



Growing Code

Python Fundamentals Through Garden Data

Summary: Discover Python's fundamental syntax and semantics by analyzing community garden data. Learn expressions, variables, functions, and control flow while contributing to sustainable community initiatives.

Version: 3.0

Contents

I	Foreword	2
II	AI Instructions	3
III	Introduction	5
IV	General Instructions	6
V	Exercise 0: Hello Garden	7
VI	Exercise 1: Garden Name	8
VII	Exercise 2: Garden Plot Area	9
VIII	Exercise 3: Harvest Total	10
IX	Exercise 4: Plant Age Check	11
X	Exercise 5: Water Reminder	12
XI	Exercise 6: Count to Harvest	13
XII	Exercise 7: Seed Inventory with Type Annotations	15
XIII	Helper Resources	17
XIV	Turn in and Submission	18

Chapter I

Foreword

In community gardens across the world, data tells stories of growth, collaboration, and impact.

From tracking harvest yields that feed local families to monitoring water usage that preserves precious resources, every measurement matters. Each data point represents someone's dedication—a volunteer's weekend hours, a child's first tomato, an elderly neighbor sharing decades of gardening wisdom.

Python, like these gardens, grows from simple seeds into something powerful and nourishing. Named after Monty Python's Flying Circus, it reminds us that learning should be joyful, accessible, and inclusive. Today, Python helps scientists understand climate change, enables communities to optimize resource sharing, and empowers anyone to transform raw data into meaningful insights.

In this project, you'll discover Python's fundamental building blocks while analyzing real community garden data. You'll learn that programming, like gardening, is about nurturing growth—both in code and in the communities we serve.

Chapter II

AI Instructions

● Context

During your learning journey, AI can assist with many different tasks. Take the time to explore the various capabilities of AI tools and how they can support your work. However, always approach them with caution and critically assess the results. Whether it's code, documentation, ideas, or technical explanations, you can never be completely sure that your question was well-formed or that the generated content is accurate. Your peers are a valuable resource to help you avoid mistakes and blind spots.

● Main message

- 👉 Use AI to reduce repetitive or tedious tasks.
- 👉 Develop prompting skills — both coding and non-coding — that will benefit your future career.
- 👉 Learn how AI systems work to better anticipate and avoid common risks, biases, and ethical issues.
- 👉 Continue building both technical and power skills by working with your peers.
- 👉 Only use AI-generated content that you fully understand and can take responsibility for.

● Learner rules:

- You should take the time to explore AI tools and understand how they work, so you can use them ethically and reduce potential biases.
- You should reflect on your problem before prompting — this helps you write clearer, more detailed, and more relevant prompts using accurate vocabulary.
- You should develop the habit of systematically checking, reviewing, questioning, and testing anything generated by AI.
- You should always seek peer review — don't rely solely on your own validation.

● Phase outcomes:

- Develop both general-purpose and domain-specific prompting skills.
- Boost your productivity with effective use of AI tools.
- Continue strengthening computational thinking, problem-solving, adaptability, and collaboration.

● Comments and examples:

- You'll regularly encounter situations — exams, evaluations, and more — where you must demonstrate real understanding. Be prepared, keep building both your technical and interpersonal skills.
- Explaining your reasoning and debating with peers often reveals gaps in your understanding. Make peer learning a priority.
- AI tools often lack your specific context and tend to provide generic responses. Your peers, who share your environment, can offer more relevant and accurate insights.
- Where AI tends to generate the most likely answer, your peers can provide alternative perspectives and valuable nuance. Rely on them as a quality checkpoint.

✓ Good practice:

I ask AI: “How do I test a sorting function?” It gives me a few ideas. I try them out and review the results with a peer. We refine the approach together.

✗ Bad practice:

I ask AI to write a whole function, copy-paste it into my project. During peer-evaluation, I can't explain what it does or why. I lose credibility — and I fail my project.

✓ Good practice:

I use AI to help design a parser. Then I walk through the logic with a peer. We catch two bugs and rewrite it together — better, cleaner, and fully understood.

✗ Bad practice:

I let Copilot generate my code for a key part of my project. It compiles, but I can't explain how it handles pipes. During the evaluation, I fail to justify and I fail my project.

Chapter III

Introduction

Welcome to Growing Code!

In this project, you'll discover Python's core concepts through community garden scenarios. You'll work with practical exercises that introduce fundamental programming concepts in an engaging, real-world context.

Each exercise builds upon the previous one, helping you develop essential programming skills while working on meaningful tasks.



IMPORTANT: For each exercise, you must write **ONLY** a function (not a main program). Each file should contain only the requested function that handles input and output directly. Do not write `if __name__ == "__main__":` blocks, do not call the function directly in your file, do not write code outside functions.



HELPER TOOL: To help you test your exercises, there is a `main.py` file provided in the attachments. Copy this file to your working directory and run `python3 main.py` to test your functions easily. This helper will import and test your functions automatically.

Chapter IV

General Instructions

- Your functions must be written in Python 3.10+.
- Your code must respect the flake8 linter standards.
- Each exercise must be in its own file.
- Each file should contain only the requested function.
- Function names must match exactly what is requested.
- You don't need to handle input validation or error cases unless explicitly mentioned.
- For negative numbers or invalid inputs, the behaviour is undefined (you don't need to handle these cases).
- Type hints are optional but recommended for learning purposes in exercises 0 to 6. They are required for exercise 7.



Growing Code Specific Note: Since this is an introductory project focusing on basic Python syntax, you only need to submit your individual Python function files (e.g., `ft_hello_garden.py`, `ft_plot_area.py`, etc.). Advanced project management tools will be introduced in later projects.

Chapter V

Exercise 0: Hello Garden

	Exercise0
ft_hello_garden	
Directory: ex0/	
Files to Submit: ft_hello_garden.py	
Authorized: print()	

Welcome to your first Python function! Write a function named `ft_hello_garden` that simply displays a welcome message to the garden community.

```
>>> ft_hello_garden()
Hello, Garden Community!
```



Write only the function `ft_hello_garden()`, not a main program. The function should handle output directly.



The example above shows the result of the function inside the interactive Python interpreter. Do not try to replicate this example as is; use the `main.py` script to test your function instead.

Chapter VI

Exercise 1: Garden Name

	Exercise1
ft_garden_name	
Directory: ex1/	
Files to Submit: ft_garden_name.py	
Authorized: input(), print()	

Write a function named `ft_garden_name` that asks for a garden name, then displays it followed by a fixed status message.

```
>>> ft_garden_name()
Enter garden name: Community Garden
Garden: Community Garden
Status: Growing well!
```



Write only the function `ft_garden_name()`, not a main program. The function should handle input and output directly.



Note that "Status: Growing well!" is a fixed message that should always be displayed exactly as shown.

Chapter VII

Exercise 2: Garden Plot Area

	Exercise2
ft_plot_area	
Directory: <i>ex2/</i>	
Files to Submit: <code>ft_plot_area.py</code>	
Authorized: <code>input()</code> , <code>int()</code> , <code>print()</code>	

A community garden needs to know the area of a rectangular plot. Write a function named `ft_plot_area` that asks for length and width, then calculates and displays the area.

```
>>> ft_plot_area()
Enter length: 5
Enter width: 3
Plot area: 15
```



Write only the function `ft_plot_area()`, not a main program. The function should handle input and output directly.

Chapter VIII

Exercise 3: Harvest Total

	Exercise3
ft_harvest_total	
Directory: <i>ex3/</i>	
Files to Submit: <code>ft_harvest_total.py</code>	
Authorized: <code>input()</code> , <code>int()</code> , <code>print()</code>	

A gardener harvested vegetables on three different days. Write a function named `ft_harvest_total` that asks for the weight of each harvest and calculates the total.

```
>>> ft_harvest_total()
Day 1 harvest: 5
Day 2 harvest: 8
Day 3 harvest: 3
Total harvest: 16
```



Write only the function `ft_harvest_total()`, not a main program. The function should handle input and output directly.

Chapter IX

Exercise 4: Plant Age Check

	Exercise4
ft_plant_age	
Directory: <i>ex4/</i>	
Files to Submit: ft_plant_age.py	
Authorized: input(), int(), print()	

Write a function named `ft_plant_age` that asks for a plant's age in days and tells you whether it's ready to harvest (strictly more than 60 days) or not.

```
>>> ft_plant_age()
Enter plant age in days: 75
Plant is ready to harvest!
>>> ft_plant_age()
Enter plant age in days: 45
Plant needs more time to grow.
```



Write only the function `ft_plant_age()`, not a main program. The function should handle input and output directly.

Chapter X

Exercise 5: Water Reminder

	Exercise5
ft_water_reminder	
Directory: ex5/	
Files to Submit: ft_water_reminder.py	
Authorized: input(), int(), print()	

Write a function named `ft_water_reminder` that asks for the number of days since the last watering. If it has been more than 2 days, print "Water the plants!"; otherwise print "Plants are fine".

```
>>> ft_water_reminder()
Days since last watering: 4
Water the plants!
>>> ft_water_reminder()
Days since last watering: 1
Plants are fine
```



Write only the function `ft_water_reminder()`, not a main program. The function should handle input and output directly.

Chapter XI

Exercise 6: Count to Harvest

	Exercise6
ft_count_harvest	
Directory: <i>ex6/</i>	
Files to Submit: <code>ft_count_harvest_iterative.py</code> , <code>ft_count_harvest_recursive.py</code>	
Authorized: <code>input()</code> , <code>int()</code> , <code>print()</code> , <code>range()</code> , defining helper functions for recursion	

Write two functions named `ft_count_harvest_iterative` and `ft_count_harvest_recursive`. Both functions should count from 1 to a given number, printing each day until harvest time.

```
>>> ft_count_harvest_iterative()
Days until harvest: 5
Day 1
Day 2
Day 3
Day 4
Day 5
Harvest time!

>>> ft_count_harvest_recursive()
Days until harvest: 5
Day 1
Day 2
Day 3
Day 4
Day 5
Harvest time!
```



Write only the functions `ft_count_harvest_iterative()` and `ft_count_harvest_recursive()`, not a main program. The functions should handle input and output directly.



For the recursive version, various approaches are acceptable:

- Using a nested helper function inside your main function
- Using default parameter values
- Using a separate helper function called by your main function

Choose the approach that makes the most sense to you. Both recursive and iterative functions should produce identical output.

Chapter XII

Exercise 7: Seed Inventory with Type Annotations

	Exercise7
ft_seed_inventory	
Directory: <i>ex7/</i>	
Files to Submit: <code>ft_seed_inventory.py</code>	
Authorized: <code>print()</code> , string methods	

The garden coordinator needs to track seed inventory with precise data types. Write a function named `ft_seed_inventory` that manages seed packets, displaying information about different seed types and quantities.

```
>>> ft_seed_inventory("tomato", 15, "packets")
Tomato seeds: 15 packets available
>>> ft_seed_inventory("carrot", 8, "grams")
Carrot seeds: 8 grams total
>>> ft_seed_inventory("lettuce", 12, "area")
Lettuce seeds: covers 12 square meters
```

Your function signature must look like this:

```
def ft_seed_inventory(seed_type: str, quantity: int, unit: str) -> None:
```



Write only the function, not a main program. The function should handle input and output directly.



Note the capital letter for the seed type; you can use the method available on string objects.



Support these units: "packets" (packets available), "grams" (grams total), "area" (covers X square meters). For any other unit, print only "Unknown unit type" with no other information on the line.



As stated at the beginning of this document, you must now use Python type hints (also called "typing"). Check typing using `mypy`.

Chapter XIII

Helper Resources

To support your learning journey, we've provided a `main.py` file that helps you test your exercises easily.

Since you're just starting with Python and don't know how to write main functions yet, this helper will:

- Import and run your exercise functions automatically
- Show clear error messages if something goes wrong
- Let you test individual exercises or all at once
- Help you understand how Python imports work

To use the helper, simply run `python3 main.py` in your terminal and choose which exercise to test. Make sure your exercise files (like `ft_plot_area.py`) are in the same folder as `main.py`.



The helper file is designed to be readable and educational. Feel free to look at the code to understand how it works, but focus on writing your own exercise solutions first.



The helper file is for learning support only. Your submitted solutions should be your own work and demonstrate your understanding of the concepts.

Chapter XIV

Turn in and Submission

Turn in your assignment to your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double-check the names of your files to ensure they are correct.



During evaluation, you may be asked to explain your code, trace through execution, or modify your solutions. Make sure you understand every line you write.



You need to submit only the files requested by each exercise for this project. Focus on clean, readable code that demonstrates your understanding of Python fundamentals.