



# Cultivando Código

## Sistemas de Cosecha Orientados a Objetos

*Resumen: Construye un ecosistema digital de jardinería mientras descubres conceptos avanzados de Python. Crea herramientas para gestionar de forma eficiente huertos comunitarios mediante enfoques basados en datos.*

*Versión: 1.0*

# Índice general

I.	Prólogo	2
II.	Instrucciones sobre la IA	3
III.	Introducción	6
IV.	Instrucciones Generales	7
V.	Ejercicio 0: Plantando tu Primera Semilla	8
VI.	Ejercicio 1: Organizador de Datos de la Huerta	10
VII.	Ejercicio 2: Simulador de Crecimiento de Plantas	12
VIII.	Ejercicio 3: Fábrica de Plantas	14
IX.	Ejercicio 4: Sistema de Seguridad del Huerto	16
X.	Ejercicio 5: Tipos de Plantas Especializadas	18
XI.	Ejercicio 6: Plataforma Analítica del Huerto	20
XII.	Entrega y Evaluación	22

# Capítulo I

## Prólogo

En la era digital, hasta los huertos necesitan sistemas inteligentes.

Una persona jardinera sabia dijo una vez: “No puedes plantar flores y luego arrancarlas cada día para ver cómo van las raíces”. Lo mismo aplica al código: a veces el crecimiento más hermoso ocurre cuando confiamos en el proceso y dejamos que nuestros programas se desarrollen de forma natural, capa a capa.

Detrás de cada huerto comunitario próspero hay una red de conexiones. Las plantas no crecen aisladas; forman comunidades donde cada especie aporta algo único y a la vez comparte necesidades comunes. Algunas plantas protegen a otras de plagas, otras fijan nitrógeno en el suelo y otras proporcionan sombra a plantones delicados. Esta red interconectada de apoyo mutuo refleja el funcionamiento de los sistemas de software bien diseñados: componentes individuales con roles distintos que trabajan juntos en armonía.

Así como una persona experta en jardinería sabe que un suelo sano produce plantas sanas, quien programa con experiencia entiende que un código bien estructurado crece hasta convertirse en aplicaciones robustas. Descubrirás que organizar tu código es como planificar un huerto: algunos elementos necesitan protección, otros espacio para crecer, y los mejores sistemas emergen cuando todo tiene su lugar y propósito adecuados.

En este proyecto, cultivarás tus habilidades de programación mientras construyes herramientas que podrían ayudar de forma genuina a que los huertos comunitarios florezcan. Cada línea de código que escribas es una semilla plantada en el suelo digital de las posibilidades.

# Capítulo II

## Instrucciones sobre la IA

### ● Contexto

Durante tu proceso de aprendizaje, la IA puede ayudarte con muchas tareas diferentes. Tómate el tiempo necesario para explorar las diversas capacidades de las herramientas de IA y cómo pueden apoyarte con tu trabajo. Sin embargo, siempre debes abordarlas con precaución y evaluar de forma crítica los resultados. Ya sea código, documentación, ideas o explicaciones técnicas, nunca podrás saber con total certeza si tu pregunta está bien formulada o si el contenido generado es el adecuado. Las personas que te rodean son tu recurso más valioso para ayudarte a evitar errores y puntos ciegos.

### ● Mensaje principal:

- 👉 Utiliza la IA para reducir las tareas repetitivas o tediosas.
- 👉 Desarrolla habilidades de prompting, ya sea para programación o para otros temas, que beneficiarán tu futura carrera.
- 👉 Aprende cómo funcionan los sistemas de IA para anticipar de forma eficiente y evitar los riesgos comunes, sesgos y problemas éticos.
- 👉 Sigue trabajando con tus compañeros para desarrollar tanto habilidades técnicas como habilidades transversales.
- 👉 Utiliza únicamente contenido generado por IA que entiendas completamente y del cual puedas responsabilizarte.

### ● Reglas para estudiantes:

- Debes tomarte el tiempo necesario para explorar las herramientas de IA y comprender cómo funcionan, para poder utilizarlas de manera ética y reducir los sesgos potenciales.
- Debes reflexionar sobre tu problema antes de dar instrucciones a la IA. Esto te ayuda a escribir preguntas, instrucciones o conjuntos de datos más claros, detalladas y relevantes utilizando un vocabulario preciso.

- Debes desarrollar el hábito de revisar, cuestionar y probar sistemáticamente cualquier contenido generado por la IA.
- Debes buscar siempre la revisión de otras personas, no te limites a confiar en tu propia validación.

## ● Resultados de esta etapa:

- Desarrollar habilidades de prompting tanto generales como de ámbito específico.
- Aumentar tu productividad con un uso eficaz de las herramientas de IA.
- Seguir fortaleciendo el pensamiento computacional, la resolución de problemas, la adaptabilidad y la colaboración.

## ● Comentarios y ejemplos:

- Ten en cuenta que la IA puede no tener la respuesta correcta porque esa respuesta no esté ni siquiera en Internet. Además, si te da soluciones incorrectas, intenta no insistir y busca ayuda entre las personas que te rodean. Vas a ahorrarte tiempo y vas a sumar en compresión.
- Vas a enfretarte con frecuencia a situaciones (como exámenes o evaluaciones) donde debes demostrar una comprensión real. Prepárate, sigue construyendo tanto tus habilidades técnicas como transversales.
- Explicar tu razonamiento y debatir con otras personas suele revelar lagunas en tu comprensión de un concepto. Prioriza el aprendizaje entre pares.
- Lo normal es que la herramienta de IA que utilices no conozca tu contexto específico (a menos que se lo indiques), así que te dará respuestas genéricas. Si buscas información más adecuada y más precisa en relación a tu entorno cercano, confía en el resto de estudiantes.
- Donde la IA tiende a generar la respuesta más probable, el resto de estudiantes puede proporcionar perspectivas alternativas y matices valiosos. Confía en la comunidad de 42 como un punto de control de calidad.

### ✓ Buenas prácticas:

Le pregunto a la IA: "¿Cómo pruebo una función de ordenación?" Me da algunas ideas. Las pruebo y reviso los resultados con otra persona. Refinamos el enfoque de manera conjunta.

### ✗ Bad practice:

Le pido a la IA que escriba una función completa, la copio y la pego en mi proyecto. Durante la evaluación entre pares, no puedo explicar qué hace ni por qué. Pierdo credibilidad. Suspendo mi proyecto.

**✓ Good practice:**

Utilizo la IA para ayudarme a diseñar un parser. Luego, reviso la lógica con otra persona. Encontramos dos errores y lo reescribimos juntos: mejor, más limpio y comprendiendo al 100%

**✗ Bad practice:**

Dejo que Copilot genere mi código para una parte clave de mi proyecto. Compila, pero no puedo explicar cómo maneja los pipes. Durante la evaluación, no puedo justificarlo y suspendo mi proyecto.

# Capítulo III

## Introducción

¡Te damos la bienvenida a Cultivando Código!

Haciendo uso de los fundamentos de Python que construiste en la primera actividad, ahora abordarás desafíos de programación más complejos, creando un sistema de gestión exhaustivo de datos de jardines. Este proyecto presenta conceptos avanzados que convierten a Python en una herramienta poderosa para modelar sistemas del mundo real.

Trabajarás en:

- Comprender cómo se estructuran y ejecutan los programas en Python.
- Organizar estructuras de datos complejas de forma eficiente.
- Crear componentes de código reutilizables.
- Construir sistemas capaces de adaptarse y ampliarse.
- Proteger la integridad de los datos en entornos colaborativos.
- Diseñar arquitecturas de software escalables.

Cada ejercicio se apoya en los anteriores, creando al final un frondoso y completo ecosistema digital.



**IMPORTANTE:** Este módulo comienza con la estructura básica de un programa en Python, progresando luego hacia la Programación Orientada a Objetos. Cada ejercicio debe contener las definiciones solicitadas y cualquier código requerido. Puedes incluir comprobaciones simples al final de cada archivo, construyendo tus propias pruebas haciendo uso de bloques `if __name__ == "__main__":`.

# Capítulo IV

## Instrucciones Generales

- Tu código debe estar escrito en Python 3.10+
- Tu código debe respetar los estándares de linter flake8
- Cada ejercicio debe estar en su propio archivo
- Utiliza convenciones de nombres adecuadas: clases en PascalCase, funciones y variables en snake\_case
- Incluye docstrings para funciones, clases y métodos
- Se recomiendan las anotaciones de tipo para todas las funciones y métodos
- No necesitas validar la entrada de datos a menos que se mencione explícitamente
- Concéntrate en demostrar los conceptos de programación con claridad
- Tus programas deben ejecutarse siempre sin errores



**Nota sobre el Ecosistema de Huerto Digital:** Este proyecto se centra en conceptos de programación en Python, comenzando por la estructura básica de un programa y avanzando hacia la Programación Orientada a Objetos. Cada ejercicio introduce nuevas funcionalidades al mismo tiempo que se construye un sistema coherente de gestión de huertos. Los ejercicios posteriores reutilizarán conceptos de los anteriores.

# Capítulo V

## Ejercicio 0: Plantando tu Primera Semilla

	Ejercicio: 0
	ft_garden_intro
	Directorio de entrega: <i>ex0/</i>
	Archivos a entregar: <b>ft_garden_intro.py</b>
	Funciones autorizadas: <code>print()</code> , <code>if __name__ == "__main__"</code>

Antes de poder construir sistemas complejos de gestión de huertos, necesitas entender cómo funcionan los programas en Python.

Todo programa de Python necesita un punto de entrada: un lugar donde empieza la ejecución. Igual que cuando plantas la primera semilla en un jardín, este paso marca el comienzo de tu recorrido en la programación.

Tu tarea es crear tu primer programa en Python, que mostrará información sobre una planta de tu huerto.

Requisitos:

- Crea un programa que se inicie inmediatamente después de ser ejecutado
- Utiliza el patrón `if __name__ == "__main__":`
- Almacena la información de la planta en variables simples (nombre, altura, edad)
- Muestra la información de la planta usando `print()`

Ejemplo:

```
$> python3 ft_garden_intro.py
==== Welcome to My Garden ====
Plant: Rose
Height: 25cm
Age: 30 days

==== End of Program ====
```



¡Esta es la primera semilla plantada en tu huerto de la programación con Python! Comprender cómo empiezan y se ejecutan los programas es fundamental antes de pasar a conceptos más complejos, como funciones y clases.



¿Qué ocurre si eliminas la línea `if __name__ == "__main__":`? Pruébalo y observa la diferencia. Además, ¿has notado que algunos archivos de Python comienzan con una línea especial que empieza por `#!?` Investiga qué hace esta línea "shebang" y por qué puede ser útil para que tus scripts se puedan ejecutar directamente.

# Capítulo VI

## Ejercicio 1: Organizador de Datos de la Huerta

	Ejercicio: 1
	ft_garden_data
	Directorio de entrega: <i>ex1/</i>
	Archivos a entregar: <i>ft_garden_data.py</i>
	Funciones autorizadas: <i>class, __init__, print()</i>

El huerto comunitario necesita llevar un registro de las múltiples plantas con su información. Debes almacenar y mostrar datos de varias plantas de manera eficiente.

Cada planta tiene:

- Un nombre
- Altura en centímetros
- Edad en días

Crea un programa que gestione los datos de, al menos, 3 plantas diferentes, y muestre su información de forma organizada.

Debes crear una clase `Plant` que sirva como base para cualquier planta, en lugar de manejar cada una de forma individual. Por ejemplo, cada planta podría tener nombre, altura y edad; aunque necesitarás encontrar una forma con sentido de organizar estos datos.

**Obligatorio:** Crea un modelo `Plant` que pueda representar cualquier planta con sus atributos.

Ejemplo:

```
$> python3 ft_garden_data.py  
== Garden Plant Registry ==  
Rose: 25cm, 30 days old  
Sunflower: 80cm, 45 days old  
Cactus: 15cm, 120 days old
```



¿Cómo estás almacenando actualmente la información de las plantas?  
¿Qué desafíos podrían surgir con más plantas?

# Capítulo VII

## Ejercicio 2: Simulador de Crecimiento de Plantas

	Ejercicio: 2
	ft_plant_growth
	Directorio de entrega: <i>ex2/</i>
	Archivos a entregar: <i>ft_plant_growth.py</i>
	Funciones autorizadas: <i>class, __init__, print()</i>

La persona que gestiona el jardín quiere simular el crecimiento de las plantas a través del tiempo. Debes seguir su cambio durante el transcurso del tiempo y proporcionar *operaciones* para modificar su estado.

Requisitos:

- Reutiliza tu `Plant class` del Ejercicio 1
- Las plantas deben poder `grow()` y `age()` — piensa en qué acciones puede realizar una planta
- Necesitarás una forma de `get_info()` sobre el estado actual de la planta
- Simula una semana de crecimiento para múltiples plantas
- Considera cómo deben cambiar las plantas a lo largo del tiempo mediante sus propias acciones

Tu programa debe mostrar a las plantas cambiando con el tiempo. Piensa en otorgar a tus plantas *comportamientos* que puedan ejecutar por sí mismas.

Ejemplo:

```
$> python3 ft_plant_growth.py  
== Day 1 ==  
Rose: 25cm, 30 days old  
== Day 7 ==  
Rose: 31cm, 36 days old  
Growth this week: +6cm
```



¿Cómo estás gestionando las operaciones sobre los datos de las plantas? ¿Tu código empieza a repetirse?

# Capítulo VIII

## Ejercicio 3: Fábrica de Plantas

	Ejercicio: 3
	ft_plant_factory
	Directorio de entrega: <i>ex3/</i>
	Archivos a entregar: <b>ft_plant_factory.py</b>
	Funciones autorizadas: <code>class</code> , <code>__init__</code> , <code>print()</code>

El centro de horticultura necesita crear muchas plantas rápidamente usando tu `Plant` `class` con diferentes valores iniciales. Debes optimizar el proceso de *creación* de plantas e *inicializarlas* correctamente.

Requisitos:

- Las plantas deben crearse con su información inicial (nombre, altura inicial, edad inicial)
- Cada planta debe estar lista para usarse inmediatamente tras su *construcción*
- Crea, al menos, 5 plantas diferentes con características variadas
- Muestra todas las plantas creadas organizadas en algún formato
- Piensa en cómo puedes agilizar el proceso de creación de plantas

Considera cómo podrías configurar eficientemente las plantas con sus valores iniciales. ¿Qué facilitaría crear muchas plantas?

Ejemplo:

```
$> python3 ft_plant_factory.py
==== Plant Factory Output ====
Created: Rose (25cm, 30 days)
Created: Oak (200cm, 365 days)
Created: Cactus (5cm, 90 days)
Created: Sunflower (80cm, 45 days)
Created: Fern (15cm, 120 days)

Total plants created: 5
```



¿Cómo estás inicializando actualmente tus plantas? ¿Existe una forma más eficiente de configurarlas?

# Capítulo IX

## Ejercicio 4: Sistema de Seguridad del Huerto

	Ejercicio: 4
	ft_garden_security
	Directorio de entrega: <i>ex4/</i>
	Archivos a entregar: <b>ft_garden_security.py</b>
	Funciones autorizadas: <code>class</code> , <code>__init__</code> , <code>def</code> , <code>print()</code> , <code>setter/getter (custom)</code>

La persona que gestiona el huerto está preocupada por la integridad de los datos. Dentro del grupo de voluntariado, algunos corrompieron accidentalmente datos de plantas al establecer valores no válidos (alturas negativas, edades imposibles). Debes crear un sistema seguro que *proteja* y *encapsule* los datos sensibles.

Requisitos:

- Crea una `SecurePlant` que proteja sus datos frente a la corrupción
- Proporciona formas controladas de modificar datos: `set_height()`, `set_age()`
- Proporciona formas seguras de acceder a los datos: `get_height()`, `get_age()`
- Asegura que la altura de la planta no pueda ser negativa incorporando validación
- Asegura que la edad de la planta no pueda ser negativa incorporando validación
- Imprime mensajes de error cuando se intenten valores no válidos
- Piensa en la *encapsulación*: proteger datos importantes ante el acceso directo

Considera cómo podrías crear un sistema que *valide* los datos antes de almacenarlos, garantizando su integridad.

Ejemplo:

```
$> python3 ft_garden_security.py
==== Garden Security System ====
Plant created: Rose
Height updated: 25cm [OK]
Age updated: 30 days [OK]

Invalid operation attempted: height -5cm [REJECTED]
Security: Negative height rejected

Current plant: Rose (25cm, 30 days)
```



¿Cómo puedes proteger tus datos para que no se corrompan accidentalmente? ¿Qué mecanismos podrías implementar?

# Capítulo X

## Ejercicio 5: Tipos de Plantas Especializadas

	Ejercicio: 5
	ft_plant_types
	Directorio de entrega: <i>ex5/</i>
	Archivos a entregar: <b>ft_plant_types.py</b>
	Funciones autorizadas: <code>class</code> , <code>__init__</code> , <code>super()</code> , <code>print()</code>

El huerto ahora necesita manejar distintos tipos de plantas: flores, árboles y hortalizas. Cada tipo tiene características únicas, pero *hereda* rasgos comunes de su categoría *padre*.

Requisitos:

- Comienza con una **Plant** base que tenga rasgos comunes (nombre, altura, edad)
- Crea tipos especializados: **Flower**, **Tree** y **Vegetable**
- Cada tipo especializado debe *heredar* las características básicas de la planta
- **Flower** necesita: atributo color y capacidad de `bloom()`
- **Tree** necesita: `trunk_diameter` y capacidad de `produce_shade()`
- **Vegetable** necesita: `harvest_season` y `nutritional_value`
- Al crear plantas especializadas, llama a la configuración del padre con `super().__init__()`
- Crea, al menos 2, instancias de cada tipo de planta
- Evita duplicar código común entre las plantas en los distintos tipos especializados

Piensa en las *relaciones de familia* en la naturaleza: ¿cómo comparten rasgos las especies al mismo tiempo que mantienen características únicas?

Ejemplo:

```
$> python3 ft_plant_types.py
 === Garden Plant Types ===

Rose (Flower): 25cm, 30 days, red color
Rose is blooming beautifully!

Oak (Tree): 500cm, 1825 days, 50cm diameter
Oak provides 78 square meters of shade

Tomato (Vegetable): 80cm, 90 days, summer harvest
Tomato is rich in vitamin C
```



Observa cuánto código podrías estar duplicando. Cada tipo de planta comparte características comunes, pero tiene particularidades. Piensa en los vínculos familiares en la naturaleza: ¿cómo comparten rasgos las especies?



¿Cómo estás gestionando las características comunes que tienen todos los tipos de planta? ¿Existe una forma de evitar repetir código creando un árbol genealógico de tipos relacionados?

# Capítulo XI

## Ejercicio 6: Plataforma Analítica del Huerto

	Ejercicio: 6
	ft_garden_analytics
	Directorio de entrega: <i>ex6/</i>
	Archivos a entregar: <i>ft_garden_analytics.py</i>
	Funciones autorizadas: <i>class</i> , <i>__init__</i> , <i>super()</i> , <i>print()</i> , <i>staticmethod()</i> , <i>classmethod()</i>

Construye una plataforma analítica de datos del huerto que procese y analice su información. Este sistema debe gestionar relaciones de datos complejas y proporcionar análisis detallados usando componentes *anidados* y *cadenas de herencia*.

Requisitos:

- Crea un **GardenManager** que pueda gestionar múltiples huertos
- Incluye una herramienta auxiliar **GardenStats** dentro del gestor para calcular estadísticas
- Construye un árbol genealógico de plantas: **Plant** → **FloweringPlant** → **PrizeFlower**
- Incluye un método **create\_garden\_network()** que funcione sobre el tipo del gestor
- Añade funciones utilitarias que no necesiten datos de un huerto específico
- Muestra distintos tipos de métodos: de instancia, de clase y funciones utilitarias
- Cada huerto debe llevar un registro de colecciones de plantas y estadísticas
- Usa tu herramienta auxiliar de estadísticas anidada para calcular analíticas
- Organiza todo dentro de estructuras adecuadas: evita funciones globales

Considera cómo podrías organizar sistemas complejos con múltiples componentes inter-actuando. ¿Qué ocurre cuando necesitas distintos tipos de métodos?

Ejemplo:

```
$> python3 ft_garden_analytics.py
 === Garden Management System Demo ===

Added Oak Tree to Alice's garden
Added Rose to Alice's garden
Added Sunflower to Alice's garden

Alice is helping all plants grow...
Oak Tree grew 1cm
Rose grew 1cm
Sunflower grew 1cm

 === Alice's Garden Report ===
Plants in garden:
- Oak Tree: 101cm
- Rose: 26cm, red flowers (blooming)
- Sunflower: 51cm, yellow flowers (blooming), Prize points: 10

Plants added: 3, Total growth: 3cm
Plant types: 1 regular, 1 flowering, 1 prize flowers

Height validation test: True
Garden scores - Alice: 218, Bob: 92
Total gardens managed: 2
```



Este ejercicio reúne todos los conceptos de programación que has aprendido a lo largo del proyecto. Se evaluará tu comprensión sobre la interacción de los distintos componentes y su organización dentro de un sistema complejo.



¿Cómo organizas sistemas complejos con múltiples componentes que interactúan? ¿Qué ocurre cuando necesitas distintos tipos de métodos que pertenecen a la clase en sí y no a instancias individuales?

# Capítulo XII

## Entrega y Evaluación

Entrega tu trabajo en tu repositorio Git como de costumbre. Solo se evaluará el contenido que esté dentro de tu repositorio durante la defensa. No dudes en comprobar que los nombres de tus archivos sean los correctos.



Durante la evaluación, se te puede pedir que expliques tu código, sigas el flujo ejecución o modifiques tus soluciones. Asegúrate de comprender cada línea que escribes.



Debes entregar únicamente los archivos solicitados por el enunciado de este proyecto. Concéntrate en escribir un código limpio, legible y que demuestre tu comprensión de los fundamentos de Python.