

01. Del primer programa a una aplicación útil

Fecha: _____ Sesión: _____ Tiempo: _____

Crea la carpeta programar-con-python y ejecuta diagnostico.py con los dos registros del ejemplo. Anota el resultado previsto y el obtenido. Repite con una lista vacía y con una cantidad igual a cero. Después añade un tercer artículo con dos unidades y comprueba que el total sube a diez. Redacta cinco requisitos del inventario usando la forma «al introducir... el programa debe...». Incluye un alta válida, un código repetido, una cantidad negativa, una búsqueda sin resultados y una importación con una fila incorrecta. No programes todavía esas operaciones. La entrega de hoy es una especificación pequeña que podrás recuperar en la última lección. Para comprobar tu trabajo pide a otra persona que lea los requisitos o léelos como si desconocieras tu idea. Si no puedes decidir si una prueba pasa o falla falta precisión. Sustituye expresiones como «debe ir bien» por resultados concretos. Escribe además dos funciones que decides dejar fuera de esta versión y explica qué complicación evitará cada exclusión.

Diagnóstico: resultado previsto y obtenido

Cinco requisitos comprobables

Dos funciones fuera del alcance

Observaciones

02. Preparar un entorno de trabajo separado

Fecha: _____ Sesión: _____ Tiempo: _____

Crea el entorno virtual y ejecuta `ubicacion.py` con su ruta explícita. Copia en el cuaderno la versión, la ruta del ejecutable y el directorio de trabajo. Cierra la terminal, vuelve a abrirla y repite la ejecución. Debes poder obtener el mismo intérprete sin depender de una activación que ya no existe. Crea las carpetas `practicas` y `proyecto`. Guarda una copia del diagnóstico anterior en `practicas` con un nombre que no coincida con un módulo estándar. Ejecútala desde la carpeta principal indicando su ruta. Explica por qué el directorio de trabajo sigue siendo el principal aunque el archivo esté dentro de `practicas`. Finalmente escribe cuatro instrucciones para preparar el curso en otro equipo. Incluye el requisito de Python, la creación de `.venv`, la ubicación de los archivos y el comando exacto de ejecución. No incluyas pasos de instalación de paquetes porque esta aplicación no los necesita.

Versión y ruta del intérprete

Carpeta de trabajo y ubicación del archivo

Instrucciones para reconstruir el entorno

Observaciones

03. Limpiar y comparar textos con criterio

Fecha: _____ Sesión: _____ Tiempo: _____

Guarda limpiar_codigo en practica_textos.py. Ejecuta la función con « cu001 », «LA-02», una cadena vacía, «CU 001» y un código de trece caracteres. Anota qué entradas se aceptan y cuáles deben producir ValueError. Para continuar con las pruebas puedes ejecutar cada caso por separado hasta estudiar las excepciones con más detalle. Crea una lista de tres nombres con tildes y espacios interiores. Busca «CUAD» mediante casefold y el operador in. Repite con una consulta vacía. Escribe por qué ese segundo resultado es coherente con la definición de búsqueda que hemos elegido. Diseña dos ejemplos de correcciones que no debería realizar el programa automáticamente. Uno debe distinguir cero de la letra O. El otro debe conservar una diferencia que consideres significativa en el nombre de un artículo. La práctica estará terminada cuando puedas explicar qué transformaciones admite tu función sin responder simplemente que limpia el texto.

Códigos: aceptados y rechazados

Búsquedas con mayúsculas y consulta vacía

Dos cambios automáticos que debes evitar

Observaciones

04. Filtrar, ordenar y agrupar colecciones

Fecha: _____ Sesión: _____ Tiempo: _____

Añade un artículo nuevo con cantidad cero. Obtén una lista con los códigos agotados y otra con los nombres de los artículos que tienen al menos dos unidades. Predice ambas salidas antes de ejecutar. Comprueba además que el orden de artículos no cambia después de llamar a `sorted`. Introduce deliberadamente un segundo registro con el código CU001. Adapta el recorrido con `views` para informar del duplicado. No lo soluciones convirtiendo toda la lista en un conjunto porque los diccionarios no se pueden usar directamente como elementos de un conjunto y además perderías contexto. Termina escribiendo dos resultados distintos: número de registros y suma de unidades. En el cuaderno explica por qué representan cosas diferentes. Si ordenas una lista filtrada y cambias después un diccionario compartido anota dónde aparece el cambio. Este pequeño experimento te preparará para el siguiente capítulo.

Filtros: entradas y resultados

Código duplicado detectado

Número de registros frente a unidades

Observaciones

05. Diseñar funciones que puedas probar

Fecha: _____ Sesión: _____ Tiempo: _____

Ejecuta `ajustar_cantidad` con variaciones de menos dos, cero y cinco. Anota tanto el resultado devuelto como el contenido del artículo original. Después prueba una variación de menos cuatro y comprueba que se comunica un error antes de crear un resultado inválido. Escribe el contrato de una futura función que sume las unidades de varios artículos. Debe recibir una colección y devolver un entero sin imprimir. Implementa la función con el bucle que ya conoces y úsala dos veces con colecciones diferentes. El resultado no debe depender del orden de las llamadas. Repite `agregar_nota` sin proporcionar una lista y después pasando una lista existente. Comprueba que la lista original permanece igual. En el cuaderno explica cuándo preferirías ese comportamiento y cuándo aceptarías modificar la entrada. Lo que debes entregar es una decisión razonada junto a una ejecución que demuestre que tu código la cumple.

Ajustes: resultado y original

Contrato y prueba del total de unidades

Listas independientes entre llamadas

Observaciones

06. Separar el código en módulos claros

Fecha: _____ Sesión: _____ Tiempo: _____

Crea `reglas_demo.py` y `principal_demo.py` exactamente en la misma carpeta. Ejecuta el segundo y comprueba que muestra Cuaderno. Después abre una sesión de Python desde esa carpeta e importa `principal_demo`. No debe imprimir el nombre ni solicitar datos. Anota qué diferencia produce la condición `__name__`. Añade a `reglas_demo` una función que valide una cantidad entera no negativa. Llámala desde `main` con un valor válido. No introduzcas `input` dentro del módulo de reglas. Explica cómo podrías utilizar esa misma función desde una prueba automática sin tener que escribir nada en la terminal. Dibuja en el cuaderno las dependencias previstas: `app` utiliza `almacen` e `intercambio`; ambos pueden utilizar `modelo`. Señala dónde colocarías un cambio de nombre permitido y dónde un cambio del separador CSV. La práctica estará completa cuando cada modificación tenga un lugar razonable sin necesidad de que `modelo` conozca la interfaz.

Ejecución directa frente a importación

Nueva función en el módulo de reglas

Responsabilidades de los cuatro módulos

Observaciones

07. Tratar errores sin esconder problemas

Fecha: _____ Sesión: _____ Tiempo: _____

Prueba leer_cantidad con «3», «0», «-2» y «tres». Anota qué casos devuelven un entero y cuáles comunican ValueError. Añade «2.5» y explica por qué una cantidad de unidades completas no admite esa escritura aunque represente un número. Escribe una pequeña función exterior que llame a leer_cantidad y muestre el mensaje de error. Después introduce de forma temporal una variable mal escrita fuera del bloque try. Comprueba que el fallo de programación no se convierte en «cantidad incorrecta». Retira ese defecto una vez entendido el resultado. Diseña un mensaje para un archivo ausente y otro para un registro CSV inválido. En cada caso indica una acción concreta que la persona pueda realizar. No prometas recuperación automática ni guardado seguro si todavía no has implementado esos mecanismos. Conserva los mensajes para revisarlos cuando construyas la interfaz final.

Cuatro entradas y sus excepciones

Fallo de programación que permanece visible

Mensajes para archivo y registro incorrectos

Observaciones

08. Convertir las comprobaciones en pruebas

Fecha: _____ Sesión: _____ Tiempo: _____

Crea los dos archivos y ejecuta las tres pruebas. Añade una cuarta para el texto «tres» y una quinta para «2.5». Ambas deben comprobar `ValueError`. Guarda la salida de ejecución o copia en el cuaderno el número de casos y su resultado. Introduce el defecto que rechaza cero y confirma que una prueba falla. Anota su nombre antes de restaurar la función. Después escribe una prueba para `limpiar_codigo` usando un código con espacios exteriores y minúsculas. La expectativa debe ser `CU001`. Elige un fallo que hayas cometido en una lección anterior y conviértelo en una prueba de regresión. Describe la entrada mínima que lo reproduce y el resultado correcto. La práctica termina cuando puedes ejecutar todas las comprobaciones de nuevo sin introducir datos a mano y sin depender del orden en que se ejecutan.

Ejecución inicial de las pruebas

Defecto deliberado y prueba que lo detecta

Nueva prueba de regresión

Observaciones

09. Trabajar con fechas sin hacer cuentas a mano

Fecha: _____ Sesión: _____ Tiempo: _____

Ejecuta el ejemplo y comprueba las dos salidas. Después calcula catorce días a partir del 25 de diciembre de 2026. Anota antes la fecha que esperas y verifica que el cambio de año se resuelve sin condiciones escritas por ti. Prueba leer_fecha con 2026-02-28, 2026-02-30 y una fecha con espacios exteriores. Añade una escritura que no cumpla el formato canónico. Registra cuáles se aceptan y qué regla incumple cada rechazo. No te limites a copiar el nombre de la excepción. Escribe una función que reciba fecha_actual y fecha_revision y devuelva los días de diferencia. Compruébala con una revisión futura, otra del mismo día y otra pasada. Explica por qué recibir la referencia como argumento permite repetir la prueba en cualquier momento sin que el resultado dependa del calendario real.

Cálculo que cruza de año

Fechas aceptadas y rechazadas

Tres diferencias de días con referencia fija

Observaciones

10. Calcular cantidades decimales con precisión

Fecha: _____ Sesión: _____ Tiempo: _____

Calcula tres unidades a 2,35 usando leer_importe y Decimal. Anota el resultado antes y después de presentar dos decimales. Repite construyendo un Decimal desde el texto «0.1» y desde el float 0.1. Compara las representaciones y explica de dónde procede la diferencia. Prueba «12,50», «12.50», «-1», «NaN» y «1.234,56». Las dos primeras se aceptan. Las otras deben rechazarse en las reglas de esta práctica. El último caso contiene un separador de miles que hemos decidido no admitir. Compara redondear dos valores 1.005 por separado con sumar primero y redondear después. Escribe ambas salidas y formula una regla explícita para una aplicación imaginaria. La práctica estará terminada cuando puedas explicar qué dato aceptas, cuándo redondeas y cómo lo muestras sin confundir esos tres pasos.

Importe calculado y presentación

Entradas decimales aceptadas y rechazadas

Redondeo por línea frente a redondeo del total

Observaciones

11. Controlar rutas y archivos con pathlib

Fecha: _____ Sesión: _____ Tiempo: _____

Crea prueba_archivos y escribe notas.txt con el modo x. Anota la ruta absoluta y confirma su contenido desde el editor. Vuelve a ejecutar la escritura y registra la excepción prevista. Comprueba que el texto original no se ha sustituido. Crea una segunda salida llamada revision.txt con una palabra que contenga ñ y otra con tilde. Léela usando UTF-8 y compara el texto. Después ejecuta el programa desde otra carpeta indicando la ruta del archivo Python. Observa qué ocurre si base se construye desde Path.cwd. Escribe una función guardar_nota que reciba ruta y texto. Debe utilizar el modo x y dejar que quien la llama decida cómo gestionar un archivo existente. Termina anotando qué cambia si el punto de referencia es el directorio de trabajo o la carpeta del programa. No borres ninguna carpeta para resolver la práctica.

Rutas de trabajo y salida

Creación repetida y archivo conservado

Texto con tildes leído correctamente

Observaciones

12. Intercambiar tablas mediante CSV

Fecha: _____ Sesión: _____ Tiempo: _____

Genera `articulos_demo.csv` y ábrelo como texto. Localiza el nombre que contiene punto y coma y observa cómo queda entrecomillado. Léelo con `DictReader` y comprueba que vuelve a aparecer como un único nombre. Anota el tipo que tiene la cantidad después de la lectura. Añade un tercer registro con un nombre que contenga una tilde y comillas. Escribe una nueva salida con otro nombre de archivo para conservar la primera. Comprueba el ciclo de escritura y lectura sin utilizar `split`. Si abres el documento con una hoja de cálculo revisa manualmente el separador elegido. Crea una copia del CSV y cambia su cabecera. La comprobación debe rechazarla. En el cuaderno redacta el contrato del formato en una sola lista: columnas, orden, separador, codificación y significado de cantidad. Esa lista será la referencia de las próximas importaciones.

Contrato del formato CSV

Campo con separador y comillas conservado

Cabecera incorrecta detectada

Observaciones

13. Validar una importación antes de guardar

Fecha: _____ Sesión: _____ Tiempo: _____

Prepara tres filas como diccionarios con las claves del CSV. Una debe usar CU001 y otra « cu001 ». Ejecuta `revisar_filas` y comprueba que detecta la repetición tras normalizar. Anota qué registro se señala y por qué no debes conservar automáticamente uno de los dos. Repite con una fila a la que falte un campo usando `None` como valor. Después utiliza tres filas correctas. Comprueba que el resultado es una lista nueva y que las entradas originales no han cambiado su código. El programa no debe crear ningún archivo durante esta práctica. Escribe el recorrido completo que tendrá la importación final: abrir, comprobar cabecera, validar registros, detectar duplicados internos e insertar en una transacción. Añade qué capa detectará un código que ya existe en la base. Esa distinción será la comprobación principal de tu ejercicio y se utilizará en la prueba final del curso.

Duplicados después de normalizar

Fila incompleta y mensaje producido

Fases de la importación y responsabilidades

Observaciones

14. Representar artículos con un modelo de datos

Fecha: _____ Sesión: _____ Tiempo: _____

Crea `modelo.py` y construye un artículo con « cu001 », « Cuaderno » y «3». Comprueba que el objeto resultante contiene CU001, Cuaderno y el entero 3. Intenta después asignar otra cantidad al mismo objeto y observa que la `dataclass` congelada lo impide. Prepara pruebas para nombre vacío, código con espacio interior, cantidad negativa, `True` y 1000001. Todos deben rechazarse. Añade los límites válidos cero y un millón. Escribe cuál de estas reglas responde al lenguaje y cuál has elegido para el proyecto. Compara `crear_articulo` con llamar directamente a `Articulo`. Anota por qué el constructor no debe utilizarse con entradas externas. Guarda las pruebas porque este archivo se incorporará sin cambios a la aplicación final. Si modificas una regla registra también qué comprobación deberás actualizar en el almacenamiento.

Artículo normalizado y tipos resultantes

Entradas inválidas y límites válidos

Constructor directo frente a función validada

Observaciones

15. Guardar el inventario en SQLite

Fecha: _____ Sesión: _____ Tiempo: _____

Crea una base en memoria y añade CU001 y LA001 mediante `crear_articulo` y `agregar_varios`. Comprueba que el recuento es dos. Cierra la conexión y explica por qué esos datos no reaparecen al crear otra base en memoria. En una nueva prueba añade primero CU001. Después intenta guardar un lote con CA001 seguido de otro CU001. Debe producirse `sqlite3.IntegrityError`. Consulta la tabla después del error y confirma que CA001 no quedó guardado. El registro inicial CU001 sí debe permanecer. Anota las responsabilidades de validación, restricción y transacción. No son tres nombres para lo mismo. La primera acepta entradas, la segunda impide ciertos estados en la tabla y la tercera agrupa escrituras. Conserva este caso porque será una prueba de aceptación central del proyecto final.

Inserción y recuento en memoria

Duplicado: registros antes y después

Validación, restricción y transacción

Observaciones

16. Consultar datos y presentar resultados claros

Fecha: _____ Sesión: _____ Tiempo: _____

Prepara una base en memoria con Cuaderno, Lápiz y Carpeta. Utiliza códigos que no estén insertados en orden alfabético. Llama a consultar sin texto y comprueba que el resultado aparece ordenado por código, no por el orden de inserción. Busca «CUAD», «LÁPIZ», «lapiz» y una palabra que no exista. Anota las coincidencias y explica qué resultados dependen de casefold y cuáles de conservar tildes. Repite con una consulta formada solo por espacios y verifica que aparecen todos los artículos. Añade una prueba automática para la búsqueda sin resultados. Debe esperar una lista vacía sin excepción. Después intenta consultar una conexión cerrada y observa que eso sí es un error. En el cuaderno explica por qué no deberías convertir ambos casos en el mismo mensaje ni ocultar un problema de conexión como si no hubiera artículos.

Orden de inserción frente a orden de consulta

Búsquedas con tildes y mayúsculas

Lista vacía frente a conexión incorrecta

Observaciones

17. Utilizar argumentos desde la terminal

Fecha: _____ Sesión: _____ Tiempo: _____

Ejecuta las ayudas general y de alta. Después interpreta un alta con un nombre de dos palabras usando comillas. Repite sin ellas y observa el error del analizador. En el cuaderno distingue este fallo estructural de un nombre vacío que el modelo rechazaría más adelante. Prueba --db antes del comando y --buscar después de listar. Anota dos órdenes completas que puedas reutilizar. Añade al analizador un comando llamado exportar con un argumento destino pero conserva la demostración sin escrituras. Comprueba que el valor aparece en el objeto de argumentos. Finalmente llama a parse_args con una lista desde otra función y verifica el valor de código. Explica por qué esa prueba no necesita escribir nada en la terminal. Guarda la práctica como referencia, ya que el capítulo diecinueve proporcionará el archivo app.py completo con las operaciones reales.

Órdenes válidas y ayuda

Comillas y errores de estructura

Prueba del analizador con lista explícita

Observaciones

18. Preparar copias y comprobar la recuperación

Fecha: _____ Sesión: _____ Tiempo: _____

Crea una base de prueba con dos artículos y cierra todas sus conexiones. Ejecuta `copia_cerrada.py` para obtener un destino nuevo. Anota las rutas absolutas del original y de la copia. Repite la orden y confirma que no sobrescribe la copia existente. Abre el respaldo en modo de solo lectura, ejecuta `quick_check` y cuenta los artículos. Consulta además uno de sus códigos conocidos. Registra los resultados. Si alguna comprobación falla conserva el original y repite la copia con otro nombre después de investigar la causa. Ensaya el uso de la copia como base de trabajo en una carpeta separada. Añade allí un artículo y comprueba que el original mantiene su recuento. Escribe instrucciones de recuperación de cinco pasos y señala expresamente que el procedimiento de copia exige que la base esté cerrada. No borres el original como parte del ejercicio.

Rutas y copia con conexiones cerradas

Integridad y contenido del respaldo

Ensayo de recuperación sin modificar el original

Observaciones

19. Integrar el inventario y el intercambio de archivos

Fecha: _____ Sesión: _____ Tiempo: _____

Completa los cuatro archivos y ejecuta la ayuda. Inicializa una base nueva, añade dos artículos y consulta uno por una parte de su nombre. Exporta a un destino nuevo e importa ese CSV en una segunda base vacía. Compara códigos, nombres y cantidades de ambas. Repite el alta de un código existente y comprueba que no se añade otro registro. Introduce una cantidad negativa y verifica que se rechaza antes de guardar. Intenta exportar sobre la salida ya creada y confirma que su contenido permanece igual. Anota cada orden, su resultado esperado y su resultado real. No des por terminado el ejercicio solo porque una ejecución funciona. La entrega de esta lección es el recorrido completo con los casos correctos y tres errores previstos, además de los archivos conservados para las pruebas de aceptación del siguiente capítulo.

Alta y consulta desde la terminal

Exportación e importación en segunda base

Duplicado, cantidad inválida y destino existente

Observaciones

20. Verificar el proyecto y ampliarlo con método

Fecha: _____ Sesión: _____ Tiempo: _____

Ejecuta las tres pruebas de integración y añade una cuarta que rechace un CSV con códigos repetidos después de normalizar. Añade también una prueba que intente exportar a un destino existente y confirme que conserva su contenido anterior. Usa una carpeta temporal en ambas. Completa los cinco requisitos de aceptación de la primera hoja y registra sus resultados. Ensaya la recuperación de una copia sin borrar la base original. Prepara las instrucciones mínimas para ejecutar el proyecto desde una carpeta nueva con el intérprete del entorno. Elige una sola mejora y escribe tres criterios de aceptación antes de tocar el código. Guarda una copia de la versión actual, realiza el cambio y repite las pruebas. Si una comprobación falla averigua si has cambiado una regla deliberadamente o si has introducido un defecto. Conserva en el cuaderno la decisión y la evidencia que la justifica.

Pruebas de integración y casos añadidos

Requisitos iniciales y recuperación comprobada

Una mejora con tres criterios de aceptación

Observaciones
