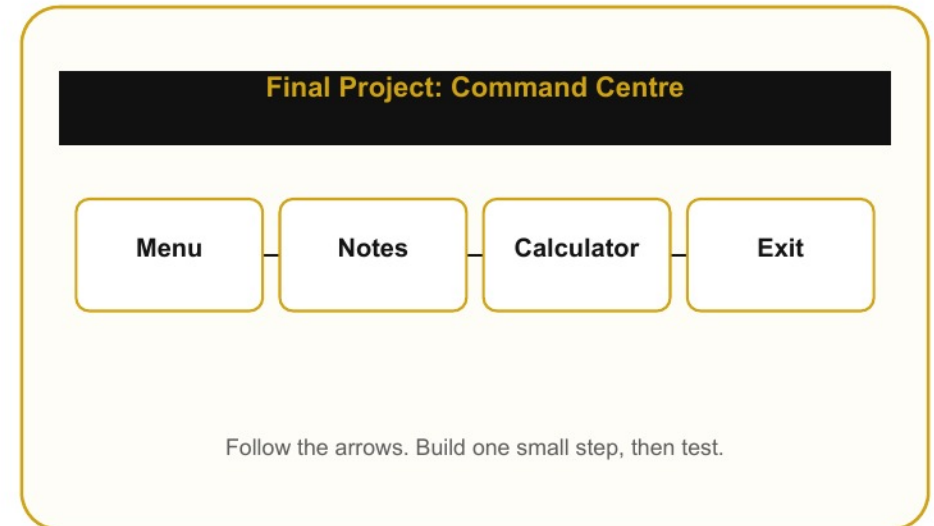
Module 12: Example

See the idea running



```
sample run

>>> 1. Add note
>>> 2. Read notes
>>> 3. Calculator
>>> 4. Exit
Choose: 4
Goodbye.
```



What happened

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

Module 12: Workbook

Write, predict, change, test

Task 1

Add a new menu option.

Task 2

Change the notes file name.

Task 3

Add one more calculator operation.

Answer guide hint

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

Useful Beginner Projects

Ready-to-adapt programs for real jobs

Budget Tracker

Build and adapt: Personal budget tracker.

Product Listing Generator

Build and adapt: Product listing generator.

To-Do List App

Build and adapt: To-do list app.

Contact Book

Build and adapt: Contact book.

How to use these

Run each project once, then make one small practical change.

Project 1: Budget Tracker

Practical starter tool

Budget report



What it builds

This project builds a Personal budget tracker that a learner can run and adapt.

Beginner goal

Keep the program simple, readable, and useful before adding extras.

Project 1: Program (1 of 2)

Useful working program

This program builds the personal budget tracker. Type it, run it, then change one small part.

FILE: budget_tracker.py

```
income = float(input("Monthly income: £"))

rent = float(input("Rent or mortgage: £"))
food = float(input("Food: £"))
bills = float(input("Bills: £"))
other = float(input("Other spending: £"))

total_spending = rent + food + bills + other
remaining = income - total_spending

print("----- BUDGET REPORT -----")
print("Income: £", round(income, 2))
print("Spending: £", round(total_spending, 2))
print("Remaining: £", round(remaining, 2))
```

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

Project 1: Program (2 of 2)

Useful working program

This program builds the personal budget tracker. Type it, run it, then change one small part.

FILE: budget_tracker.py

```
if remaining > 0:
    print("You are within budget.")
elif remaining == 0:
    print("You have used the full budget.")
else:
    print("You are over budget.")
```

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

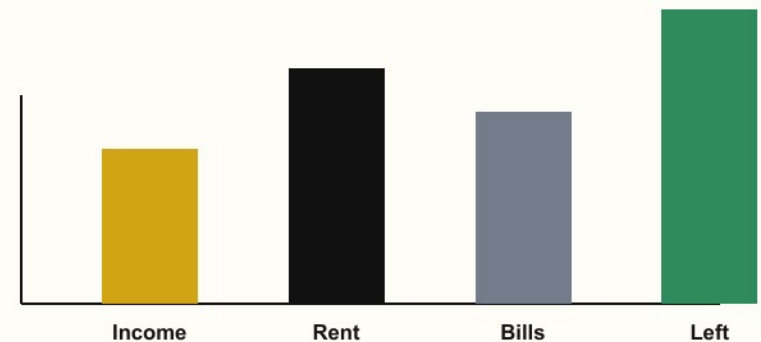
Project 1: Example

See the idea running

sample run

```
>>> Monthly income: £1000  
Spending: £ 750.0  
Remaining: £ 250.0  
You are within budget.
```

Budget report



What happened

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

Project 1: Workbook

Write, predict, change, test

Task 1

Add a savings category.

Task 2

Add travel spending.

Task 3

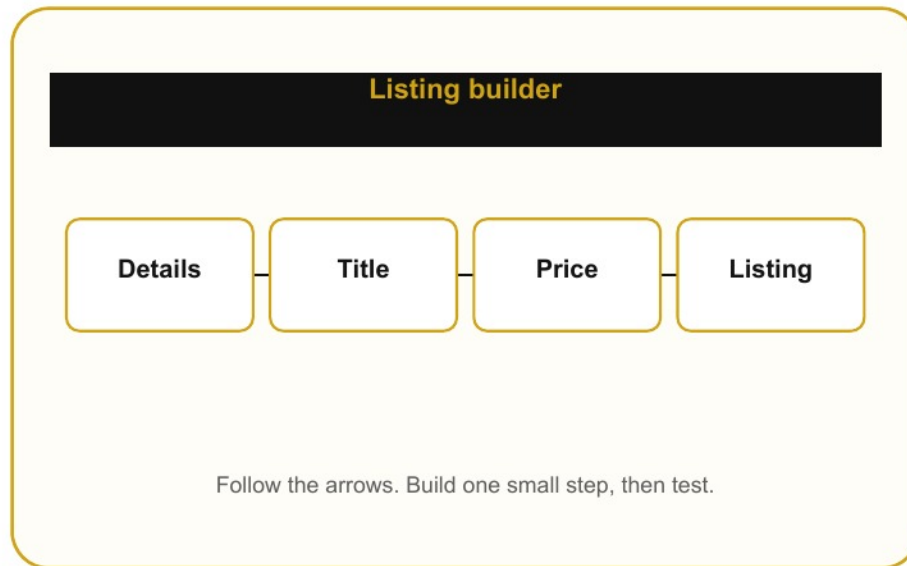
Change the printed report title.

Answer guide hint

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

Project 2: Product Listing Generator

Practical starter tool



What it builds

This project builds a Product listing generator that a learner can run and adapt.

Beginner goal

Keep the program simple, readable, and useful before adding extras.

Project 2: Program (1 of 2)

Useful working program

This program builds the product listing generator. Type it, run it, then change one small part.

FILE: product_listing_generator.py

```
product = input("Product name: ")
condition = input("Condition: ")
colour = input("Colour: ")
price = input("Price: £")
description = input("Short description: ")

listing_title = f"{condition.title()} {colour.title()} {product.title()}"

listing = f"""
----- PRODUCT LISTING -----

Title:
{listing_title}

Price:
```

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

Project 2: Program (2 of 2)

Useful working program

This program builds the product listing generator. Type it, run it, then change one small part.

FILE: product_listing_generator.py

```
£{price}

Description:
{description}

Condition:
{condition}

Colour:
{colour}

Thank you for looking.
"""

print(listing)
```

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

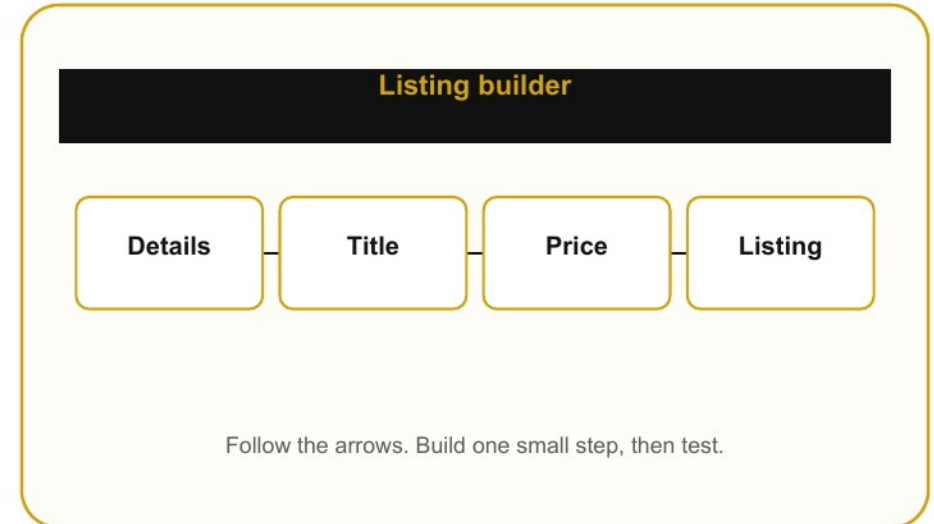
Project 2: Example

See the idea running



```
sample run

>>> Product name: mug
>>> Condition: new
Title:
New Blue Mug
```



What happened

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

Project 2: Workbook

Write, predict, change, test

Task 1

Add brand name.

Task 2

Add postage information.

Task 3

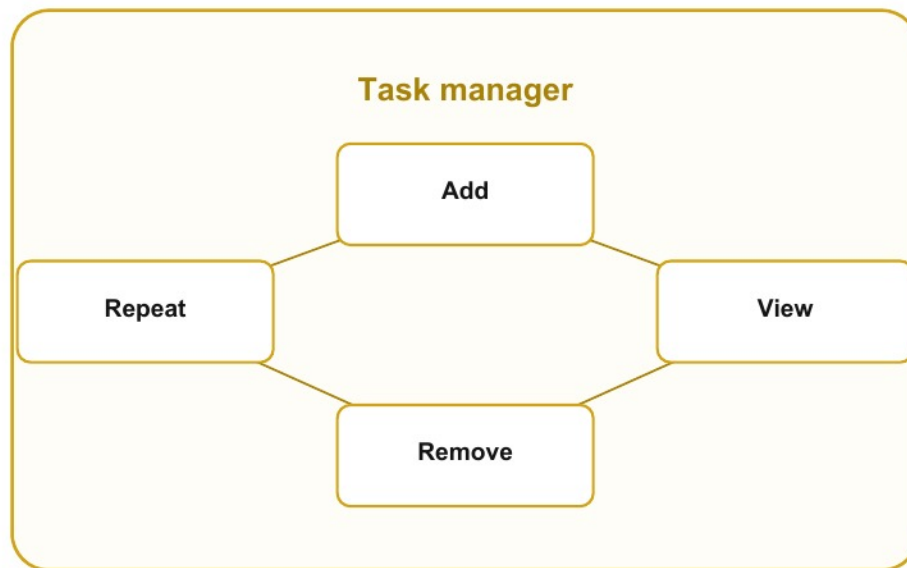
Change the thank-you line.

Answer guide hint

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

Project 3: To-Do List App

Practical starter tool



What it builds

This project builds a To-do list app that a learner can run and adapt.

Beginner goal

Keep the program simple, readable, and useful before adding extras.

Project 3: Program (1 of 3)

Useful working program

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

FILE: todo_list_app.py

```
tasks = []

while True:
    print("\n--- TO-DO LIST ---")
    print("1. Add task")
    print("2. View tasks")
    print("3. Remove task")
    print("4. Exit")

    choice = input("Choose: ")

    if choice == "1":
        task = input("Enter task: ")
        tasks.append(task)
        print("Task added.")
```

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

Project 3: Program (2 of 3)

Useful working program

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

FILE: todo_list_app.py

```
elif choice == "2":
    if len(tasks) == 0:
        print("No tasks yet.")
    else:
        for index, task in enumerate(tasks, start=1):
            print(index, task)

elif choice == "3":
    for index, task in enumerate(tasks, start=1):
        print(index, task)

remove_number = int(input("Task number to remove: "))
tasks.pop(remove_number - 1)
print("Task removed.")
```

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

Project 3: Program (3 of 3)

Useful working program

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

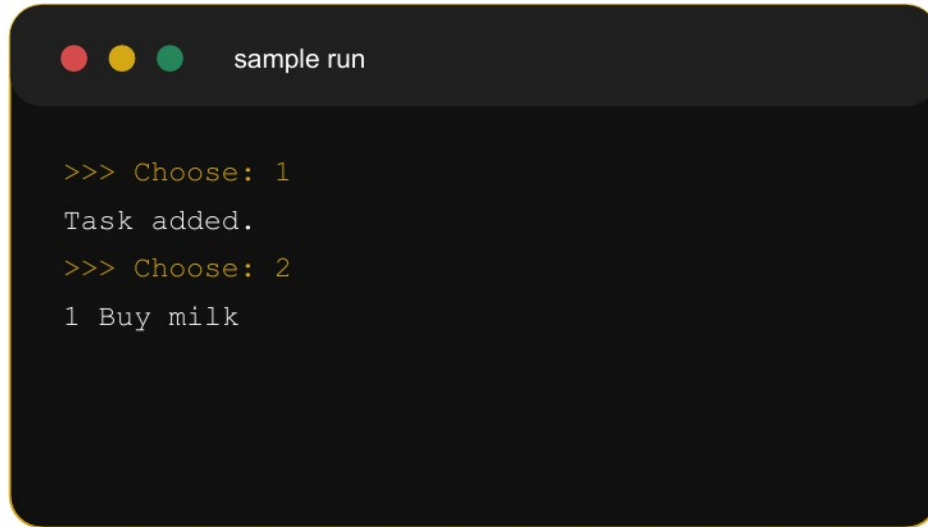
FILE: todo_list_app.py

```
elif choice == "4":  
    print("Goodbye.")  
    break  
  
else:  
    print("Invalid choice.")
```

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

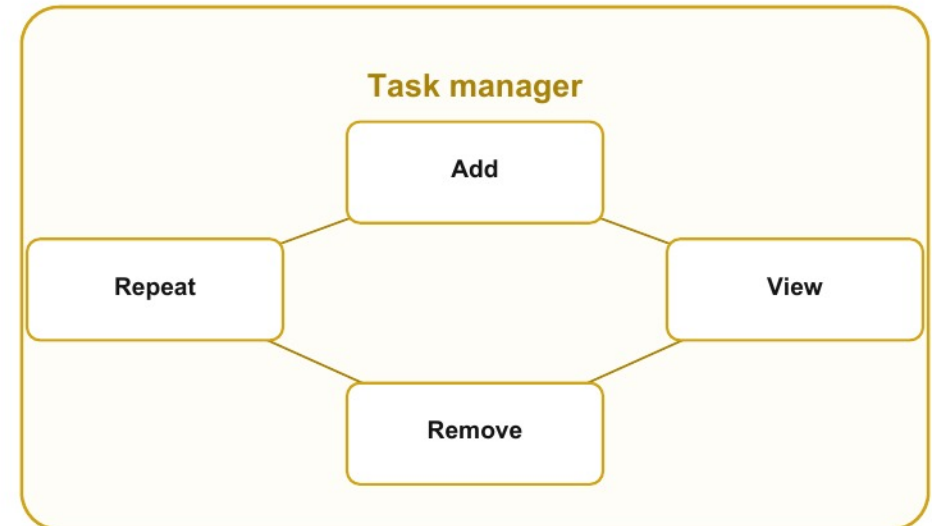
Project 3: Example

See the idea running



```
sample run

>>> Choose: 1
Task added.
>>> Choose: 2
1 Buy milk
```



What happened

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

Project 3: Workbook

Write, predict, change, test

Task 1

Add a completed message.

Task 2

Change the menu wording.

Task 3

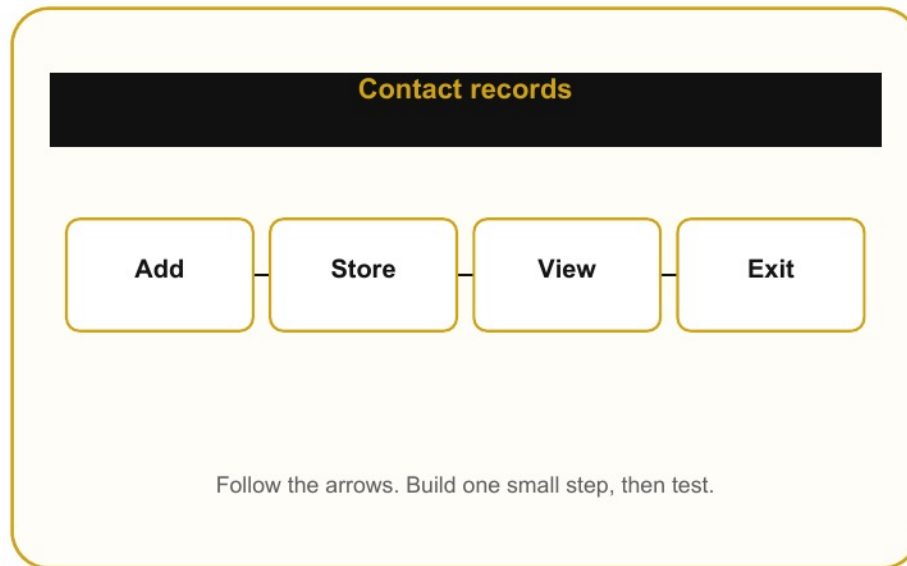
Add a clear-all option.

Answer guide hint

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

Project 4: Contact Book

Practical starter tool



What it builds

This project builds a Contact book that a learner can run and adapt.

Beginner goal

Keep the program simple, readable, and useful before adding extras.

Project 4: Program (1 of 3)

Useful working program

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

FILE: contact_book.py

```
contacts = []

while True:
    print("\n--- CONTACT BOOK ---")
    print("1. Add contact")
    print("2. View contacts")
    print("3. Exit")

    choice = input("Choose: ")

    if choice == "1":
        contact = {
            "name": input("Name: "),
            "phone": input("Phone: "),
            "email": input("Email: ")
```

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

Project 4: Program (2 of 3)

Useful working program

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

FILE: contact_book.py

```
    }
    contacts.append(contact)
    print("Contact added.")

elif choice == "2":
    for contact in contacts:
        print("-----")
        print("Name:", contact["name"])
        print("Phone:", contact["phone"])
        print("Email:", contact["email"])

elif choice == "3":
    print("Goodbye.")
    break
```

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

Project 4: Program (3 of 3)

Useful working program

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

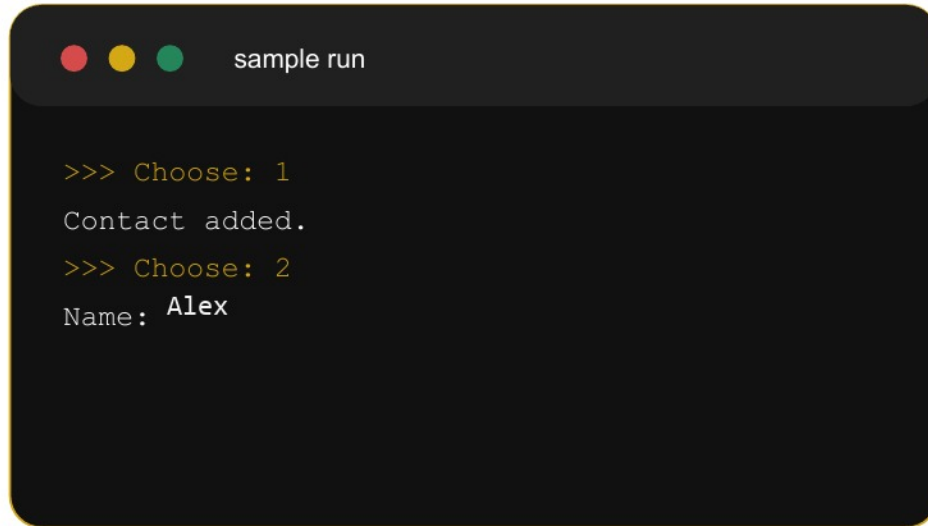
FILE: contact_book.py

```
else:  
    print("Invalid choice.")
```

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

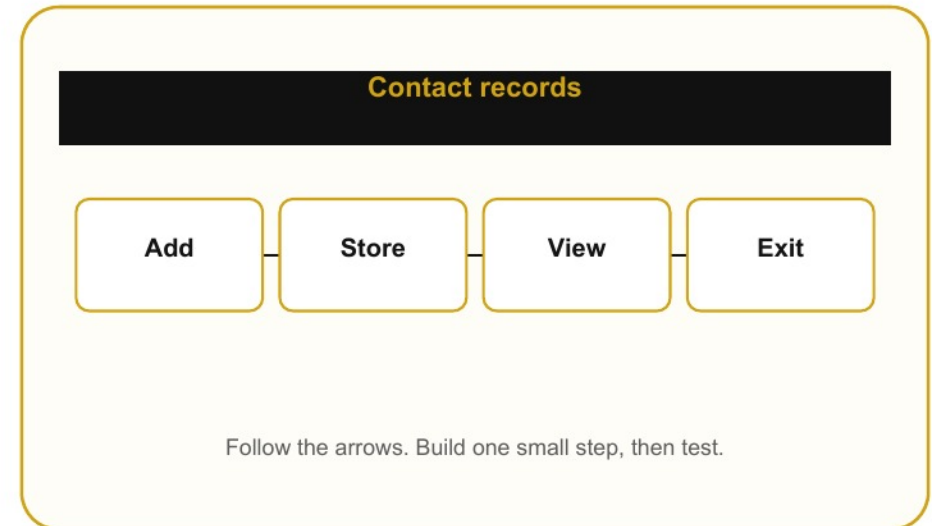
Project 4: Example

See the idea running



```
sample run

>>> Choose: 1
Contact added.
>>> Choose: 2
Name: Alex
```



What happened

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

Project 4: Workbook

Write, predict, change, test

Task 1

Add an address field.

Task 2

Add a search idea.

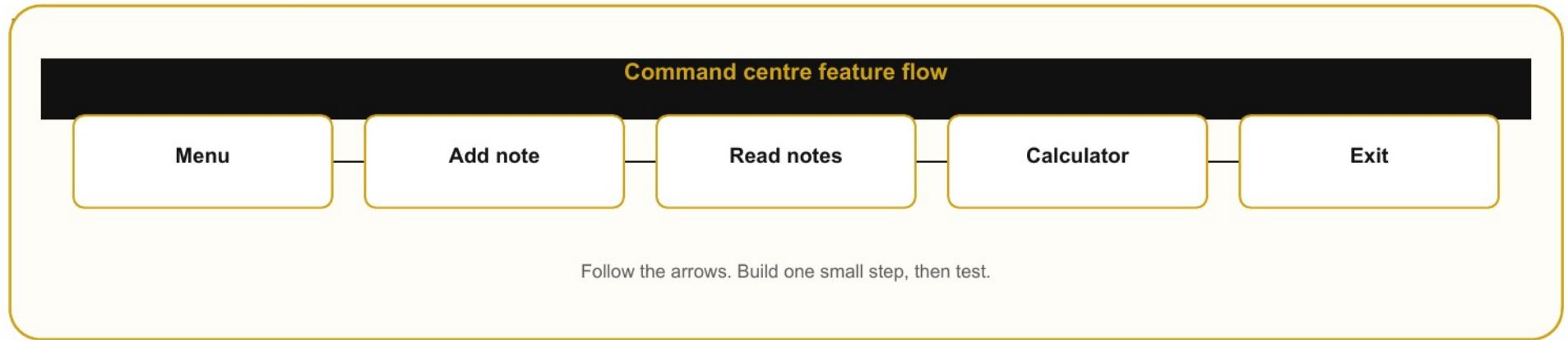
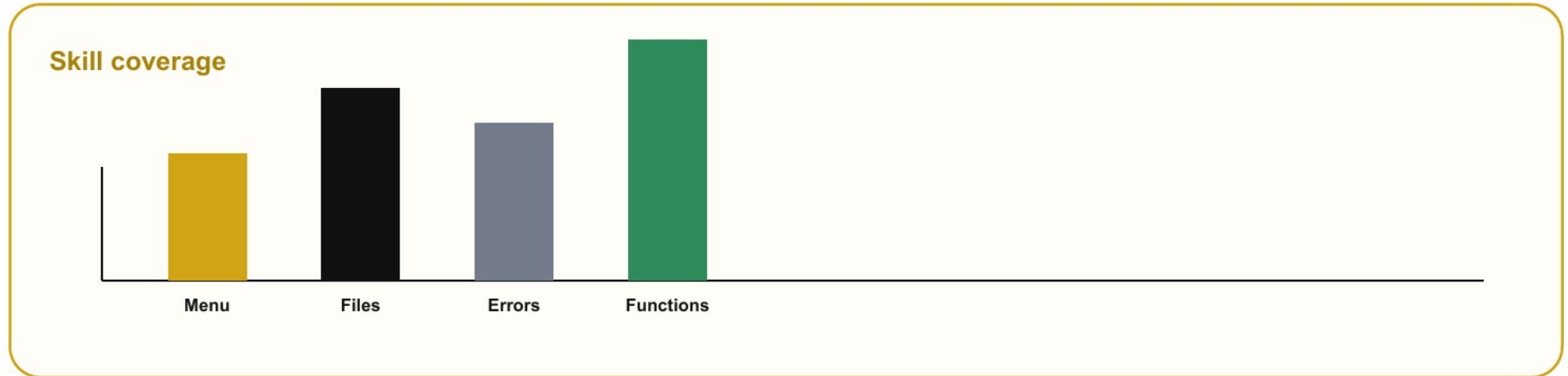
Task 3

Print contacts with numbers.

Answer guide hint

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

Final Project Build Map



Answer Guide

Suggested checks, not the only possible answers

Module 1

A good answer runs without errors and changes one

Module 2

A good answer runs without errors and changes one

Module 3

A good answer runs without errors and changes one

Module 4

A good answer runs without errors and changes one

Module 5

A good answer runs without errors and changes one

Module 6

A good answer runs without errors and changes one small detail.

Module 7

A stretch answer reuses the idea in another small

Module 8

A stretch answer reuses the idea in another small

Module 9

A stretch answer reuses the idea in another small

Module 10

A stretch answer reuses the idea in another small

Module 11

A stretch answer reuses the idea in another small

Module 12

A stretch answer reuses the idea in another small program.

Glossary

Plain-English terms

Program

A set of instructions for a computer.

Code

The written instructions in a programming language.

Variable

A labelled place to store information.

String

Text inside a program.

Loop

Code that repeats work.

Function

A named block of reusable code.

Completion Page

Volume 1 checkpoint

Well done

You have moved from zero programming knowledge to running useful beginner Python programs with input, variables, decisions, loops, lists, dictionaries, functions, files, and errors.

- I can run a Python program.
- I can use variables, strings and numbers.
- I can use decisions and loops.
- I can store data in lists and dictionaries.
- I can build small useful beginner tools.