NotCo

Our favorite food in a better way.

Ingeniería de software en NotCo

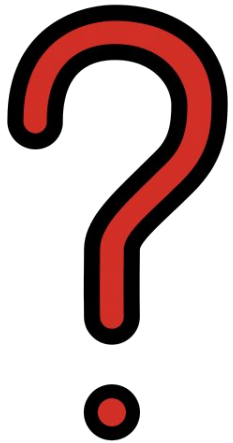
Machine Learning @ The Not Company

Algunos consejos de cómo ser viejo, para gente joven

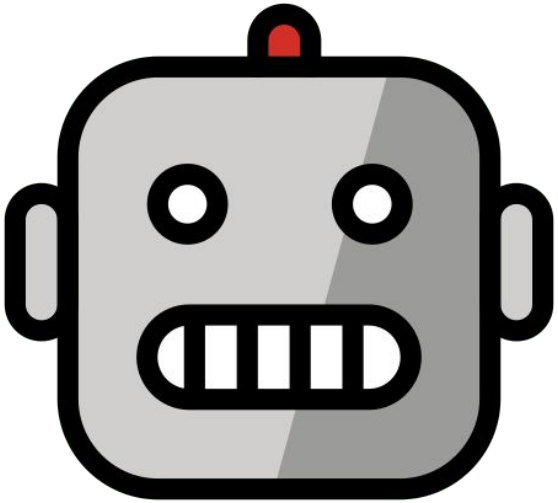
Machine Learning @ The Not Company

El temario de hoy

- ¿Qué es NotCo? ¿Por qué?
- Trabajando con el software 2.0
- Algunos guidelines para mantener software
- <algunas preguntas del público>



¿Qué es la inteligencia artificial?



¿Qué es la inteligencia artificial?

- La habilidad de un computador (o máquina) de **pensar** y **aprender**.
- Es una pieza de software para **resolver** alguna tarea en particular.
- Parte como disciplina en los **años 1950**.
- Es actualmente utilizada como herramienta en **múltiples disciplinas**.

- Algunos ejemplos concretos,
  - un motor para jugar ajedrez,
  - encontrar el camino más corto desde A hasta B,
  - recomendar el siguiente video en YouTube,
  - etcétera.

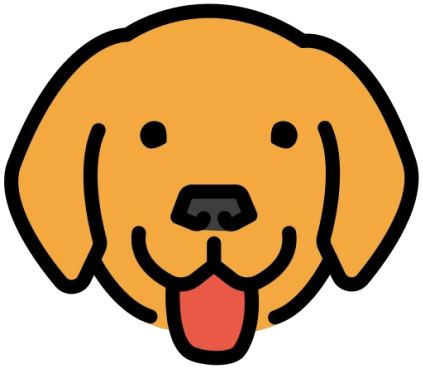
¿Qué es machine learning?

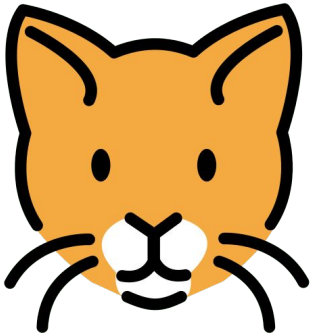
- Es una **subdisciplina** en inteligencia artificial.
- El nombre «aprendizaje de máquina» es porque… la máquina aprende.
- Este aprendizaje está basado a partir de **datos** que el computador ya ha visto.

- Algunas aplicaciones que ocurren **hoy en día**,
  - clasificar *spam* en un correo,
  - identificar objetos en una imagen,
  - realizar un diagnóstico médico,
  - predecir el tráfico de automóviles,
  - reconocer y entender la voz humana,
  - etcétera.

¿Qué es machine learning?

- El ejemplo clásico: clasificación de imágenes.





¿Qué es NotCo? ¿Por qué?

- The Not Company (NotCo) fue fundada el año 2015 en **Chile**.
- NotCo es una compañía de base **tecnológica** y **científica**.
- Ya somos casi **250** personas repartidas en **Chile**, **Argentina**, **Brasil**, & **Estados Unidos**.
- El equipo de tecnología cuenta con ~30 personas.

La misión de NotCo

Ofrecer alimentos deliciosos, saludables, & respetuosos con el medioambiente.

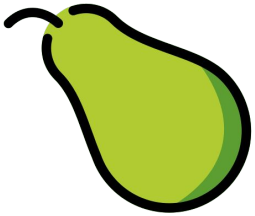
¿Qué es NotCo? ¿Por qué?

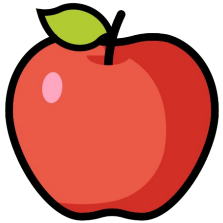
- Hasta hoy, hemos lanzado cuatro productos.
  - NotMayo (original, spicy, garlic, olive)
  - NotMilk
  - NotIceCream
  - NotBurger

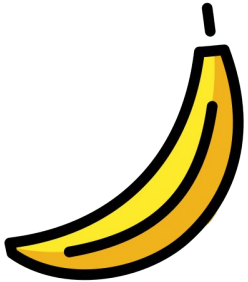
¿Por qué usamos machine learning en NotCo?

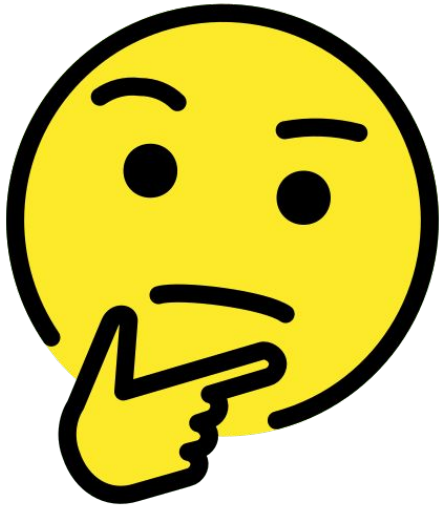
- **Qué no es:** un robot en la cocina (no todavía al menos)
- **Qué sí es:** una colección de módulos que nos ayudan en la toma de decisiones
  - generar nuevas formulaciones para lograr un producto
  - predecir textura, sabor, color, aroma, regusto
  - sugerir pasos en una preparación
  - minimizar la cantidad de experimentos científicos
  - etcétera.

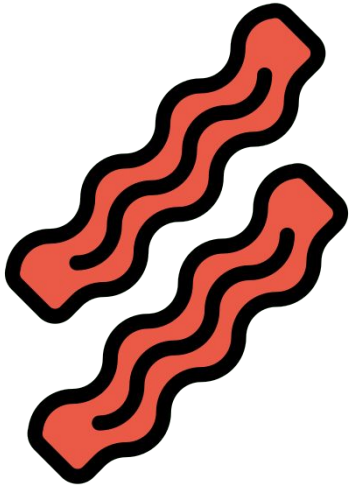
Exploración de nuevas fórmulas



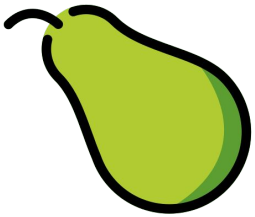


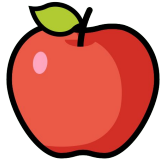


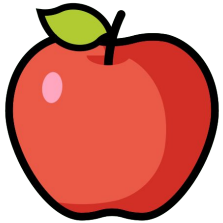


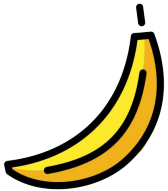


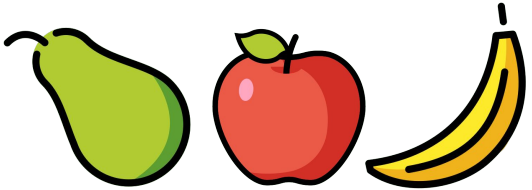
Exploración de nuevas fórmulas

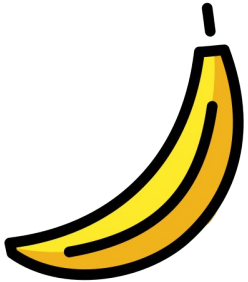


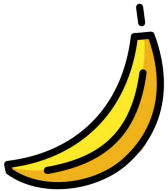




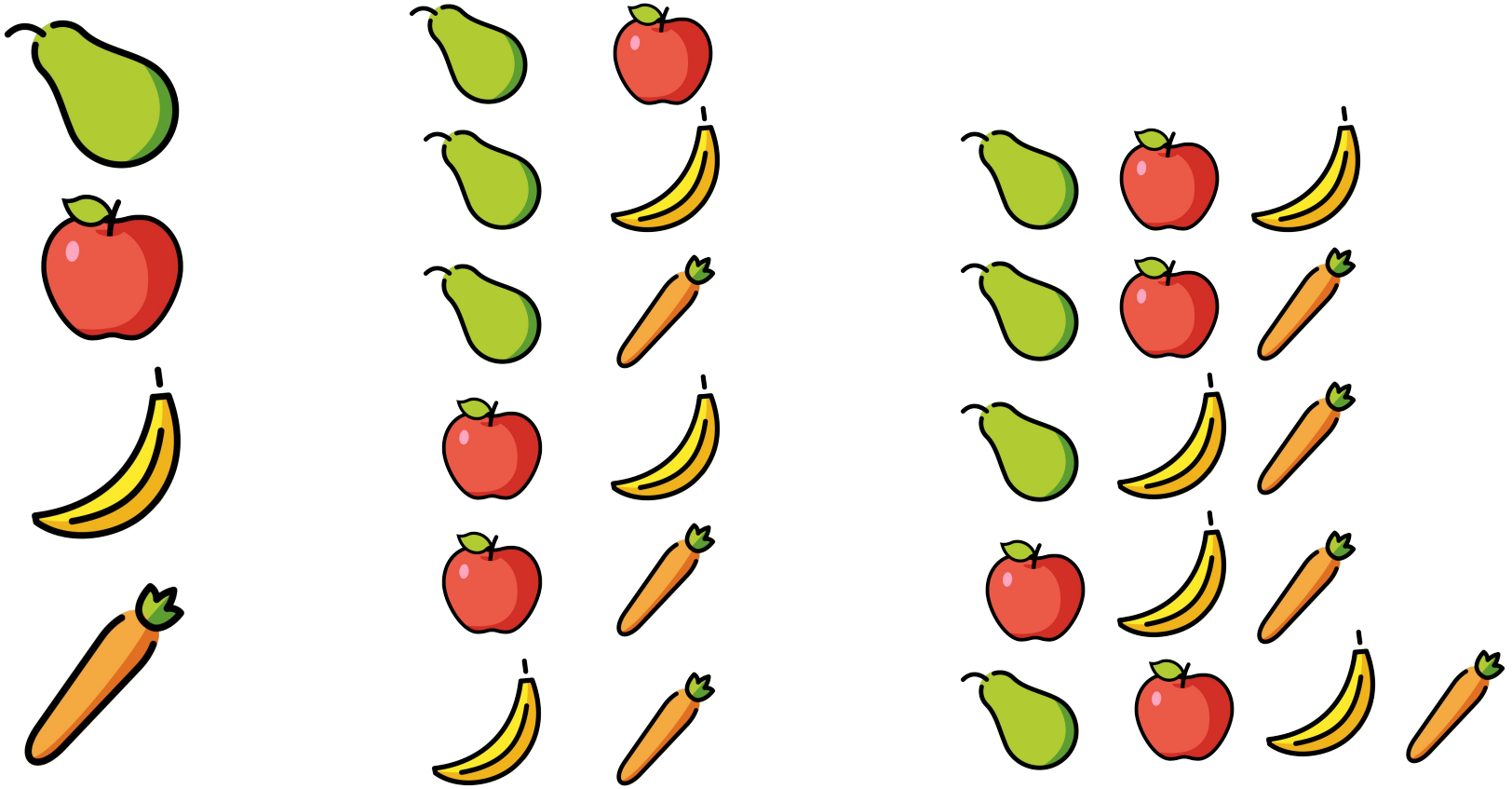




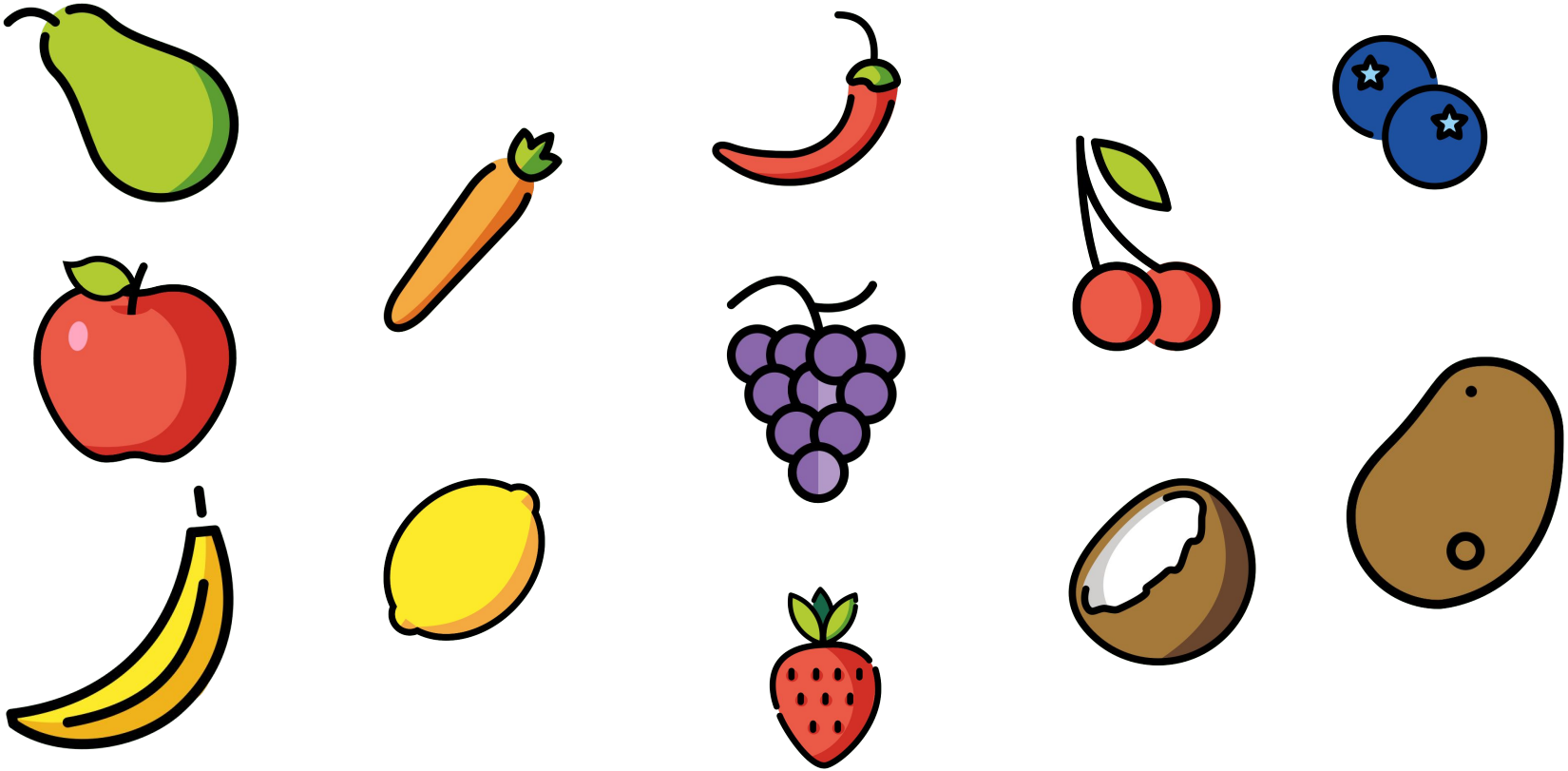




Exploración de nuevas fórmulas



Exploración de nuevas fórmulas



Exploración de nuevas fórmulas

- Si tengo **n** ingredientes disponibles…

$$f(n) = 2^n - 1$$

- Calculemos algunos valores de **n**.
  - **f(3) = 7**
  - **f(5) = 31**
  - **f(10) = 1.023**
  - **f(20) = 1.048.575**
  - **f(50) = 1,12e+15**
  - **f(100) = 1,26e+30**

¿Qué es NotCo? ¿Por qué?

- **Nuestra apuesta:** la colaboración entre humanos y sistemas de *machine learning* funciona mejor que cada uno de ellos por separado.
- Esta colaboración es factible en la industria alimentaria y podría entregar una nueva forma de crear alimentos en base a plantas.

¿Y dónde está el **software**?

- Para que sistemas basados en *machine learning* funcionen, necesitamos construir una **interfaz** entre ellos y los humanos.
- Esto está relacionado con el «software 2.0» de Andrej Karpathy.
- En NotCo, estamos construyendo software para…
  - aumentar y limpiar datasets,
  - automatizar procesos repetitivos,
  - ofrecer interfaces amigables hacia usuarios no expertos.

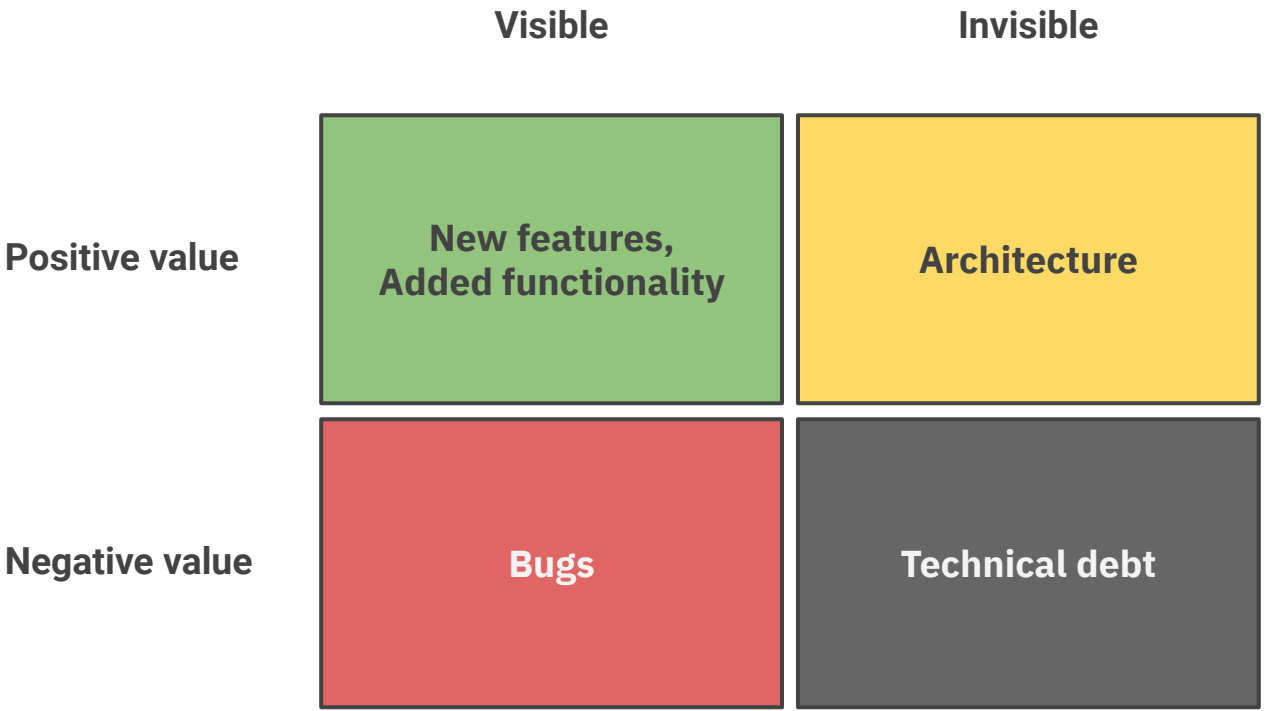
¿Y dónde está el software?

- En NotCo necesitamos software porque:
  - Los chefs deben pedir nuevas fórmulas para cocinar *animal-based targets*,
  - Los chefs deben visualizar e interactuar con estas fórmulas,
  - Los chefs deben documentar y revisar sus recetas,
  - Y más software para los food scientists,
  - Y más software para otras áreas de la compañía,
  - Y más software si licenciamos esta tecnología,
  - Y más software si…

Y ahora, algunos *guidelines*

- **Disclaimer:** estos son sólo *guidelines*.
- Finalmente, todo esto **depende** fuertemente de…
  - el dominio y el problema a resolver,
  - el tamaño del equipo y su experiencia,
  - la etapa actual de la compañía,
  - otros criterios que ahora no puedo predecir.

Manejo de deuda técnica



Gentileza de [Philippe Kruchten]

Manejo de deuda técnica

- ¿Qué es la **deuda técnica**?

  def -- el costo implícito del retrabajo adicional causado por elegir una solución, en lugar de utilizar un (¿mejor?) enfoque que llevaría más tiempo en su implementación.
- Algunas causas:
  - escribir pocos tests,
  - tomar atajos para cumplir con *deadlines*,
  - escribir código sin documentación,
  - posponer un *refactor* de manera indefinida,
  - etcétera.

Manejo de deuda técnica

- Es una analogía con el concepto financiero: la deuda **no es mala *per se***, pero mucho ojo con los intereses que van **acumulándose**.
- Entonces, una pregunta podría ser: «¿cómo hago para no tener más deuda?»
- Pero una mejor pregunta es: «¿cómo hago para **manejar** esta deuda?»
- Mantén un **balance** entre *value creation* y mantención del código.

Manejo de deuda técnica

- Hacer la deuda técnica **explícita**. Lo importante es estar consciente de ella.
- Algunas técnicas para manejarla:
  - incluir la deuda al momento de estimar o de definir el apetito,
  - mantener un *technical backlog* para visibilizar,
  - realizar un *cleanup release* (¿cooldown?)

Manejo de deuda técnica

- En conclusión:
  - (casi) siempre habrá deuda técnica,
  - no es malo tener deuda,
  - no es necesario pagarla completamente.
- *Machine learning: the high-interest credit card of technical debt*

Boring technologies

- Otra pregunta: «¿cómo hago para elegir tecnologías?»
- Utiliza tecnologías estables y conocidas, cuyas fallas son conocidas.
- Esto te permitirá poner atención en lo que importa: **resolver el problema**.
- En pocas palabras: **optimiza globalmente** y elige un conjunto pequeño que pueda resolver los distintos problemas.
- Un stack *aburrido*: Python, Django, Vue.js, PostgreSQL.

Monocultura tecnológica

- Intenta, tanto como sea posible, **mantener un *stack* unificado de tecnologías**.
- Pueden haber algunos problemas con esto.
  - Ya no buscamos la mejor herramienta para el trabajo.
  - Esto es probablemente intelectualmente insatisfactorio.
- ¡Pero tiene **beneficios**!
  - El conocimiento de la tecnología es **compartido** entre el equipo.
  - Un cambio en algún lugar se **replicará** en otras partes.
    (e.g. mejorar un problema de *performance*, sirve para todos)
  - Escribir y **mantener** código es difícil: intenta minimizar esto.
- Y claro, hay excepciones: *machine learning* con Ruby.

Escribe código para problemas de tu negocio

- Cómo elegir entre: escribir tu propia solución o comprar algo *off-the-shelf*.
- Algunos ejemplos:
  - **Heroku** — *platform as a service*
  - **Sentry** — monitoreo de aplicaciones
  - **SendGrid** — envío de emails
  - **Cloudinary** — manejo de imágenes

Agrega complejidad sólo si es necesario

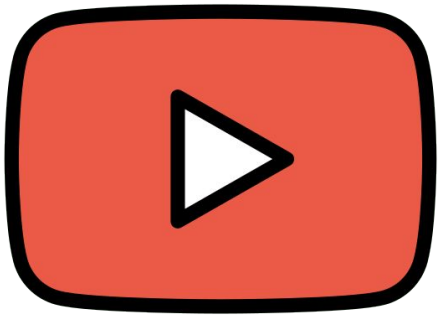
- Tener código no es algo bueno: ***code is a liability, not an asset***.
- ***The best code is the code you don't write.***
- **KISS** (keep it simple, stupid)
- **DTSTTCPW** (do the simplest thing that could possibly work)
- No sólo a nivel de código, pero también de soluciones.
  - e.g. *web sockets* vs. *long polling*
- Y también en flujos de trabajo.
  - GitHub issues, pull requests, milestones, code reviews, &c.
  - Linters, formatters, GitHub Actions = baja complejidad con mucho beneficio
  - Static typing = tests unitarios casi gratuitos

Testing & documentation

- Ojo con hacer tests sobre los **detalles de implementación**.
  - Si haces un refactor, los tests deberían seguir pasando.
- Los docs deben ser lo más alto nivel posible.
  - Podrían quedar desactualizados rápidamente.
  - Deben dar contexto a conceptos de alto nivel.
  - La idea es que expliquen el **por qué**; no tanto el cómo.
- Hay diferencias entre código de **librería** y código de **aplicación**: usa esto a tu favor.

Para saber más...

- Un video desarrollado por Google sobre NotCo
- Un episodio de la serie The Age of A.I. sobre NotCo



¡Muchas gracias!

¡Estamos contratando!

Si algo de estas slides tiene sentido, entonces escríbeme un correo a **nebil@notco.com**.

¿Hay preguntas?

NotCo

Our favorite food in a better way.

Referencias

- *Building software 2.0 stack* por Andrej Karpathy (director of AI @ Tesla)
- *Boring technology club* por Dan McKinley
- Greg Brockman (CTO @ Open AI)
- Kent C. Dodds

Agradecimientos

- Los bonitos emoji de OpenMoji.