Automatiza en 3,2,1…
con Python

Antes de programar…
necesitamos entender la teoría

En toda disciplina física, se inicia con movimientos controlados, respiración consciente y técnica básica.

En programación **primero se aprende a pensar, estructurar y decidir.**

Nadie corre 42 km sin haber recorrido muchos más entrenando.
no empiezas a programar sin entender cómo diseñar, probar y mantener.

DISCLAIMER:
¡ALERTA MÁXIMA DE SESIÓN "ABURRIDA"! 🚨

Cero código, pura teoría.
Historia Python, Zen, venv, Git, commits, principios de diseño.

Prepara café EXTRA FUERTE

¡Juro por todos los indentados de Python que esta BRUTAL TORTURA TEÓRICA será el cimiento sobre el que construirás tu imperio de automatización!

Yo viendo que invite 15 personas a mi curso y llegaron 200

Me da anmsiedad

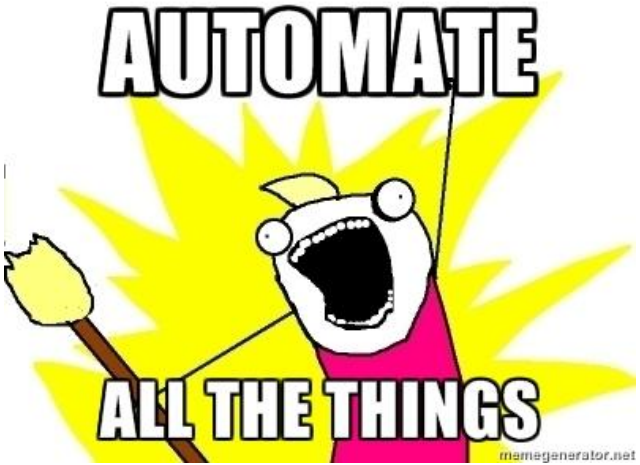
Sobre mi:

👨‍💻 Ingeniero en Computación
🐍 Desarrollador Backend Python
🏙️ CDMX 🇲🇽
✍️ Zurdo
🌟 Colaboro en:

https://www.meetup.com/python-mexico/events/307557668/
https://es.wikipedia.org/wiki/Usuario_discusi%C3%B3n:Pixelead0

Redes:
https://github.com/pixelead0
https://gist.github.com/pixelead0
https://pixelead0.blogspot.com/
https://www.linkedin.com/in/pixelead0/
https://medium.com/@pixelead0

Sobre mi:



Mi **superpoder** es **transformar hojas de cálculo** confusas **en archivos JSON** limpios, y hacerlo con código que alguien más pueda entender mañana sin querer prenderme fuego.

He sobrevivido a 11,394 bugs y sigo contando

Firme creyente de que cualquier problema puede resolverse con suficiente **café y Python**

Sobre mi:

Aprendí a manejar a mis 41 años,

<u>Nunca es tarde para iniciar.</u>

Tienes que aprender las reglas del juego.

Y luego tienes que jugar mejor que nadie.

Dianne Feinstein, la primera alcaldesa mujer de San Francisco, durante una entrevista en 1985 para la revista Cosmopolitan

Historia de Python

«Hace seis años, en diciembre de 1989, ***estaba buscando un proyecto*** *de programación como* ***hobby*** *que me mantuviera ocupado durante las semanas de* ***Navidad****.*

Mi oficina estaría cerrada y no tendría más que mi ordenador de casa a mano.

Decidí escribir un intérprete *para el nuevo lenguaje de scripting que había estado ideando recientemente: un descendiente de ABC* ***que gustaría a los hackers*** *de Unix/c.»*

—Guido Van Rossum (1996)

Historia

Python fue creado a finales de los años 80 por Guido van Rossum en el

Centro para las Matemáticas y la Informática (CWI) en los Países Bajos.

La primera versión pública (Python 0.9.0) se lanzó en febrero de 1991.

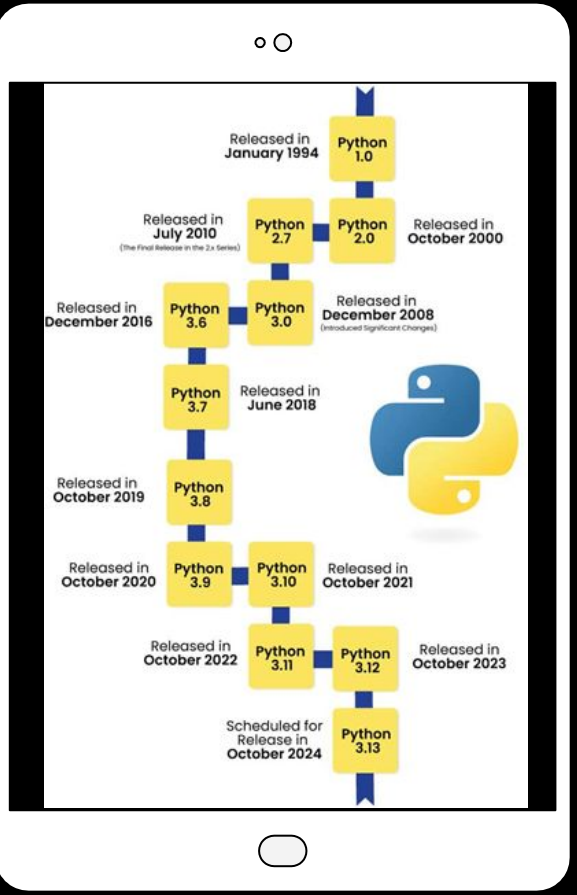
«Python» no proviene de la serpiente, sino del grupo humorístico británico **«Monty Python»**, del cual Guido era fan.

v1.0: Primera versión oficial.

v2.0: Introdujo listas por comprensión, recolección de basura y soporte Unicode.

v3.0: Cambio significativo no retrocompatible para corregir defectos fundamentales del diseño.

Versiones **3.x** continúan mejorando el lenguaje con características modernas.



CARACTERÍSTICAS

Baterías incluidas:

Python viene con una biblioteca estándar amplia y versátil.

Hay una sola forma de hacerlo:

A diferencia de otros lenguajes, Python suele favorecer una forma pythónica de resolver problemas.

Practicidad sobre pureza:

Python busca el equilibrio entre teoría y aplicabilidad práctica.

Ventajas de Python

Versatilidad

Un único lenguaje aplicable a múltiples dominios, desde desarrollo web hasta inteligencia artificial.

Facilidad

Sintaxis intuitiva que reduce la curva de aprendizaje y permite escribir soluciones rápidamente.

Ecosistema

Acceso a miles de bibliotecas especializadas que evitan reinventar la rueda.

Interoperabilidad

Capacidad para integrarse fácilmente con otros lenguajes y sistemas, permitiendo aprovechar código existente.

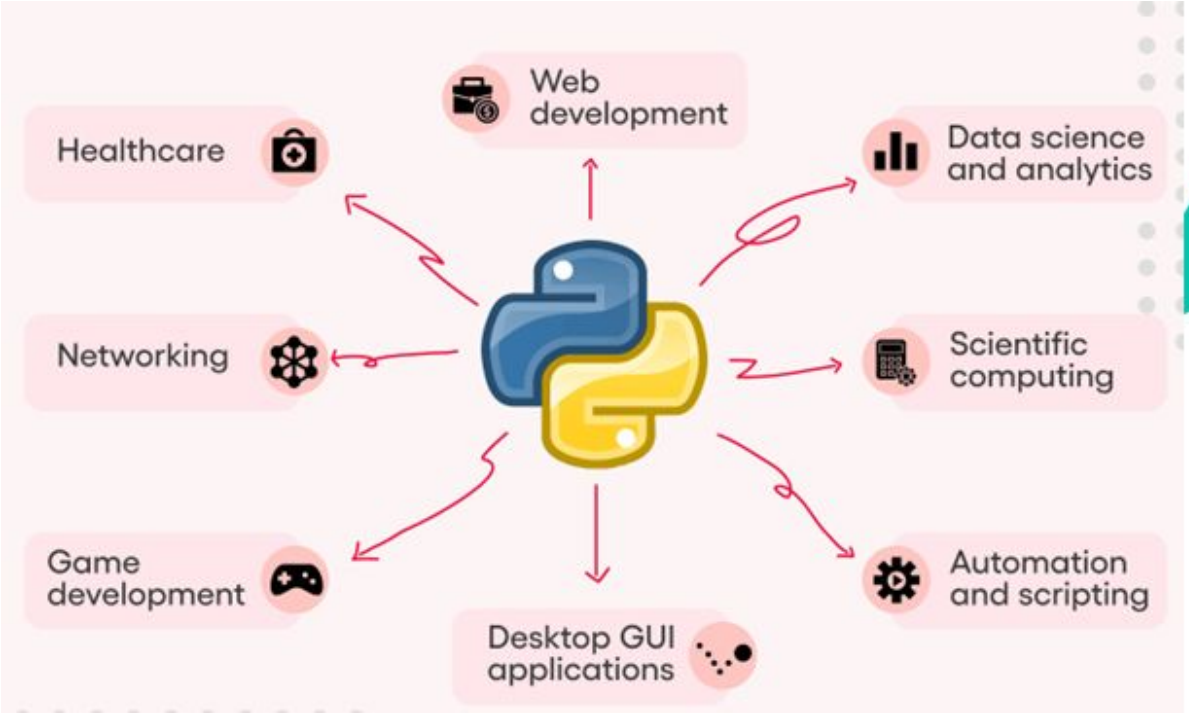
Comunidad

Soporte continuo mediante documentación extensa, foros y recursos educativos gratuitos.

Laboral

Soporte continuo mediante documentación extensa, foros y recursos educativos gratuitos.

Areas de aplicacion de Python



Limitaciones de Python

Limitaciones de Python

No tiene,
Python es perfecto

Limitaciones de Python

Rendimiento

Velocidad de ejecución inferior a lenguajes compilados, limitando su uso en aplicaciones de alto rendimiento computacional.

Desarrollo Movil

Escaso soporte para desarrollo de aplicaciones móviles nativas, requiriendo frameworks adicionales con funcionalidad limitada

Multithreading limitado

Presencia del Global Interpreter Lock (GIL) que impide la ejecución de múltiples hilos Python en paralelo, dificultando el aprovechamiento de sistemas multiprocesador.

Tipado dinámico

Mayor propensión a errores en tiempo de ejecución que lenguajes con tipado estático, aunque mitigable con anotaciones de tipo.

Consumo de Recursos

Mayor uso de memoria comparado con lenguajes de bajo nivel, dificultando su implementación en dispositivos con recursos limitados.

Limitaciones de Python

La mejor manera de aprender un idioma es hablar con nativos.

El chico que está aprendiendo Python.

Zen de python

La filosofía de Python está resumida en el "**Zen de Python**", un conjunto de 20 principios que guían el diseño del lenguaje y la forma de programar.

Se puede acceder a ellos ejecutando:

>>> import this

Ilustracion: https://hacktoon.com/posts/2016/the-zen-of-python-illustrated/

Hermoso es mejor que feo

Python promueve código legible y estéticamente agradable.

Ilustracion: https://hacktoon.com/posts/2016/the-zen-of-python-illustrated/

Explícito es mejor que implícito

El código debe ser claro en su intención, sin comportamientos ocultos.

Ilustracion: https://hacktoon.com/posts/2016/the-zen-of-python-illustrated/

Simple es mejor que complejo

Las soluciones simples son preferibles a las elaboradas.

Ilustracion: https://hacktoon.com/posts/2016/the-zen-of-python-illustrated/

Complejo es mejor que complicado

Si la complejidad es necesaria, debe estar bien estructurada, no caótica.

Ilustracion: https://hacktoon.com/posts/2016/the-zen-of-python-illustrated/

La legibilidad cuenta

Uno de los valores fundamentales: **el código debe ser fácil de leer.**

Ilustracion: https://hacktoon.com/posts/2016/the-zen-of-python-illustrated/

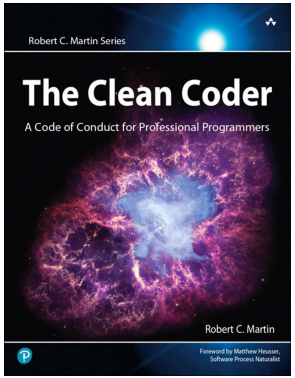
Zen de python

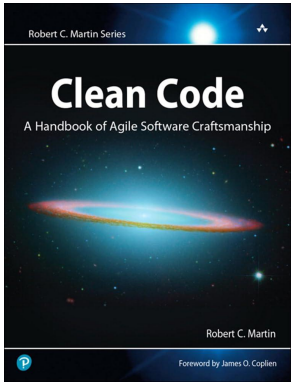
Consulta ejemplos:

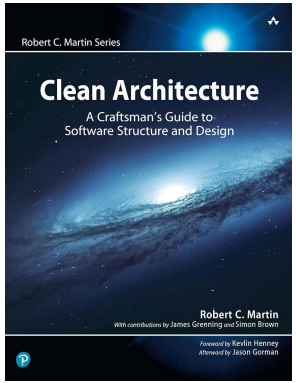
https://gist.github.com/evandrix/2030615

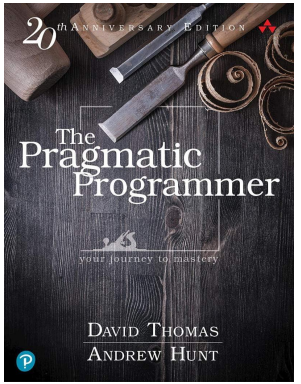
https://ai.gopubby.com/writing-good-python-code-the-zen-of-python-df7830b52195

Libros recomendados





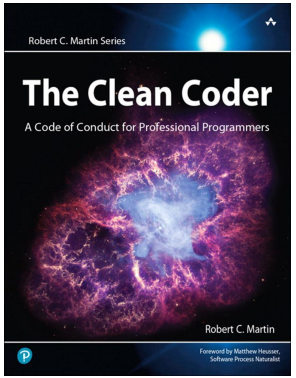




Hablemos de profesionalismo

Teradyne, 1979

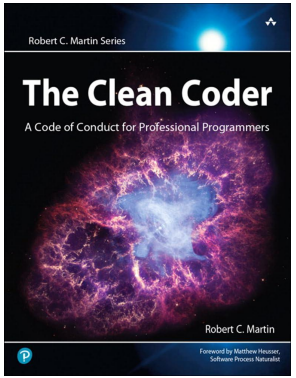
- En 1979 el tio Bob trabajaba para una empresa llamada Teradyne.
- Era el “ingeniero responsable” del software que controlaba un sistema de diagnóstico de líneas telefónicas.
- Conectaba una minicomputadora central con microcomputadoras satélite por líneas telefónicas.
- Cada noche, el sistema ejecutaba una rutina nocturna que probaba decenas de miles de líneas.
- Cada mañana, generaba un reporte con las líneas defectuosas.



The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

¿Quien lo usaba?

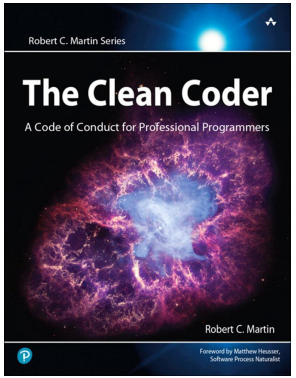
- Usado por gerentes de servicio de compañías telefónicas.
- Permitía detectar fallas antes de que los clientes se quejaran.
- Ayudaba a reducir las tasas de quejas, lo que afectaba directamente las tarifas que podían cobrar las telefónicas (reguladas por comisiones de servicios públicos).



The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

¿Por qué era tan sensible?

- Si la rutina nocturna fallaba, el sistema no generaba el informe.
- Técnicos sin trabajo programado.
- Clientes frustrados que llamaban para quejarse.
- Un solo error de una noche era suficiente para afectar métricas regulatorias.
- Reparaciones tardías.



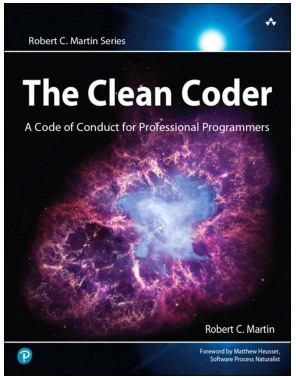
The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

Historia del error
El envío apresurado

Entrega Urgente:

- Tenía que entregar una nueva versión del software que incluía una función prometida a los clientes.
- Para cumplir con la fecha, omitió probar la rutina nocturna completa.
- En su razonamiento:

  “Nada de lo que cambié afecta esa parte... seguro funciona.”

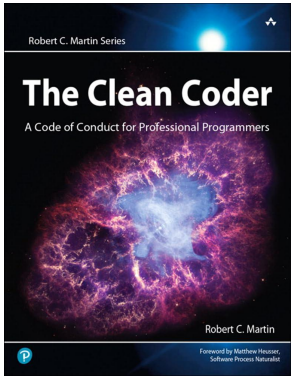


The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

Historia del error
El envío apresurado

Tensión con el equipo:

- Tom, el gerente de servicio de campo, recibió múltiples reclamos.
- Tio Bob trabajó días bajo presión para encontrar el error.
- La primera y segunda solución fallaron.
- Mientras tanto, Tom enfrentaba la frustración de los clientes.

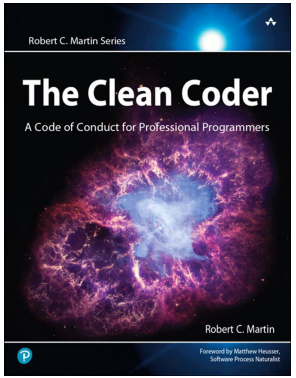


The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

Historia del error
El envío apresurado

Consecuencias inmediatas:

- La rutina nocturna falló en producción en varios clientes.
- No se generaron reportes de líneas defectuosas.
- Los clientes tuvieron que revertir a versiones anteriores, perdiendo la nueva funcionalidad.

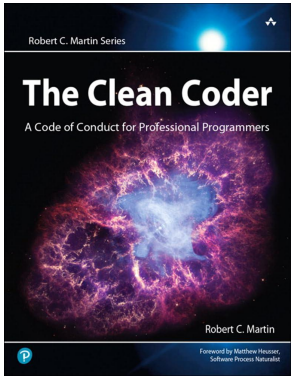


The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

Lección Profesional: Asumir Responsabilidad

Lo que NO se hizo bien

- Priorizó cumplir la fecha sobre la calidad del software.
- No comunicó que las pruebas no estaban completas.
- Asumió que todo funcionaba sin verificarlo.
- Buscó salvar su imagen personal, no el bienestar del cliente.

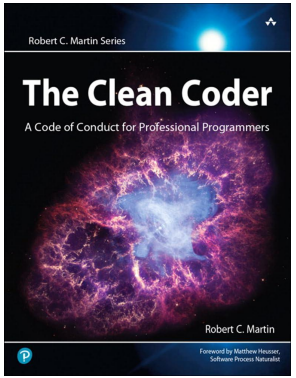


The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

Lección Profesional: Asumir Responsabilidad

Lo que se debió hacer

- Ser transparente: decir “no está listo” si las pruebas no están completas.
- Pensar en el impacto real en el cliente, no solo en el código.
- Priorizar la confiabilidad sobre la velocidad.
- Asumir el error, trabajar para solucionarlo y aprender de él.



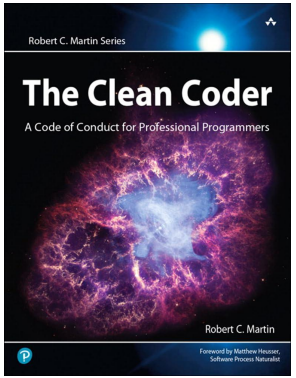
The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

El verdadero Profesionalismo

Primero, no hacer daño

La confianza del usuario se gana evitando errores que generen daño, no solo entregando funcionalidad.

- No solo importa que el software funcione, sino que funcione ***sin causar perjuicios***.
- Un error en producción puede significar pérdida de datos, tiempo y confianza.



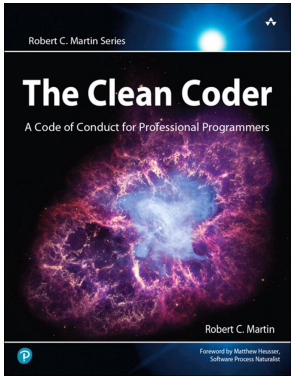
The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

El verdadero Profesionalismo

La responsabilidad es nuestra

El verdadero profesional asume la responsabilidad.

- Probar y validar no es opcional, es parte del trabajo.
- Decir “no está listo” a tiempo, aunque difícil, es más profesional que entregar algo roto.
- Salvar las apariencias no justifica el daño a otros.



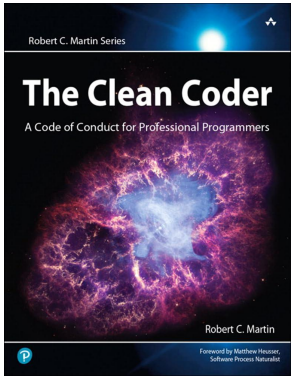
The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

El verdadero Profesionalismo

Profesionalismo ≠ Perfección

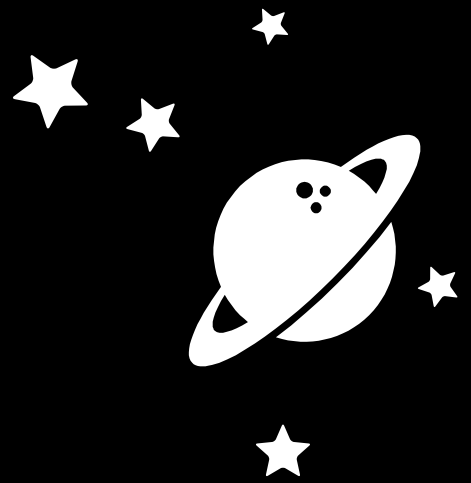
La excelencia técnica no es suficiente si no va acompañada de integridad

- El software es complejo y los errores existen.
- Ser profesional no es ser perfecto, es responder con responsabilidad.
- La confianza se construye con decisiones éticas, no solo con código brillante.



The Clean Coder: Un código de conducta para programadores profesionales.
Robert C Martin
2011

Principios de diseño



Las técnicas se convierten en principios cuando son ampliamente aceptadas, practicadas y demuestran su utilidad a lo largo del tiempo.

Principios de diseño

Son guías fundamentales que orientan la construcción de software de forma que sea:

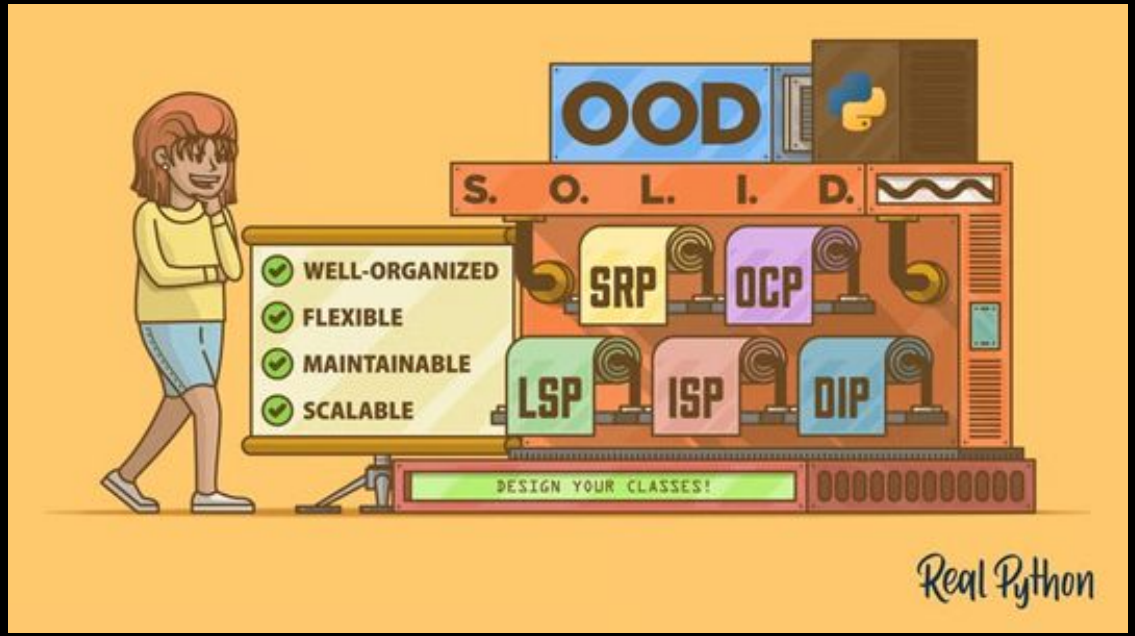
- Más **comprensible**
- Más **flexible** ante cambios
- Más fácil de **mantener y extender**

Aplicar principios de diseño no es solo una buena práctica técnica:

Es una muestra de **profesionalismo y responsabilidad** como desarrolladores.

En esta sección trataremos tres pilares esenciales:

- **SOLID**: diseño orientado a objetos profesional
- **KISS**: Keep It Simple, Stupid
- **DRY**: Don’t Repeat Yourself


OOD
S. O. L. I. D.
WELL-ORGANIZED
FLEXIBLE
MAINTAINABLE
SCALABLE
SRP
OCP
LSP
ISP
DIP
DESIGN YOUR CLASSES!
Real Python

SOLID

SOLID

En el mundo real, escribir código que "funciona" no es suficiente.

Necesitamos escribir código que sea entendible, modificable y confiable con el tiempo.

Aquí es donde entran los principios SOLID:

Un conjunto de buenas prácticas para diseñar software orientado a objetos de manera profesional y sostenible.

Propuestos por Robert C. Martin ("Tio Bob"), estos principios ayudan a:

- Reducir la complejidad del código
- Mejorar la mantenibilidad
- Evitar errores comunes de diseño
- Favorecer el trabajo en equipo y la evolución del sistema

S – Single Responsibility Principle (SRP)
Principio de Responsabilidad Única

"Una clase debe tener una sola razón para cambiar."
— Robert C. Martin

¿Qué significa?
Cada entidad (clase, función o módulo) **debe tener una única responsabilidad clara**, es decir, encargarse de una sola cosa dentro del sistema.

- Facilita el mantenimiento del código
- Reduce el riesgo de errores al hacer cambios
- Favorece el diseño modular y el trabajo en equipo

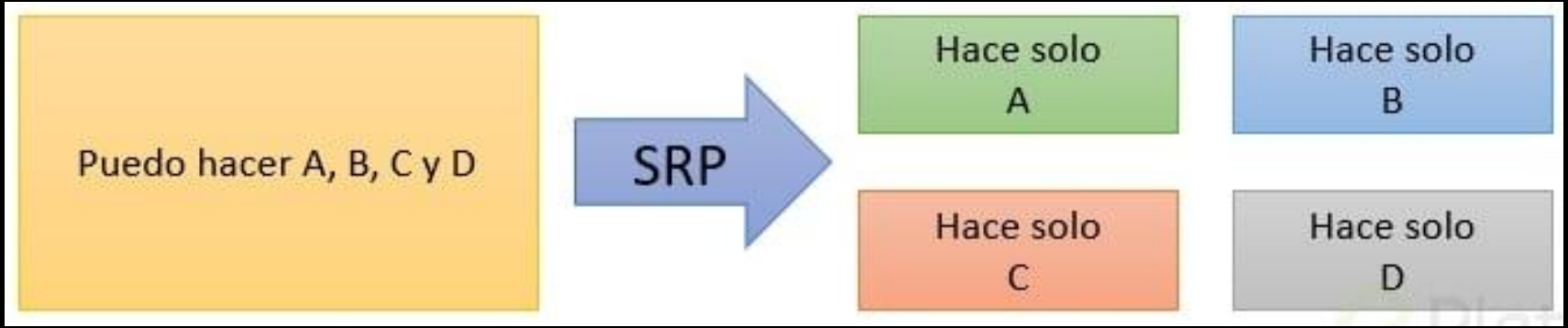
Evita: Acumular múltiples responsabilidades en una sola clase (por ejemplo, lógica de negocio + persistencia + presentación)

https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

S – Single Responsibility Principle (SRP)

Principio de Responsabilidad Única

"Una clase debe tener una sola razón para cambiar."
— Robert C. Martin

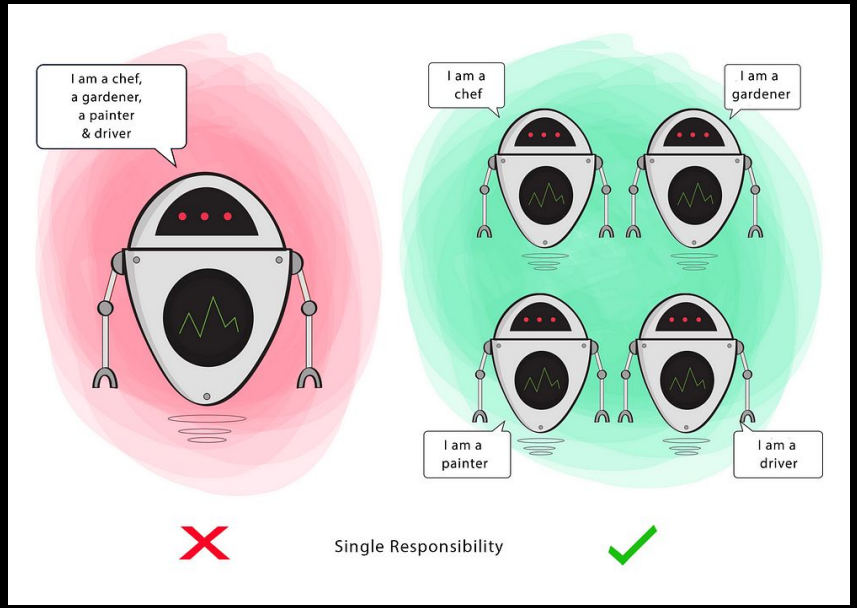


https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

S – Single Responsibility Principle (SRP)

Principio de Responsabilidad Única

"Una clase debe tener una sola razón para cambiar."
— Robert C. Martin



https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e8

O – Open/Closed Principle (OCP)
Principio de abierto/cerrado

"Las entidades de software deben estar abiertas para extensión, pero cerradas para modificación." — Bertrand Meyer

¿Qué significa?

El código debe permitir **agregar nuevos comportamientos sin alterar** el código existente. Así evitamos romper funcionalidades ya probadas.

- Facilita la evolución del sistema sin introducir regresiones
- Reduce la necesidad de modificar clases estables
- Favorece el uso de abstracciones e interfaces

Evita:

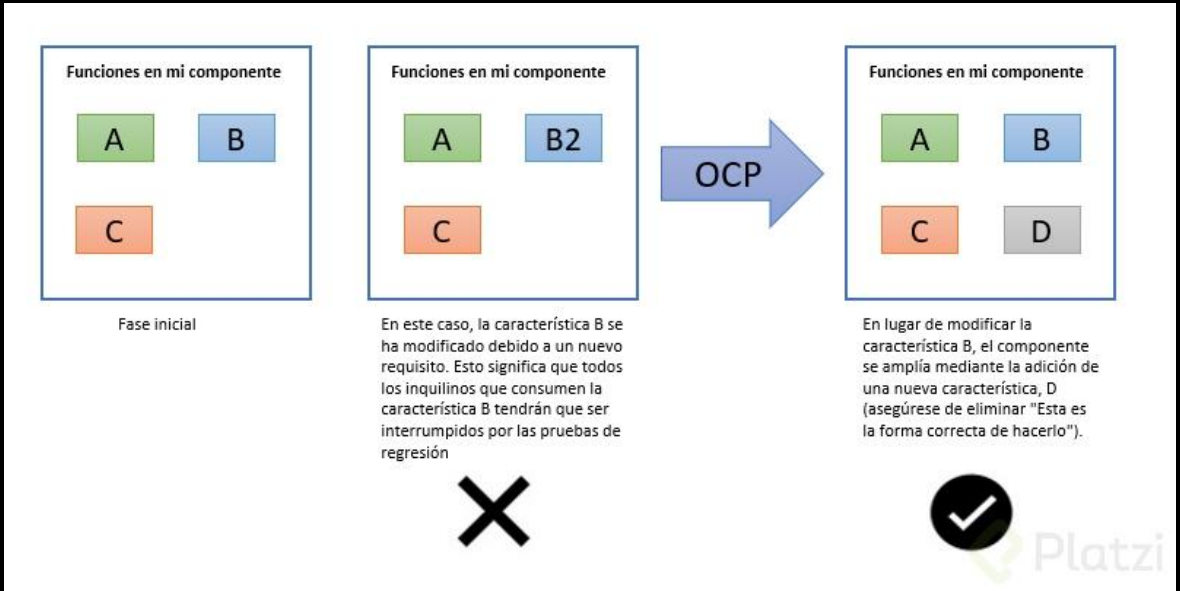
- Reescribir clases cada vez que hay un nuevo requerimiento
- Mezclar lógica nueva con lógica antigua en el mismo bloque de código

https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

O – Open/Closed Principle (OCP)

Principio de abierto/cerrado

"Las entidades de software deben estar abiertas para extensión, pero cerradas para modificación." — Bertrand Meyer

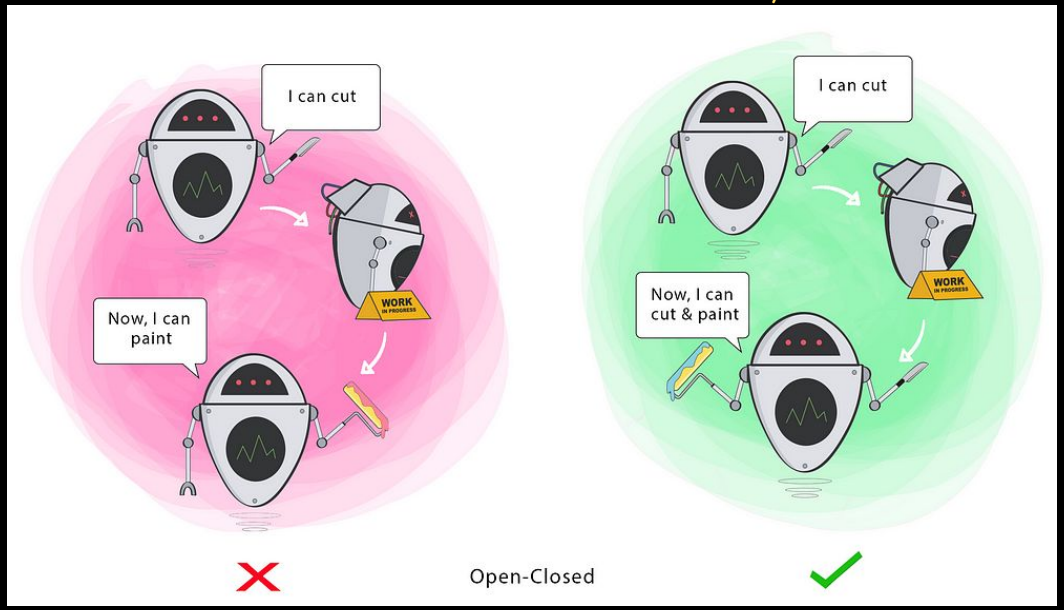


https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

O – Open/Closed Principle (OCP)

Principio de abierto/cerrado

"Las entidades de software deben estar abiertas para extensión, pero cerradas para modificación." — Bertrand Meyer



https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e8

L – Liskov Substitution Principle (LSP)
Principio de Sustitución de Liskov

"Los objetos de una clase derivada deben poder sustituir a los de su clase base sin alterar el comportamiento esperado." — Barbara Liskov

¿Qué significa?

Una subclase debe **respetar el contrato** de su superclase. Si algo espera un tipo base, debe poder usar un tipo derivado **sin problemas**.

- Permite reutilizar código con confianza
- Facilita el uso de polimorfismo de forma segura
- Mejora la robustez de los sistemas orientados a objetos

Evita:

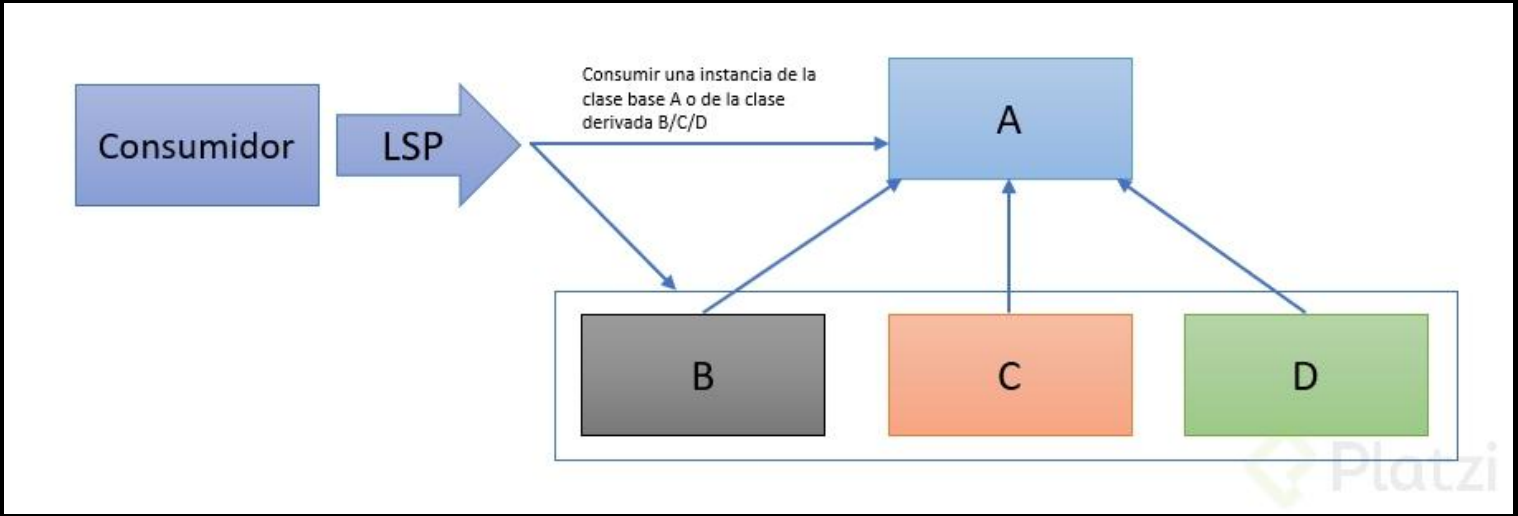
- Crear subclases que rompen el comportamiento del tipo base (por ejemplo, excepciones inesperadas o métodos no funcionales)
- Usar herencia cuando la relación "es un/a" no es clara o real

https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

L – Liskov Substitution Principle (LSP)

Principio de Sustitución de Liskov

"Los objetos de una clase derivada deben poder sustituir a los de su clase base sin alterar el comportamiento esperado." — Barbara Liskov

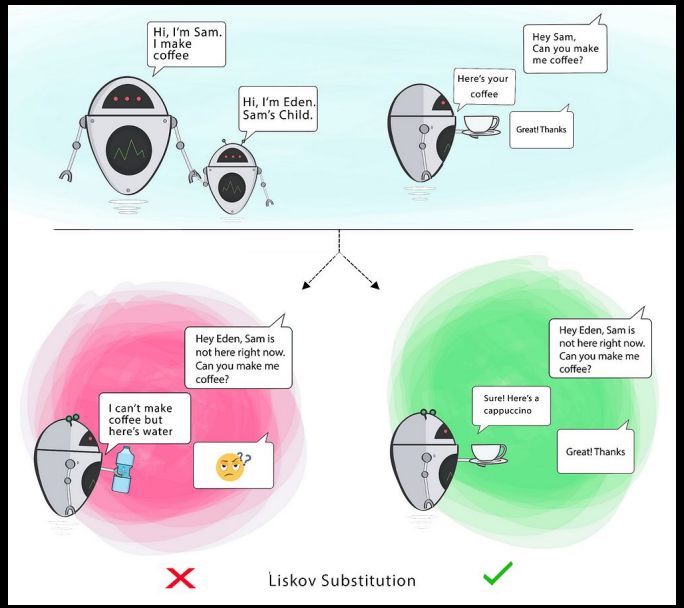


https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

L – Liskov Substitution Principle (LSP)

Principio de Sustitución de Liskov

"Los objetos de una clase derivada deben poder sustituir a los de su clase base sin alterar el comportamiento esperado." — Barbara Liskov



https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e8

I – Interface Segregation Principle (ISP)
Principio de Segregación de Interfaces

"Una clase no debe verse obligada a implementar métodos que no necesita."

¿Qué significa?

Es mejor tener interfaces pequeñas y específicas que una sola interfaz grande y genérica. Cada clase debe conocer y usar solo lo que necesita.

- Reduce el acoplamiento innecesario
- Hace el sistema más flexible ante cambios
- Facilita la implementación y el mantenimiento

Evita:

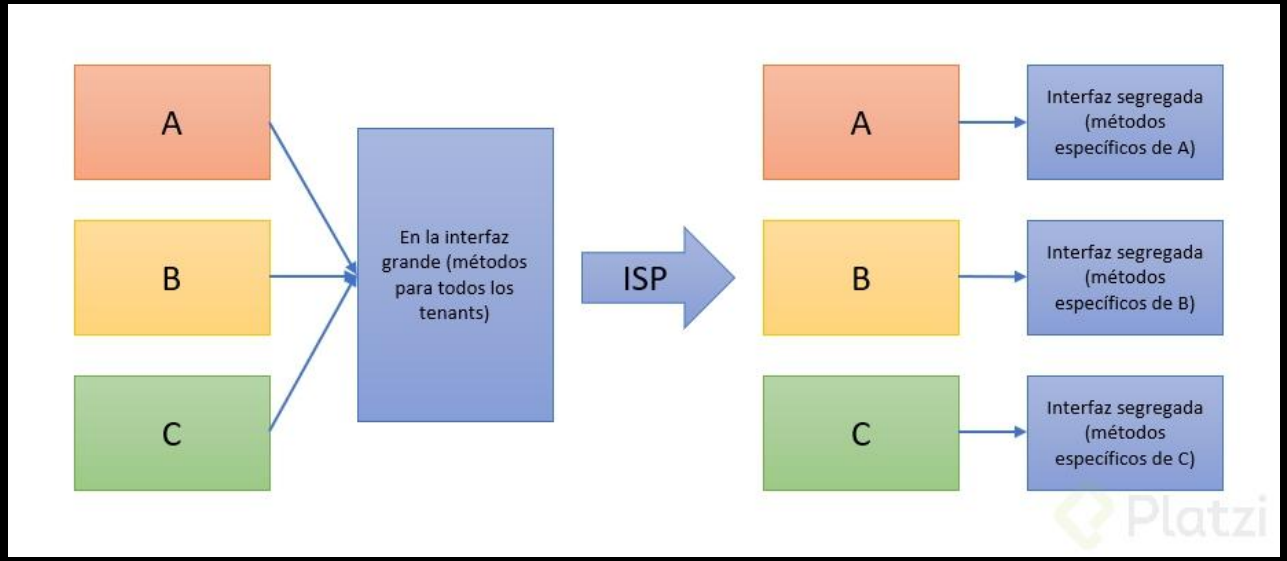
- Interfaces "todo en uno" con métodos innecesarios
- Forzar a las clases a depender de comportamientos que no utilizan

https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

I – Interface Segregation Principle (ISP)

Principio de Segregación de Interfaces

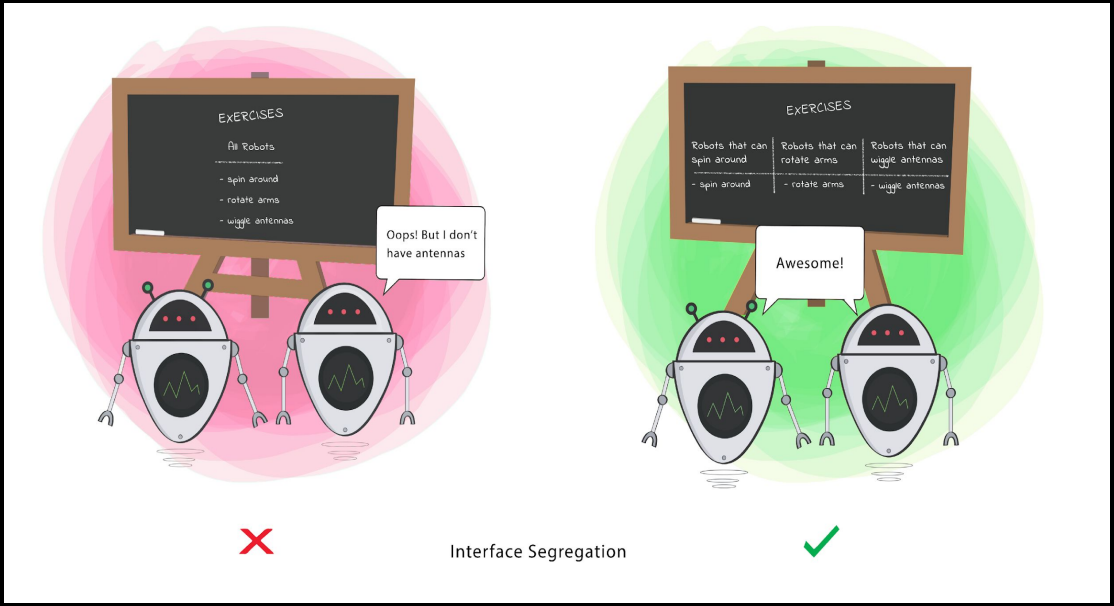
"Una clase no debe verse obligada a implementar métodos que no necesita."



https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

I – Interface Segregation Principle (ISP)
Principio de Segregación de Interfaces

"Una clase no debe verse obligada a implementar métodos que no necesita."



https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e8

D – Dependency Inversion Principle (DIP)
Principio de Inversión de Dependencias

"Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones."

¿Qué significa?

El código debe depender de interfaces (abstracciones), no de implementaciones concretas. Así se reduce el acoplamiento y se mejora la flexibilidad del diseño.

- Facilita el cambio de tecnologías o implementaciones (por ejemplo, cambiar una base de datos)
- Mejora la capacidad de hacer pruebas unitarias (mocking de interfaces)
- Refuerza la separación de responsabilidades

Evita:

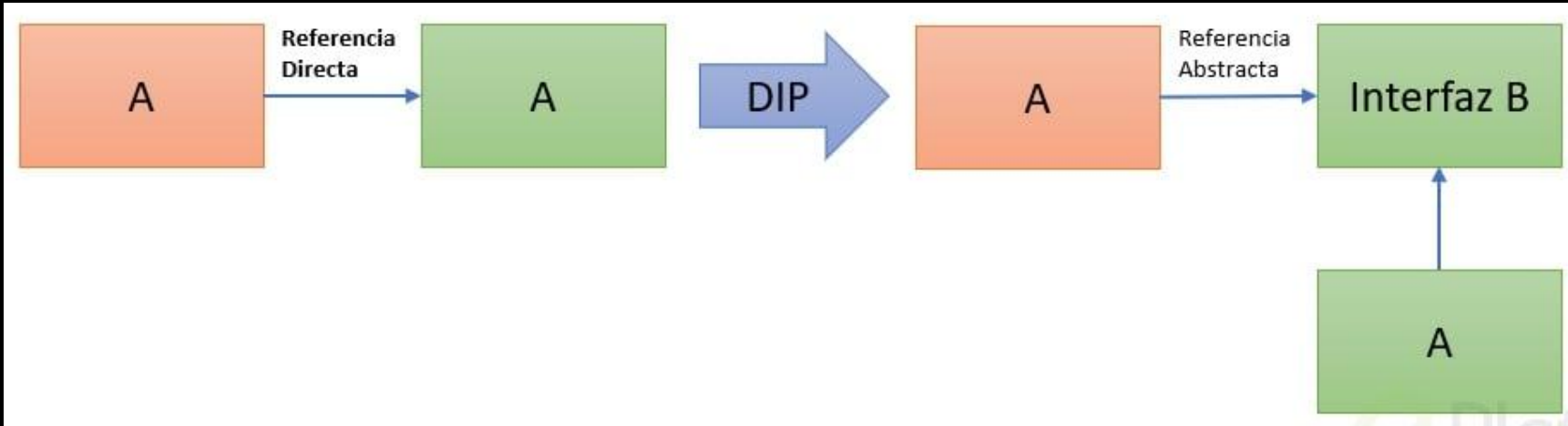
- Crear objetos concretos directamente dentro del código.
- Acoplar módulos que deberían evolucionar de forma independiente

https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

D – Dependency Inversion Principle (DIP)

Principio de Inversión de Dependencias

"Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones."

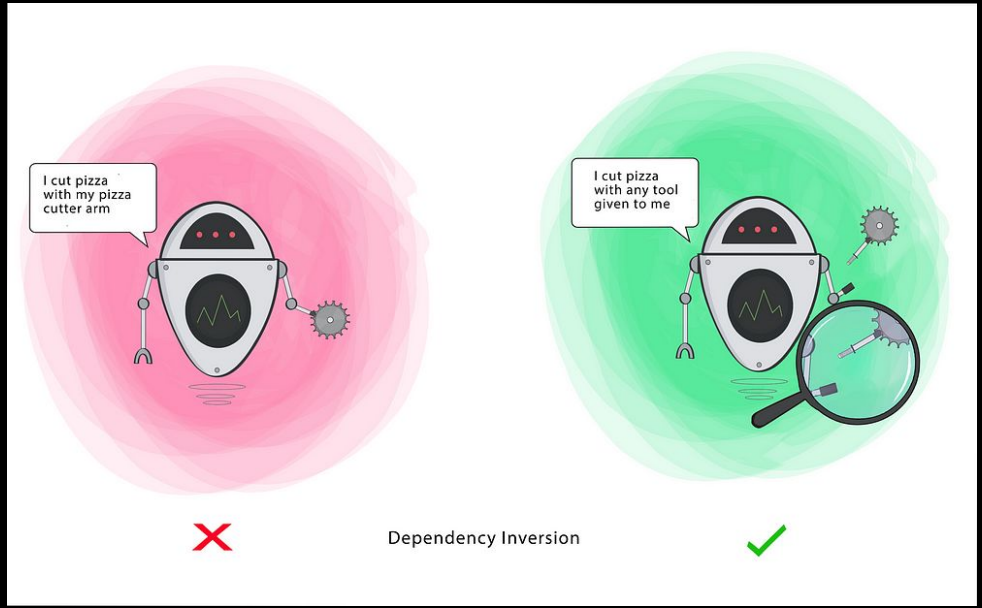


https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

D – Dependency Inversion Principle (DIP)

Principio de Inversión de Dependencias

"Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones."



https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e8

¿Qué significa SOLID?

Un acrónimo de cinco principios clave:

- **S**: Single Responsibility – Una clase, una responsabilidad
- **O:** Open/Closed – Extensible sin modificar
- **L**: Liskov Substitution – Herencia coherente
- **I**: Interface Segregation – Interfaces enfocadas
- **D**: Dependency Inversion – Acoplamiento mínimo

SOLID es tu chaleco antibalas contra el caos.

https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

DRY

DRY – Don’t Repeat Yourself

No te repitas

“Cada pieza de conocimiento debe tener una única representación no ambigua y definitiva en el sistema.” — Andy Hunt & Dave Thomas

¿Qué significa?

Evita repetir lógica, estructuras o datos. Un sistema debe diseñarse para que cada funcionalidad se implemente en un solo lugar, y sea reutilizable.

- Reducción de errores por inconsistencias
- Menor esfuerzo de mantenimiento
- Cambios más seguros y localizados

Evita:

- Copiar y pegar código o estructuras de datos
- Repetir validaciones, consultas o reglas de negocio en varios lugares

https://k3no.medium.com/how-to-dry-your-python-code-639a17c65207

DRY – Don’t Repeat Yourself

No te repitas

“Cada pieza de conocimiento debe tener una única representación no ambigua y definitiva en el sistema.” — Andy Hunt & Dave Thomas

```python
# VERBOSE

text1 = 1
text2 = 2
text3 = 3
text4 = 4
text5 = 5

print(f"Number: {text1}")
print(f"Number: {text2}")
print(f"Number: {text3}")
print(f"Number: {text4}")
print(f"Number: {text5}")

# DRY

texts = [1, 2, 3, 4, 5]

for i in range(len(texts)):
    print(f"Number: {texts[i]}")
```

DRYCodeProblem.py hosted with ❤ by GitHub view raw

Ejemplo práctico:

❌ **Código VERBOSE (Repetitivo)**
Difícil de mantener, propenso a errores si cambia la lógica.

✅ **Código DRY (Reutilizable y mantenible)**
Un solo lugar para definir la lista, y una sola línea para imprimir.

Más limpio, más escalable, más profesional.

https://k3no.medium.com/how-to-dry-your-python-code-639a17c65207

KISS

KISS – Keep It Simple, ~~Stupid~~

Mantenlo simple, no lo compliques

"La simplicidad es la máxima sofisticación." — Leonardo da Vinci

¿Qué significa?

Diseña el sistema de la forma más simple posible, evitando complejidades innecesarias, soluciones sobreingenieradas o tecnologías no probadas.

- Reduce errores en el diseño y desarrollo
- Facilita el mantenimiento del sistema por otros equipos
- Permite entender y modificar el sistema más fácilmente
- Mejora la gestión de cambios y reduce el riesgo de regresiones

Evita:

- "Reinventar la rueda" sin necesidad
- Incluir nuevas tecnologías por moda
- Sobrecomplicar lo que puede resolverse con una solución clara y probada

https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

KISS – Keep It Simple, Stupid

Mantenlo simple, no lo compliques

"La simplicidad es la máxima sofisticación." — Leonardo da Vinci

```python
# Complejo
name = input("What's your name?")
if name != "":
    print("Hello, " + name + "!")
else:
    print("You didn't enter your name.")
```

```python
# Simple y Clara
name = input("What's your name?") or "Stranger"
print(f"Hello, {name}!")
```

Ejemplo práctico:

❌ **Versión innecesariamente compleja**
Funciona... pero tiene ruido y lógica redundante.

✅ **Versión simple y clara (KISS)**
Una sola línea extra le da un valor por defecto.

Menos líneas, más claridad, mismo resultado.

https://k3no.medium.com/how-to-dry-your-python-code-639a17c65207

DRY

D – Dependency Inversion Principle (DIP)
Principio de Inversión de Dependencias

"Los módulos de alto nivel no deben depender de módulos de bajo nivel. Ambos deben depender de abstracciones."

¿Qué significa?

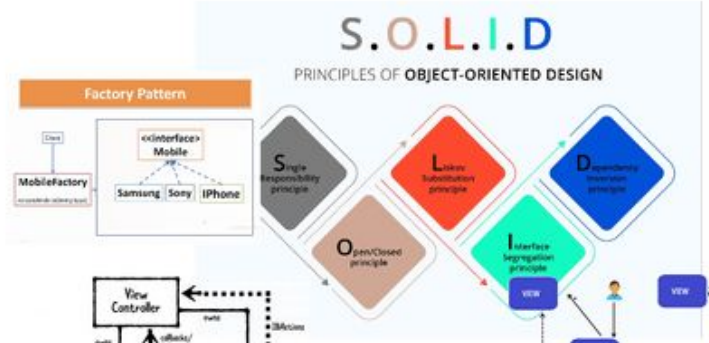
El código debe depender de interfaces (abstracciones), no de implementaciones concretas. Así se reduce el acoplamiento y se mejora la flexibilidad del diseño.

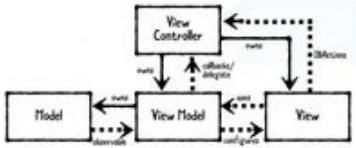
- Facilita el cambio de tecnologías o implementaciones (por ejemplo, cambiar una base de datos)
- Mejora la capacidad de hacer pruebas unitarias (mocking de interfaces)
- Refuerza la separación de responsabilidades

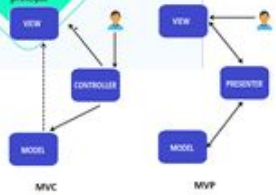
Evita:

- Crear objetos concretos directamente dentro del código.
- Acoplar módulos que deberían evolucionar de forma independiente

https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/







I will write only the code needed to solve the problem

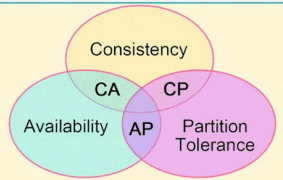
I will write only the code needed to solve the problem

System Design Acronyms

ByteByteGo

CAP

- C Consistency
- A Availability
- P Partition Tolerance



BASE

- BA Basically Available
- S Soft State
- E Eventual Consistency

ACID

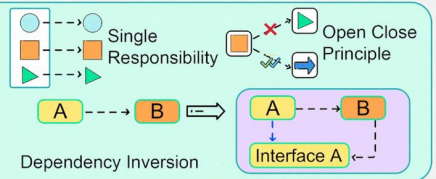
RDBMS golden standard

BASE

- Used in NoSQL
- States will be consistent over time

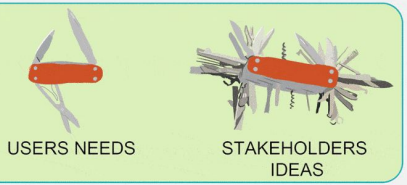
SOLID

- S Single Responsibility
- O Open Close Principle
- L Liskov Substitution
- I Interface Segregation
- D Dependency Invers



KISS

- K Keep
- I It
- S Simple,
- S Stupid



https://blog.bytebytego.com/p/ep86-cap-base-solid-kiss-what-do

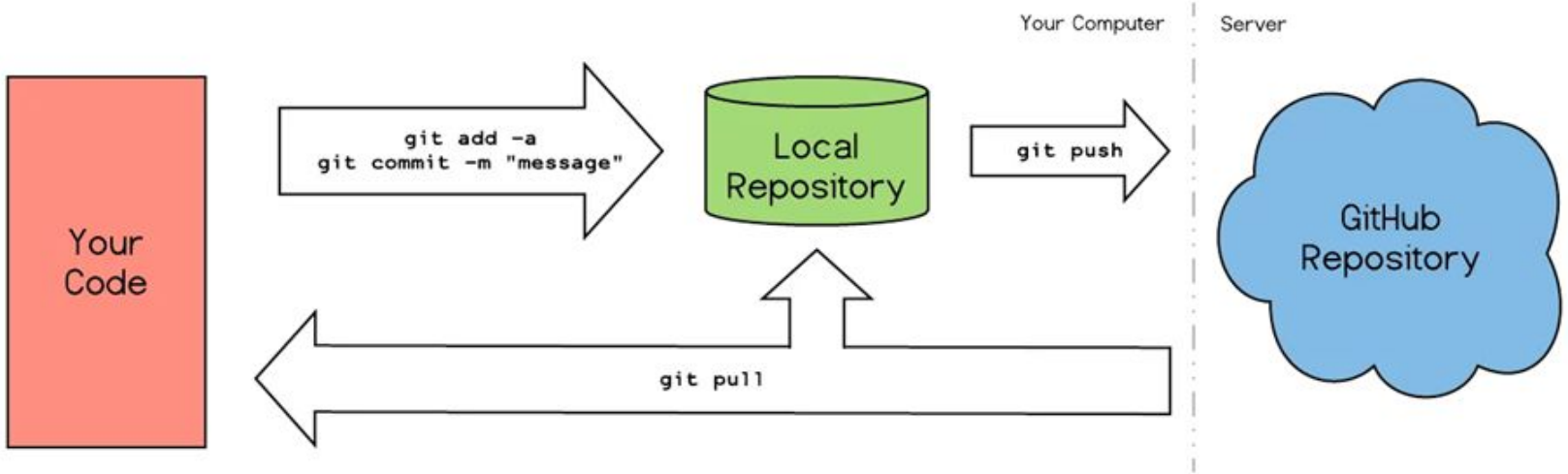
NAVAJA DE OCKHAM

https://impulso06.com/descubre-como-la-navaja-de-ockham-te-puede-ayudar-en-tu-trabajo/

CONTROL DE VERSIONES

El control de versiones es una herramienta esencial para todo desarrollador.
Permite registrar, comparar, revertir y colaborar en los cambios de código a lo largo del tiempo.

Control de Versiones



Commits Convencionales

Son una especificación para dar estructura a los mensajes de commit en Git. Esta convención proporciona un conjunto fácil de reglas para crear un historial de commits explícito, lo que facilita la creación de herramientas automatizadas y la generación de changelogs.

VENTAJAS:

Generación automática de changelogs

Determinación automática de incrementos de versión semántica

Comunicar la naturaleza de los cambios a compañeros de equipo y otros interesados

Facilitar la contribución a proyectos al estandarizar la estructura de commits

Hacer más útil el historial de commits y facilitar la navegación

Control de Versiones

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Commits Convencionales

Tips principales:

Feat	Nueva característica
Docs	Cambios en documentación
Style	Cambios que no afectan al significado del código (espaciado, formato, etc.)
Refactor	Cambio de código que no corrige error ni añade funcionalidad
Perf	Mejora de rendimiento
Test	Añadir o corregir pruebas
Build	Cambios en sistema de build o dependencias externas
Chore	Otros cambios que no modifican src o test

Control de Versiones

"Esto es GIT. Rastrea el trabajo colaborativo en proyectos a través de un hermoso modelo de árbol basado en teoría de grafos distribuidos."

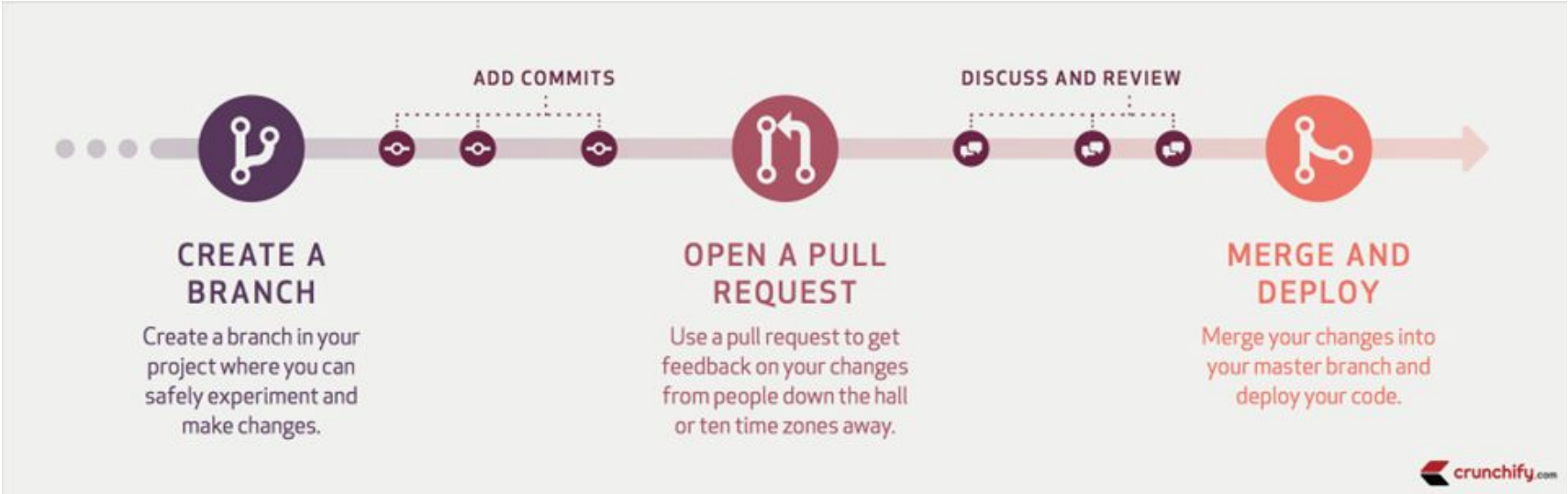
"Genial. ¿Cómo lo usamos?"

"Ni idea. Solo memoriza estos comandos de terminal y escríbelos para sincronizar. Si obtienes errores, guarda tu trabajo en otro lugar, borra el proyecto, clónalo de nuevo y descarga una copia nueva."

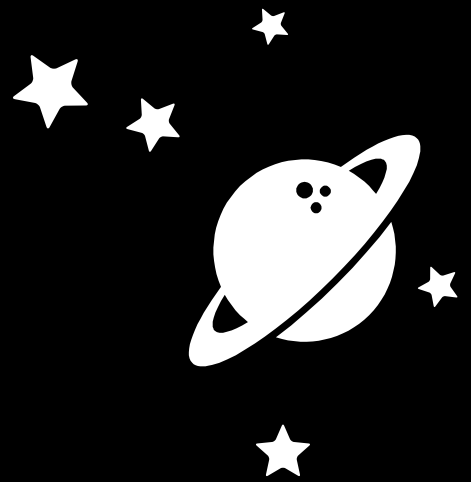
Ciclo de vida de un cambio

1. Crear rama desde main:
   git checkout main
   git pull
   git checkout -b feature/nueva-funcionalidad
2. Trabajar con commits pequeños y significativos
3. Mantener la rama actualizada con la base:
   git checkout main
   git pull
   git checkout feature/nueva-funcionalidad
   git rebase main
4. Crear Pull Request cuando esté listo
5. Después de la revisión y aprobación, merge a la rama base

Ciclo de vida de un cambio



Entornos de desarrollo



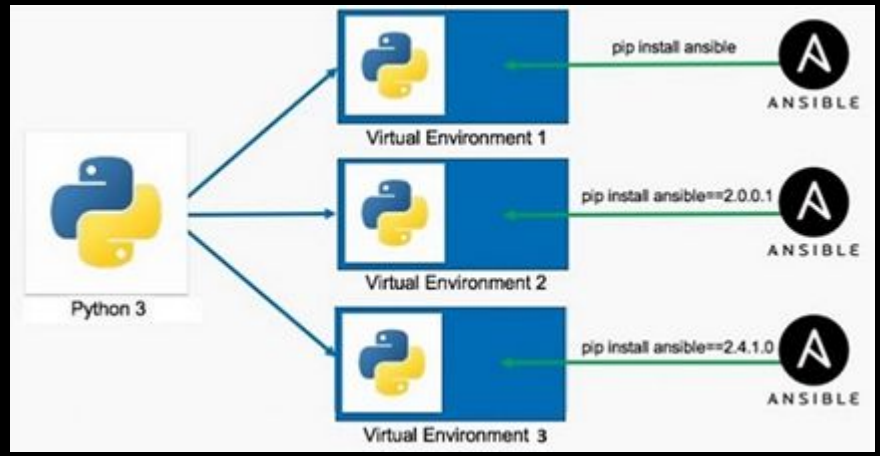
El laboratorio donde las ideas se convierten en código y los errores se transforman en lecciones.

Entornos Virtuales

Son espacios aislados donde podemos instalar dependencias de Python sin afectar al sistema global ni a otros proyecto

Ventajas:

- Gestionar dependencias específicas para cada proyecto.
- Evitar conflictos entre versiones de paquetes.
- Reproducir fácilmente el entorno de desarrollo en otras máquinas.



Creación y activación con venv

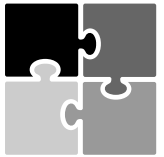
```
# Crear un entorno virtual
python -m venv .venv

# Activar el entorno virtual
# En Linux/Mac:
source .venv/bin/activate

# Verificar activación (el prompt cambia y muestra el entorno)
which python  # Debe mostrar el python del entorno virtual
```

https://docs.python.org/3/tutorial/venv.html

Creación y activación con venv

```
back-to-the-future@LAPTOP-F1S8CTES:~/python_para_todos$ ls -la
total 8
drwxr-xr-x  2 back-to-the-future back-to-the-future 4096 Apr 19 23:55 .
drwx------ 16 back-to-the-future back-to-the-future 4096 Apr 19 23:53 ..
back-to-the-future@LAPTOP-F1S8CTES:~/python_para_todos$ which python3
/usr/bin/python3
back-to-the-future@LAPTOP-F1S8CTES:~/python_para_todos$ python3 -m venv .venv
back-to-the-future@LAPTOP-F1S8CTES:~/python_para_todos$ ls -la
total 12
drwxr-xr-x  3 back-to-the-future back-to-the-future 4096 Apr 19 23:56 .
drwx------ 16 back-to-the-future back-to-the-future 4096 Apr 19 23:53 ..
drwxr-xr-x  5 back-to-the-future back-to-the-future 4096 Apr 19 23:56 .venv
back-to-the-future@LAPTOP-F1S8CTES:~/python_para_todos$ source .venv/bin/activate
(.venv) back-to-the-future@LAPTOP-F1S8CTES:~/python_para_todos$ which python3
/home/back-to-the-future/python_para_todos/.venv/bin/python3
(.venv) back-to-the-future@LAPTOP-F1S8CTES:~/python_para_todos$
```

Gestión de Dependencias con requirements.txt



El archivo **requirements.txt** se utiliza para gestionar las dependencias de un proyecto Python.

Especifica las librerías necesarias para ejecutar el proyecto.

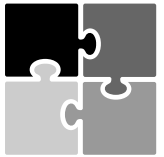
Facilita la instalación de todas las dependencias con un solo comando

```
pip install -r requirements.txt
```

Permite reproducir el entorno de desarrollo fácilmente en diferentes máquinas o entornos.

Es una forma estándar de asegurar que cualquier persona que trabaje en el proyecto tenga las mismas dependencias instaladas.

Gestión de Dependencias **con** requirements.txt



Generar archivo requirements.txt

```
pip freeze > requirements.txt
```

Instalar dependencias

```
pip install -r requirements.txt
```

¿Listo para automatizar o necesitas más café?

https://forms.office.com/r/tzdaBiBwtM

Enlaces recomendados

https://97cosas.com/programador/

https://platzi.com/tutoriales/2198-csharp-introduccion/11519-principios-de-diseno/

https://medium.com/backticks-tildes/the-s-o-l-i-d-principles-in-pictures-b34ce2f1e898

https://gist.github.com/evandrix/2030615

https://ai.gopubby.com/writing-good-python-code-the-zen-of-python-df7830b52195

¿Qué sigue?

Vamos a aplicar todo esto en un caso real:
convertir un archivo Excel lleno de información sucia, ambigua, repetitiva…
en un XML limpio, validado, profesional.

Porque automatizar **no es solo que funcione**.
Es que funcione **bien, siempre, para todos**.


GRACIAS POR
SU ATENCIÓN
BUENO A LOS QUE
PRESTARON ATENCIÓN!
memegenerator.es