Cronología del Desarrollo de Python

Python, uno de los lenguajes de programación más populares del mundo, fue creado por Guido van Rossum en los Países Bajos a finales de la década de 1980. Su desarrollo ha sido impulsado por una comunidad abierta, con contribuciones de numerosos desarrolladores clave.

Vamos a ver una cronología detallada de su evolución, destacando versiones principales, hitos, desarrolladores involucrados y las partes, módulos o características que crearon o contribuyeron. Esta cronología se basa en fuentes históricas y se extiende hasta 2025, con Python 3.13 como la versión estable más reciente y Python 3.14 en fase de preview.

- **Finales de 1980 - 1989: Concepción e Inicio del Desarrollo**
  Guido van Rossum, trabajando en el Centro de Matemáticas e Informática (CWI) en los Países Bajos, comienza a desarrollar Python en diciembre de 1989 como sucesor del lenguaje ABC. Inspirado en frustraciones con ABC, enfatiza la legibilidad del código, el manejo de excepciones y la interfaz con sistemas operativos como Amoeba. Van Rossum es el autor principal y se convierte en el "Benevolent Dictator for Life" (BDFL) hasta 2018. Contribuye con los elementos básicos: clases con herencia, funciones, tipos de datos como list, dict y str, y el sistema de módulos inspirado en Modula-3.

- **1991: Lanzamiento de Python 0.9.0 y 0.9.1**
  La primera versión pública (0.9.1) se publica en febrero de 1991 en alt.sources. Incluye manejo de excepciones similar a Modula-3 (con una cláusula else agregada por Van Rossum) y el modelo de excepciones. Guido van Rossum lidera el desarrollo inicial.

- **1994: Python 1.0**
  Lanzada en enero de 1994, introduce herramientas de programación funcional como lambda, map, filter y reduce, contribuidas por un hacker de Lisp que envió parches funcionales, según Van Rossum. Se forma el foro comp.lang.python como principal espacio de discusión.

- **1995-1996: Python 1.2 a 1.4 y Traslado a CNRI**
  Python 1.2 se lanza en abril de 1995 (última versión en CWI). Van Rossum se muda a la Corporation for National Research Initiatives (CNRI) en EE.UU., donde inicia el proyecto Computer Programming for Everybody (CP4E) financiado por DARPA. Python 1.4 (octubre de 1996) agrega argumentos por palabra clave inspirados en Modula-3, soporte para números complejos y ocultamiento básico de datos mediante mangling de nombres.

- **2000: Python 1.6, 2.0 y Traslados Institucionales**
  En septiembre de 2000, se lanza Python 1.6 bajo una licencia de CNRI, con negociaciones para hacerla compatible con GPL. El equipo central se mueve a BeOpen.com. Python 2.0 (16 de octubre de 2000) introduce comprehensions de listas (inspiradas en SETL y Haskell), un recolector de basura con detección de ciclos, conteo de referencias y soporte para Unicode. El proceso de desarrollo se vuelve más transparente y respaldado por la comunidad. Posteriormente, Van Rossum y el equipo PythonLabs se unen a Digital Creations.

- **2001: Python 2.2 y Unificación de Tipos**
  Lanzada en diciembre de 2001, unifica tipos y clases en una jerarquía orientada a

objetos pura. Agrega generadores inspirados en Icon (PEP 255, contribuidos por Neil Schemenauer, Tim Peters y Magnus Hetland). Tim Peters también contribuye con el "Zen of Python" (PEP 20) y el algoritmo Timsort para ordenamiento.

- **2003: Contribuciones en PEPs Clave**
  Raymond Hettinger co-autoriza el PEP 308 para expresiones condicionales. Phillip J. Eby co-autoriza el PEP 333 para la interfaz Python Web Server Gateway (WSGI). Facundo Batista autoriza el PEP 327 para el tipo de datos decimal en aritmética precisa.

- **2006: Python 2.5**
  Lanzada en septiembre de 2006, introduce la declaración with para gestores de contexto (RAII-like), reemplazando try/finally común.

- **2008: Python 2.6 y 3.0 (Py3K)**
  Python 2.6 (octubre de 2008) incluye características de Python 3, como modo de warnings (PEP 361, por Neal Norwitz y Barry Warsaw). Python 3.0 (3 de diciembre de 2008) es una revisión mayor incompatible hacia atrás: print como función (PEP 3105, por Georg Brandl), remoción de input antiguo, movimiento de reduce a functools, anotaciones de funciones opcionales (PEP 3107, por Collin Winter y Tony Lownds), unificación de str/unicode (PEP 3137, por Guido van Rossum), y cambios en división entera para retornar floats.

- **2010: Python 2.7 (Última de la Serie 2.x)**
  Lanzada como la versión final de Python 2, con backports de Python 3. Fin de soporte en 2020 (extendido desde 2015).

- **2014: Python 3.4**
  Introduce asyncio para programación asíncrona y pathlib para manejo de rutas de archivos orientado a objetos.

- **2015: Python 3.5**
  Agrega hints de tipos (PEP 484) y sintaxis async/await para programación asíncrona.

- **2016: Python 3.6**
  Incluye f-strings para interpolación de cadenas, underscores en literales numéricos y generadores asíncronos.

- **2018: Python 3.7 y Retiro de Van Rossum como BDFL**
  Introduce clases de datos y variables de contexto. Guido van Rossum anuncia su retiro como BDFL el 12 de julio de 2018; se forma un Steering Council de cinco miembros en 2019.

- **2019: Python 3.8**
  Agrega el operador walrus (:=) y parámetros solo posicionales. Łukasz Langa actúa como release manager para 3.8 y 3.9, y crea Black (formateador de código).

- **2020: Python 3.9**
  Introduce operadores de fusión y actualización de diccionarios (|, |=), y métodos como str.removeprefix(). Victor Stinner contribuye al núcleo de CPython.

- **2021: Python 3.10**
  Agrega pattern matching y gestores de contexto parentetizados.

- **2022: Python 3.11**
  Mejoras en rendimiento y mensajes de error más informativos.
- **2023: Python 3.12**
  Mejoras en mensajes de error, flexibilidad en f-strings, sintaxis de parámetros de tipo y formalización sintáctica de f-strings.
- **2024-2025: Python 3.13 y 3.14**
  Python 3.13 (lanzada en octubre de 2024) trae optimizaciones adicionales. Python 3.14 (preview en agosto de 2025) continúa la evolución con soporte anual de releases (PEP 602). Desarrolladores como Brett Cannon (sistema de imports), Barry Warsaw (módulo argparse) y Raymond Hettinger (módulo collections: deque, defaultdict; mejoras en itertools) siguen contribuyendo al núcleo.

El desarrollo de Python ha migrado a GitHub en 2017, con un enfoque en la comunidad. Desarrolladores como David Beazley (educador y contribuidor al núcleo) y Ned Batchelder han influido en la educación y mantenimiento. Hoy, Python es mantenido por un equipo de core developers, algunos pagados (como Van Rossum, Barry Warsaw y Brett Cannon con tiempo parcial), y voluntarios.

Concepto de BDFL

El término **BDFL** significa **Benevolent Dictator for Life** (en español, "Dictador Benevolente de por Vida"). Es un título utilizado en comunidades de desarrollo de software de código abierto para describir a un líder que tiene la autoridad final sobre las decisiones técnicas y de diseño de un proyecto, pero que ejerce este poder de manera benevolente, buscando el bien común del proyecto y su comunidad.

En el contexto de Python, **Guido van Rossum**, el creador del lenguaje, fue el BDFL desde los inicios del proyecto hasta su retiro de este rol el **12 de julio de 2018**. Como BDFL, Van Rossum tenía la última palabra en disputas sobre el desarrollo de Python, especialmente en la aprobación o rechazo de **Python Enhancement Proposals (PEPs)**, que son propuestas para mejoras o cambios en el lenguaje. Su liderazgo fue clave para mantener la coherencia y la visión de Python, aunque siempre fomentó la participación de la comunidad.

El término "benevolente" implica que el BDFL no actúa de manera autoritaria o arbitraria, sino que escucha a la comunidad, evalúa argumentos técnicos y toma decisiones en beneficio del proyecto. Sin embargo, la transición de Python 2 a Python 3 y controversias como el PEP 572 (sobre el operador de asignación :=) generaron tensiones que llevaron a Van Rossum a renunciar al rol, tras lo cual Python adoptó un modelo de gobernanza con un **Steering Council** (Consejo Directivo) compuesto por cinco miembros electos.

El concepto de BDFL no es exclusivo de Python; otros proyectos de código abierto, como Linux (con Linus Torvalds) y Django (con Jacob Kaplan-Moss y Adrian Holovaty en sus inicios), también han utilizado este modelo. Sin embargo, el término es más emblemático en Python debido al impacto y la longevidad del liderazgo de Van Rossum.

La conflictiva etapa de transición de 2.x a 3.x

Sin duda la etapa más conflictiva del desarrollo de Python fue la de la transición de 2.x a 3.x.

Historia Detallada de la Coexistencia de Python 2.x y 3.x

La historia de la coexistencia entre las versiones 2.x y 3.x de Python es un capítulo fascinante y controvertido en la evolución de este lenguaje de programación. Python, creado por Guido van Rossum en la década de 1980, experimentó un punto de inflexión con la introducción de Python 3, que buscaba corregir deficiencias fundamentales de Python 2, pero a costa de romper la compatibilidad hacia atrás. Esto generó una transición prolongada, llena de desafíos técnicos, debates comunitarios y divisiones. A continuación, relato la cronología, el proceso de transición, los problemas principales y las posturas de desarrolladores clave, basándome en fuentes históricas y análisis retrospectivos.

Cronología de los Eventos y Lanzamientos Principales

La coexistencia de Python 2.x y 3.x se extendió por más de una década, desde la concepción de Python 3 en 2006 hasta el fin oficial de soporte para Python 2 en 2020. Aquí va un timeline detallado:

- **Finales de la década de 1980 - 2000: Orígenes y Consolidación de Python 2.x**
  Python fue concebido por Guido van Rossum en 1989 mientras trabajaba en el Centro de Matemáticas e Informática (CWI) en los Países Bajos. La versión 2.0 se lanzó el 16 de octubre de 2000, introduciendo características como comprehensions de listas, recolección de basura con detección de ciclos y soporte inicial para Unicode. Siguieron actualizaciones como 2.1 (2001), 2.2 (2001), 2.3 (2003), 2.4 (2004), 2.5 (2006) y 2.6 (1 de octubre de 2008), que agregaron mejoras en bibliotecas estándar y rendimiento. Python 2 se convirtió en un estándar en la industria, con una base de código masiva en proyectos científicos, web y automatización.

- **2006: Inicio de la Planificación de Python 3**
  El 5 de abril de 2006, se creó el PEP 3000 (Python Enhancement Proposal), que marcó el comienzo formal de Python 3.0. Guido van Rossum, como Benevolent Dictator for Life (BDFL), impulsó esta versión para "limpiar" el lenguaje, eliminando duplicidades y corrigiendo problemas como el manejo inconsistente de Unicode y cadenas de texto. El PEP 3000 enfatizaba que no se requeriría compatibilidad total con Python 2.6.

- **2008: Lanzamiento de Python 3.0 y Comienzo de la Coexistencia**
  Python 3.0 se liberó el 3 de diciembre de 2008, rompiendo deliberadamente la compatibilidad con Python 2 para introducir mejoras fundamentales, como el tratamiento de cadenas como Unicode por defecto y cambios en la sintaxis (por ejemplo, print como función en lugar de declaración). Al mismo tiempo, Python 2.6 incluyó características de Python 3 para facilitar la transición, como un modo de "warnings" que destacaba funcionalidades obsoletas.

- **2009-2010: Lanzamientos Paralelos y Extensión de Soporte**
  Python 3.1 (27 de junio de 2009) y Python 2.7 (3 de julio de 2010) se lanzaron en paralelo. Python 2.7 fue la última versión mayor de la serie 2.x, incorporando backports de Python 3.1 para ayudar en la migración. Inicialmente, se planeaba que Python 2 perdiera soporte en 2013, pero se extendió hasta 2020 debido a la lenta adopción.

- **2011-2019: Adopción Lenta y Mejoras Incrementales**
  Se lanzaron versiones como Python 3.2 (2011), 3.3 (2012, que reintrodujo el prefijo 'u' para cadenas Unicode para reducir incompatibilidades), 3.4 (2014), 3.5 (2015, con async/await), hasta 3.8 (2019). Durante este período, la comunidad desarrolló

bibliotecas como Six y Python-Future para código "straddling" (compatible con ambas versiones). Proyectos grandes, como Dropbox, tardaron años en migrar debido a dependencias complejas.

- **2020: Fin de Vida de Python 2**
  El 1 de enero de 2020, Python 2 alcanzó su end-of-life (EOL), con la versión final 2.7.18 lanzada el 20 de abril de 2020. Esto culminó una transición de más de 13 años. Sin embargo, algunos sistemas como Red Hat Enterprise Linux (RHEL), Maya y ArcGIS continuaron usando Python 2 en 2023, ignorando el EOL. Pip 21.0, en enero de 2021, eliminó soporte para Python 2.

- **2021-2025: Consolidación de Python 3 y Legado**
  Python 3 continuó evolucionando con lanzamientos como 3.9 (2020), 3.10 (2021), 3.11 (2022), 3.12 (2023), 3.13 (2024) y la candidata 3.14 (2025). Para 2025, Python 3 es el estándar, impulsado por el auge del machine learning, aunque vulnerabilidades de seguridad en 2021-2024 afectaron incluso a versiones legacy como 2.7.

Proceso de Transición: Cómo se Realizó

La transición se planeó como un proceso gradual, pero en la práctica fue "todo o nada" para muchos proyectos. Se manejó a través de:

- **Herramientas de Migración**: El utility 2to3 automatizaba la conversión de código Python 2 a 3, aunque no manejaba todos los casos, especialmente Unicode. El módulo __future__ permitía importar sintaxis de Python 3 en Python 2 (ej. from __future__ import print_function), facilitando código poliglota.

- **Estrategias Comunitarias**: Recomendaciones iniciales incluían mantener código separado para cada versión, pero para 2012 se pasó a bases únicas con módulos de compatibilidad. Pruebas unitarias con cobertura alta y flags como -3 en Python 2 ayudaban a identificar depreciaciones.

- **Extensión de Soporte**: El EOL de Python 2 se pospuso varias veces para dar tiempo a la migración, reconociendo la resistencia.

A pesar de esto, la transición no fue incremental; requería actualizar todas las dependencias transitivas simultáneamente, lo que aumentó costos y riesgos.

Problemas Principales

La transición fue calificada como "horrorosamente mala" por muchos, generando un costo industrial estimado en más de $100 mil millones. Los problemas clave incluyeron:

- **Incompatibilidad Hacia Atrás**: Cambios como print convirtiéndose en función (PEP-3105), división entera retornando floats (5/2 = 2.5 en lugar de 2), y cadenas como Unicode por defecto (separando bytes y strings, PEP-358). Otros: remoción de métodos redundantes en diccionarios (PEP-3106), reorganización de la biblioteca estándar (PEP-3108), y cambios en excepciones (PEP-3109/3110).

- **Adopción Lenta y Fractura Comunitaria**: Python 3 era más lento en versiones tempranas, y la herramienta de conversión fallaba en aspectos clave como Unicode. Grandes codebases tardaron años en migrar, y algunos usuarios abandonaron Python. Vulnerabilidades de seguridad persistieron en Python 2 post-EOL.

- **Complejidad Técnica**: __future__ no producía comportamientos idénticos (ej. unicode_literals), y bibliotecas como six agregaban complejidad a las pruebas. No se justificaban las rupturas con nuevas capacidades suficientes.

Esto fracturó la comunidad, con algunos jurando no volver a usar Python, y contribuyó al retiro de van Rossum como BDFL en 2018.

Desarrolladores y Posturas: Quiénes Apostaron por Cada Versión

- **Guido van Rossum**: Como creador y BDFL hasta 2018, apostó fuertemente por Python 3 para "reinventar" el lenguaje, declarando "This is not your father's Python". En 2018, admitió en PyCascades que la transición fue un error en algunos aspectos. Su liderazgo fue pivotal, pero la controversia (incluyendo PEP 572) lo llevó a renunciar.
- **Comunidad Pro-Python 3**: Desarrolladores como los maintainers de bibliotecas (ej. autores de Six y Python-Future) apostaron por la transición, creando herramientas para facilitar la adopción. Proyectos modernos en ML y data science abrazaron Python 3 rápidamente.
- **Resistencia Pro-Python 2**: Muchos en la comunidad, como David A. Wheeler y commenters en LWN, criticaron la transición como "botched" y abogaron por cambios graduales. Usuarios como linuxrocks123 se negaron a migrar, prefiriendo compilar Python 2 desde fuente. Maintainers de codebases legacy (ej. Dropbox) se mantuvieron en Python 2 por dependencias. Algunos, como sfeam, vieron Python 2.7 como una versión "estable" y prefirieron portar código a otros lenguajes.

En resumen, la coexistencia de Python 2.x y 3.x fue un período de innovación y conflicto que fortaleció Python a largo plazo, convirtiéndolo en uno de los lenguajes más populares hoy, pese a las cicatrices comunitarias. Python 3 sobrevivió donde otros (como Perl 6) fallaron, gracias a su enfoque en la limpieza y modernidad.