Guido van Rossum：可讀性、實用主義與 BDFL 裁決

Guido van Rossum: Readability, Pragmatism, and BDFL Judgment

v1.0 / 2026-07-30 / Neo.K / 公開版／第三部設計師個案正式研究

https://thisoneisneok.com/html/papers/pldst-017.html

英文名稱： Guido van Rossum: Readability, Pragmatism, and BDFL Judgment

系列： Programming Language Designer Style Taxonomy（PLDST）

文件編號： PLDST-017

版本： v1.0

日期： 2026-07-30

作者： Neo.K

文件狀態： 公開版／第三部設計師個案正式研究

SECTION

摘要

Guido van Rossum 常被描述為 Python 的創造者、可讀性語言的設計者，以及長期擔任 Benevolent Dictator For Life（BDFL）的開源領導者。這些標籤指出了三個重要事實，卻也容易造成三種誤讀：

- 把 Python 的可讀性簡化成縮排語法；
- 把 Python 的實用主義誤解成沒有一致原則；
- 把 BDFL 描述成個人任意決策，而忽略提案、實作、社群說服與最終責任之間的制度配置。

Python 源自 Guido 在 CWI 的 ABC 經驗、Amoeba 分散式作業系統工具需求、Modula-3 的模組與例外觀念，以及 Unix／C 的可擴展實作環境。ABC 強調清楚、教學和精練表面，但較封閉，難以連接作業系統與外部程式庫；Python 保留可讀性與互動性，同時加入模組、例外、類別、可擴展 C 介面和真實系統腳本能力。

Guido 的核心設計不是追求最小語法，而是建立一套讓程式「適合被其他人閱讀、修改、重用和維護」的預設。官方前言明確把 Readability 視為 Python 特徵的共同方向，並將 ABC 的清楚價值與 Modula-3 的語言影響列為主要來源。[R2] PEP 20 所整理的 Python 指導原則亦同時包含：

- Readability counts；
- Explicit is better than implicit；
- Simple is better than complex；
- Although practicality beats purity；
- In the face of ambiguity, refuse the temptation to guess；
- There should be one—and preferably only one—obvious way to do it。[R4]

這些原則並非完全一致。明示可能增加樣板；唯一明顯做法可能與多範式衝突；實用性又可能突破簡潔。Guido 的真正作用，正是對這些衝突作出品味判斷，而不是套用單一演算法。

本文將 Guido 的設計生涯分成六個相位：

- ABC 吸收與反省期：可讀教學語言的優點與封閉系統缺陷；
- Amoeba 實用工具期：互動腳本、例外、模組及 C 擴展；
- 早期社群與 BDFL 形成期：創始者最終裁決、開放貢獻與一致品味；
- PEP 制度化期：把口頭領導轉成公開提案、理由、實作及歷史記錄；
- Python 3 修正歷史期：為語言一致性與 Unicode／資料模型修正承擔不相容遷移；
- PEP 572 與後 BDFL 治理期：個人裁決的社會成本顯現，權力轉移至選舉產生的 Steering Council。

本文核心判斷為：

> 【Guido 的風格不是「少功能」，
>
> 而是讓常見、正確、可讀的路徑成為阻力最低的路徑。】

其深層配置可表示為：

> 【可讀表面
> +
> 少量明顯慣例
> +
> 動態實用能力
> +
> 可擴展實作
> +
> 集中品味裁決】

但 Python 的歷史也顯示這種模型的限制。創始者的品味可防止委員會式功能聯合體，卻會把高爭議決策的情緒、說服和責任集中到個人；Python 3 證明 Guido 願意為一致性破壞相容，PEP 572 則證明單人最終裁決在全球社群中具有不可持續的社會成本。

因此，Guido 的後期最重要設計可能不再是某項語法，而是接受：

> 【語言風格可以制度化，
>
> 但創始者本人不必永久成為制度。】

關鍵詞： Guido van Rossum、Python、可讀性、實用主義、BDFL、PEP、Python 3、Assignment Expressions、治理、PLDST

SECTION

一、本文研究範圍

本文主要分析：

- ABC 對 Python 的影響；
- Amoeba 系統工具需求；
- 1989–1991 年 Python 起源；
- CPython 的早期架構；

- Python 社群、PEP 與 BDFL；
- Python 2／3 的歷史修正；
- PEP 572 及 Guido 退任；
- PEP 13 的後 BDFL 制度。

本文不把下列成果全部歸於 Guido：

- CPython 所有模組；
- Standard Library；
- NumPy、Django、PyTorch 等生態；
- 所有 PEP；
- Python 3 的所有具體功能；
- 現代 Steering Council 決策；
- PyPy、Jython、IronPython 等實作。

SECTION

二、Python 的創始權重

Guido 對下列事項具有高度直接權重：

- 原始問題設定；
- 第一版語言和 Interpreter；
- 早期語法與資料模型；
- C 擴展介面；
- 模組、例外、類別等整體組合；
- 長期 BDFL 裁決；

- Python 3 的總體方向；
- 多項個別 PEP。

因此稱其為 Python 創造者和主要早期作者合理。

SECTION

三、Python 是社群語言

Python 官方 License 歷史明確寫道：Python 由 Guido 在 CWI 建立，他是 Principal author，但語言包含許多其他人的貢獻。[R5]

重要共同主體包括：

- ABC 團隊；
- CWI 與 Amoeba 研究者；
- Tim Peters；
- Barry Warsaw；
- Paul Prescod；
- Greg Stein；
- Fred Drake；
- Marc-André Lemburg；
- Brett Cannon；
- Raymond Hettinger；
- Nick Coghlan；
- Core developers；
- PEP authors；
- PSF；

- 實作者和使用者。

  PEP 20 本身即由 Tim Peters 撰寫，用 19 條格言濃縮 Guido 的設計原則，而不是 Guido 自己起草的憲法。[R4]

SECTION

四、ABC 的優點

ABC 追求：

- 初學者可學；
- 互動式；
- 高階資料型別；
- 清楚語法；
- 少量概念；
- 不暴露機器細節。

  Guido 在 Python 官方前言中指出，Python 對可讀性的重視直接反映 ABC 的哲學。[R2]

SECTION

五、ABC 的不足

ABC 的問題包括：

- 封閉環境；
- 難以接入 OS；
- 難以擴展 C Library；
- 不適合作為一般 Unix 工具語言；
- 缺少真實大型系統所需的模組和整合能力。

因此 Guido 沒有直接延續 ABC，而是保留其人本表面，移除其封閉邊界。

SECTION

六、第一個深層風格：保留體驗，重建邊界

Python 對 ABC 的繼承可表示為：

> Keep(
> Readability,
> Interactivity,
> HighLevelData,
> TeachingClarity
>)

> Replace(
> ClosedWorld,
> LimitedInterop,
> WeakSystemAccess
>)

這是實用主義，不是無原則折衷。

SECTION

七、Amoeba 的工具問題

Guido 在 CWI 參與 Amoeba 分散式作業系統時，需要快速建立：

- 系統管理工具；
- 測試；
- 檔案和程序操作；
- 分散式服務介面；

- 可由非 C 專家修改的程式。

Shell 太弱，C 太慢於開發，ABC 又難以接入系統。

SECTION

八、Python 的問題定位

Python 被設計為：

比 Shell 更適合大型程式
比 C 更快速開發
比 ABC 更可連接現實系統
這種中間定位成為長期優勢。

SECTION

九、Modula-3 影響

Guido 在官方前言中把 Modula-3 稱為除 ABC 外的主要影響來源之一。[R2]

重要方向包括：

- Module；
- Exception；
- Clean structured design；
- Object-oriented ideas；
- 清楚 Interface。

Python 沒有複製 Modula-3 的靜態型別，而是吸收部分結構觀念。

SECTION

十、C 擴展與雙層責任

Python 讓：

- 常見控制和資料操作以高階動態語言完成；
- 性能敏感或系統專屬部分以 C 實作；
- 第三方模組可進入 Interpreter；
- Python 作為 Glue language。

  複雜度配置為：

> C_(application development)downarrow

> C_(interpreter)
> +
> C_(extension boundary)
> uparrow

SECTION

十一、縮排是結構，不只是美學

Python 使用縮排表達 Block，使：

- 視覺結構與語法結構一致；
- 不允許花括號與縮排互相欺騙；
- Code review 更直接；
- 省略 End marker。

  但可讀性並不只來自縮排，還包括：

- 命名；
- Statement／Expression 分工；
- 少量特殊符號；
- 一致 Protocol；

- 清楚 Exception；
- Module；
- Standard Library 慣例。

SECTION

十二、程式首先是給人讀的

Guido 的設計假設：

> ReadFrequency
> >
> WriteFrequency

尤其在：

- 開源；
- 團隊；
- 維護；
- 教學；
- Library；
- 長期系統。

因此語言不能只優化輸入速度。

SECTION

十三、可讀性不是冗長

Python 經常允許：

- Dynamic typing；

- Type inference by runtime；
- List comprehension；
- Iterator；
- Context manager；
- Decorator；
- Generator。

這些都能縮短程式。

關鍵不是字數，而是：

> 省略後是否仍能從局部結構清楚推斷意圖？

SECTION

十四、明示與實用的衝突

PEP 20 同時說：

Explicit is better than implicit
Although practicality beats purity
Python 因此仍有：

- Duck typing；
- Implicit protocol；
- Truthiness；
- Context management；
- Descriptor；
- Import behavior；
- Dynamic attribute lookup。

Guido 並非拒絕隱式，而是拒絕不可預測且難解釋的隱式。

SECTION

十五、唯一明顯做法的真正含義

「最好只有一個明顯做法」不是：

- 每個問題只能有一個 Algorithm；
- 禁止多範式；
- Library 不得競爭；
- 語言永不增加替代語法。

較合理的解讀是：

> 對常見語言動作，避免讓使用者在多套等價且互不相容的核心構造間選擇。

SECTION

十六、BDFL 的功能

Python 早期需要有人：

- 接受或拒絕功能；
- 維持整體風格；
- 解決無法共識的爭議；
- 承擔最後決策；
- 防止功能聯合體；
- 對不完整提案說不。

Guido 的權力不是只來自稱號，而來自：

- 原始實作；

- 歷史知識；
- 社群信任；
- 持續工作；
- 最終維護責任。

SECTION

十七、BDFL 不是無程序獨裁

在 PEP 制度形成後，一項功能通常需要：

- 問題描述；
- 規格；
- 理由；
- 替代方案；
- 參考實作；
- 社群討論；
- BDFL 或 Delegate 決定。

Guido 可以裁決，但裁決需進入公共歷史。

SECTION

十八、品味裁決的優勢

集中品味可以：

- 保持一致；
- 快速結束爭議；
- 拒絕局部便利；

- 在資料不足時作整體判斷；
- 承認設計不能完全形式化。

SECTION

十九、品味裁決的風險

它也可能造成：

- Bus factor；
- 個人情緒成本；
- 社群依賴；
- 難以反駁的主觀判斷；
- 創始者偏誤；
- 權力接班問題；
- 反對者將技術爭議人格化。

SECTION

二十、PEP 的作用

PEP 不只收錄成功功能，也保存：

- Rejected；
- Deferred；
- Withdrawn；
- Process；
- Informational；
- Governance。

因此：

> Decision
> →
> PublicReasoning
> →
> HistoricalRecord

SECTION

二十一、PEP 讓作者和裁決者分離

許多 Python 功能由社群成員提出，Guido：

- 評估；
- 指定 Delegate；
- 要求修訂；
- 接受或拒絕。

這降低「Guido 設計所有功能」的錯誤歸因。

SECTION

二十二、PEP 20 的制度地位

PEP 20 是 Informational、Non-normative 指引，由 Tim Peters 撰寫。[R4]

因此它：

- 不是形式規格；
- 不是自動決策器；
- 格言彼此具有張力；
- 需要依案例解釋。

SECTION

二十三、為何需要 Python 3

Python 2 累積：

- Text／Bytes 混淆；
- 舊式類別；
- Integer division；
- print Statement；
- Iterator／Collection 不一致；
- Exception syntax；
- Library 歷史負擔。

  若只保持相容，語言核心難以徹底修正。

SECTION

二十四、Python 3 的責任判斷

Guido 接受：

> Coherence_(future)
> >
> Compatibility_(short-term)

但這不是輕率重寫，因為：

- Python 2 長期並行；
- 多項功能提前 Backport；
- 提供 2to3 等工具；

- 進行多年遷移；
- 最終生態承擔巨大成本。

SECTION

二十五、Python 3 的成功與代價

成功：

- Text／Bytes 更清楚；
- 語言規則更一致；
- 清除部分歷史化石；
- 建立長期現代基線。

代價：

- Library 分裂；
- 使用者延後升級；
- 教材混亂；
- 雙版本維護；
- Python 2 壽命超出早期預期。

這證明「更乾淨」不等於低遷移成本。

SECTION

二十六、Guido 並非絕對相容主義者

其風格是：

- 日常演化高度保守；
- 若歷史阻塞核心一致性，可建立明確重大斷點；

- 破壞必須服務長期公共語言，而非個人美學。

SECTION

二十七、Assignment Expressions 的問題

PEP 572 提議 NAME := expr，讓表達式中可命名中間結果。[R6]

其理由包括：

- 避免重複計算；
- 改善真實程式；
- 支援 Debug；
- 減少額外縮排；
- 命名子表達式。

SECTION

二十八、實用證據

PEP 572 強調真實程式，而不是只用玩具例子。

Guido 搜尋 Dropbox Code base，發現程式設計者為節省行數或縮排，會：

- 重複昂貴表達式；
- 提前執行不必要操作；
- 寫出不理想結構。[R6]

這是典型 Guido 風格：

> 語言設計不只根據理想 Style，也要觀察人實際如何逃避阻力。

SECTION

二十九、可讀性爭議

反對者認為：

- 賦值進入表達式會增加密度；
- 容易濫用；
- 破壞 Statement／Expression 分工；
- 產生類 C 風格錯誤。

PEP 最終以：

- 新運算子；
- 低優先序；
- 括號限制；
- Style guidance；
- 特定禁止位置；

控制風險。[R6]

SECTION

三十、從技術爭議到治理危機

Guido 在接受 PEP 572 後不久退任 BDFL。

PEP 13 的歷史記錄確認：他自 Python 創始起擔任 BDFL，至 2018 年 7 月退任；之後社群提出多套治理方案，最後建立五人 Steering Council。[R7]

不能簡化為：

Walrus operator 單獨導致退任
但該爭議集中呈現：

- 討論負荷；
- 社群敵意；
- 反覆說服；

- 個人最終責任；
- BDFL 接班不可持續。

SECTION

三十一、Steering Council

PEP 13 規定：

- 五人委員會；
- 由 Core team 選舉；
- 維持語言及 CPython 品質和穩定；
- 尋求共識；
- 作最終上訴機關；
- 具有廣泛權力，但應盡量少直接使用；
- 決策盡可能公開。[R7]

SECTION

三十二、權力由人格轉成程序

轉換為：

> PersonalLegitimacy
> →
> ElectedInstitution

最終權威不再依：

- 創始者身分；
- 終身稱號；

- 單一品味。

  而依：

- Core team 信任；
- 定期選舉；
- 多人表決；
- 利益衝突限制；
- No-confidence mechanism。

SECTION

三十三、Guido 的後期位置

Guido 之後仍：

- 參與 Python；
- 提出技術方向；
- 擔任 Core developer；
- 曾參選和加入 Steering Council；
- 推進性能與型別等工作。

  但不再使用 BDFL 身分作最後裁決。

  這是一種創始者退場而不退出知識共同體的模式。

SECTION

三十四、ABC 反省期

問題：清楚教學語言過於封閉
策略：保留可讀性，增加真實系統邊界

SECTION

三十五、Amoeba／早期 Python 期

問題：Shell 太弱、C 開發太慢
策略：動態高階語言＋C 擴展

SECTION

三十六、BDFL 社群期

問題：開放貢獻可能失去一致品味
策略：創始者最終裁決

SECTION

三十七、PEP 制度期

問題：口頭決策無法支撐大型社群
策略：公開提案、替代與歷史記錄

SECTION

三十八、Python 3 修正期

問題：歷史相容阻塞核心一致性
策略：重大版本斷點與長期遷移

SECTION

三十九、後 BDFL 期

問題：個人裁決的社會成本不可持續
策略：選舉 Steering Council

SECTION

四十、問題 framing

Guido 的核心問題是：

> 如何讓一般程式設計者快速寫出真實系統程式，同時讓後來的讀者不必重建作者腦中的隱藏規則？

SECTION

四十一、價值優先序

> V_(Guido)
> ≈
> (
> Readability,
> Practicality,
> Consistency,
> Approachability,
> Extensibility,
> Maintainability,
> CommunityUsability
>)

SECTION

四十二、核心—擴張偏好

偏好：

- 小而清楚語法；
- Protocol；
- Standard Library；
- C Extension；
- PEP 漸進增加；
- 拒絕多套等價核心機制。

SECTION

四十三、顯式—推導偏好

偏好：

- 名稱清楚；
- Block 明示於縮排；
- Ambiguity 時拒絕猜測；
- 允許動態 Protocol 與 Duck typing；
- 後期接受 Optional typing，不使 Runtime 強制靜態化。

SECTION

四十四、效率—可讀性偏好

Python 優先：

- 人類開發速度；
- 清楚意圖；
- 快速組合。

性能可由：

- C Extension；
- Interpreter improvement；
- Alternative implementation；
- Vectorized Library；

補足。

SECTION

四十五、安全—自由偏好

Python 提供：

- Memory-managed runtime；
- Exception；
- 高階資料；
- 一般安全預設。

  但動態執行、Monkey patch、Native extension、eval 等仍保留高自由。

SECTION

四十六、相容性偏好

日常保守，但願意在稀有重大版本中修正歷史。

Python 3 是最強反例，證明 Guido 並非「相容永遠高於一致」。

SECTION

四十七、治理偏好

早期：

- Benevolent central judgment。

  中期：

- Central judgment＋PEP procedure。

  後期：

- Elected council＋Delegated decisions＋Public process。

SECTION

四十八、Python 不是只有一種做法

語言中仍有：

- Class／Function；
- Comprehension／Loop；
- Format styles；
- Async models；
- Error handling alternatives；
- 多種 Packaging tool。

PEP 20 是方向，不是現況描述。

SECTION

四十九、可讀性具有社群與工具前提

Python 的動態特徵可能使：

- 大型程式型別不明；
- Runtime Error 延後；
- Monkey patch 非局部；
- Decorator／Metaclass 難追蹤。

Optional typing 和工具是後期補強。

SECTION

五十、Python 3 不是純成功敘事

它產生真實長期分裂。評價須同時保存：

- 核心修正收益；
- 生態遷移代價；
- 規劃低估；
- 社群投入。

SECTION

五十一、BDFL 不是所有社群都可複製

它需要：

- 創始者能力；
- 長期信任；
- 持續投入；
- 願意承擔情緒成本；
- 合法退場。

缺一項就可能成為任意權力。

SECTION

五十二、Guido 的後期回顧有創始者視角

Python History Blog 和訪談極重要，但需與：

- PEP；
- Commit；
- Release；
- 其他 Core developer；

- 治理文件；

  交叉校對。

  時期 問題 決策 複雜度去向 風格

  1980s ABC 清楚但封閉 保留可讀性、開放 OS／C Interpreter／FFI 實用重建

  1989–91 Shell 與 C 中間缺口 Python Runtime 高階腳本

  1990s 社群擴大 BDFL 裁決 創始者 品味集中

  2000s 決策需可追蹤 PEP 公開程序 制度記憶

  2008 歷史不一致 Python 3 生態遷移 原則修正

  2018 Assignment expression 爭議 接受受限 := 語言／Style 真實程式實用主義

  2018–19 BDFL 不可持續 Steering Council 選舉制度 權力退場

SECTION

五十三、主要原型

Guido van Rossum 同時屬於：

- 可讀實用主義語言設計者；
- 常用路徑塑形者；
- 創始者品味裁決者；
- PEP 制度建設者；
- 可退場的開源治理者。

SECTION

五十四、不適合的簡單標籤

不應只稱：

縮排語言發明者

簡單語言設計者
一種做法教條者
永久獨裁者
Walrus operator 作者
較精確的描述是：

> 以可讀性和常見使用行為塑造語言表面，透過 BDFL 品味維持一致，又在個人裁決不可持續時將語言交還制度的設計者。

SECTION

五十五、最重要的連續性

ABC 到 Python 3 的共同方向：

> 【讓程式結構與人類可理解結構一致】

SECTION

五十六、最重要的不連續性

Guido 願意在 Python 3：

破壞來源相容
以換取：

未來語義一致

SECTION

五十七、最重要的治理修正

由：

語言需要一個最終品味
修正為：

語言需要最終決策，但不必永遠由同一個人承擔
Guido van Rossum 的設計價值不只在 Python 看起來像偽程式碼，也不只在縮排。

他建立的是一套完整配置：

- 語言表面優先服務閱讀；
- 動態模型優先服務快速實作；
- C 邊界服務性能和系統整合；
- Standard Library 服務常見任務；
- PEP 保存公共理由；
- BDFL 維持長期風格；
- Python 3 提供修正歷史的例外；
- Steering Council 讓創始者權力可以退場。

本文對 Guido 的 PLDST 判定為：

> 【Readable Pragmatic Language Designer
> →
> BDFL Taste Arbiter
> →
> Post-Founder Governance Transitioner】

其核心優勢是：

- 可讀性具有跨語法、資料、錯誤及模組的一致方向；
- 常見工作阻力低；
- 動態語言與 C 生態互補；
- PEP 形成高品質制度記憶；
- 創始者品味能拒絕局部功能堆疊；
- 最終願意將權力制度化。

其核心代價是：

- 動態性增加大型系統推理成本；
- 「可讀」帶有共同體慣例；
- BDFL 集中情緒及政治責任；
- Python 3 造成長期生態分裂；
- 實用例外會逐步增加語言表面；
- 全球社群已不可能只依創始者個人直覺治理。

  最終原則為：

> 【可讀性不是外觀
>
> ∧
> 實用主義不是無原則
>
> ∧
> 最終裁決不是終身人格依賴】

Guido 的歷史最值得保留的，不只是「程式應容易閱讀」，而是：

> 語言設計者必須觀察真實程式設計者如何工作，讓良好行為成為自然行為；當個人品味無法再承擔全球共同體時，還必須設計一個能繼承品味、卻不繼承終身權力的制度。
>
> 人物：Guido van Rossum
> 主要語言／制度：Python、CPython、PEP、BDFL／Steering Council transition
> 核心時期：1989–2019
> 主要問題：清楚語言與真實系統能力分裂
> 主要策略：可讀語法、動態資料、模組、例外、C 擴展
> 複雜度去向：Interpreter、Library、PEP、社群治理
> 責任去向：常見路徑由語言塑形，最終風格由治理裁決
> 主要保護對象：一般程式設計者、讀者、Library 使用者
> 主要限制：動態推理、BDFL 負荷、Python 3 遷移、品味主觀性
> 歸因信心：高
> [R1] Guido van Rossum, The History of Python essays, 2009–2018 archive.
>
> — Python 功能、ABC、Indentation、Class、Functional features 及社群歷史。

[R2] Guido van Rossum, “Foreword for Programming Python,” Python.org.

— Readability、ABC、Modula-3 及 Python 的重用方向。

[R3] Charles Severance, “Guido van Rossum: The Early Years of Python,” Computer, 2015, based on interviews with Guido.

— CWI、Amoeba、1989 起源、第一版和開源授權。

[R4] Tim Peters, “PEP 20 – The Zen of Python.”

— Python 設計格言、可讀性、明示、實用及拒絕猜測；Informational、非規範性。

[R5] Python official License and History notices.

— CWI 起源、Guido principal author 及多方貢獻。

[R6] Chris Angelico, Tim Peters and Guido van Rossum, “PEP 572 – Assignment Expressions.”

— 真實程式證據、Named expression、限制位置及 Style guidance。

[R7] Python core team and community, “PEP 13 – Python Language Governance”; PEP 8000／8016.

— BDFL 退任、五人 Steering Council、選舉、權力及共識原則。

[R8] Python Enhancement Proposal index and relevant Python 3 PEPs, including PEP 3100.

— Python 3 不相容修正及多作者制度。

[R9] Computer History Museum, Guido van Rossum Fellow profile and oral-history materials.

— Python 創建、社群領導及長期演化。

[T-A] ABC reflection phase
[T-M] Amoeba／early Python phase
[T-B] BDFL community phase
[T-P] PEP institutional phase
[T-3] Python 3 correction phase
[T-G] Post-BDFL governance phase
[S-R] Readability
[S-P] Pragmatism
[S-O] Obvious-path design
[S-E] Extensibility
[S-B] BDFL taste arbitration
[S-I] Institutional succession

附錄 D　第二輪史實與歸因校對紀錄

D.1 Python 起源與「聖誕假期」敘事

第二輪重新核對 Python 官方歷史、Guido 的歷史文章及 2015 年訪談：

- Python 的第一批工作確實在 1989 年末開始；
- 但它不是一個假期內突然完成的語言；
- Guido 在此之前已累積數年 ABC、Interpreter、CWI 和 Amoeba 經驗；
- 第一個公開版本於 1991 年出現；
- Python 官方資料將它描述為 ABC 的後繼，但也有 Modula-3、C／Unix、Smalltalk 等影響。

本文因此使用「1989–1991 起源期」，而不採用單一假期天才神話。

D.2 Readability 與 Indentation

已重新核對 Python.org 的 Guido 前言：

- Guido 明確說 Python 的縮排直接來自 ABC；
- 他認為縮排同時減少 Visual clutter、縮短程式，並限制格式自由，使不同作者程式更一致；
- 他又明確表示，可讀性的重視不是偶然，而是可重用程式的必要條件；
- 除 ABC 外，Modula-3 被他稱為主要影響來源。

因此本文沒有把 Readability 只歸結為縮排，而把它定位成跨語法、模組、例外和重用的設計方向。

D.3 PEP 20 的作者與規範地位

第二輪直接核對 PEP 20：

- 作者是 Tim Peters；
- 建立於 2004 年；
- 狀態為 Active；
- 類型為 Informational；
- PEP 自稱是 Tim Peters 對 BDFL 指導原則的濃縮；
- 它不是 Python 語言的 Normative specification。

本文因此把「Zen of Python」視為共同體對 Guido 品味的文化編碼，而不是 Guido 單人發布的形式憲法。

D.4 PEP 572 的作者、證據與限制

第二輪直接核對 PEP 572：

- 作者是 Chris Angelico、Tim Peters 和 Guido van Rossum；
- 目標版本為 Python 3.8；
- 最終狀態為 Final；
- PEP 強調真實程式碼，而非只用玩具例子；
- Guido 搜尋 Dropbox Code base 的觀察，是支持提案的經驗材料之一；
- 最終語法禁止或限制多個可能造成視覺歧義的位置；
- Assignment Expression 並未把一般 Assignment Statement 直接變成可任意使用的 Expression。

本文因此沒有把 Walrus Operator 歸為 Guido 單人設計，也沒有把它寫成不受限制的 C 式賦值表達式。

D.5 BDFL 退任的因果邊界

PEP 13 確認：

- Guido 自 Python 創始起擔任 BDFL；
- 於 2018 年 7 月退任；
- 之後社群提出並投票選擇多套治理方案；
- Steering Council 模型最終成為正式制度。

PEP 572 爭議與退任時間緊密相連，但正式文件不支持「Walrus 是唯一原因」這種單因敘事。

本文採用：

PEP 572 是治理壓力的集中事件

≠

它單獨解釋所有退任動機

D.6 Python 3 的多作者性

Python 3 的變更分布在大量 PEP 和實作工作中：

- `print` Function；
- Text／Bytes；
- Iterator；
- Exception；
- Class model；
- I/O；
- Standard Library；
- Annotation；
- Numeric behavior。

Guido 對重大方向和最終裁決具有高權重，但各功能有獨立 PEP 作者、實作者和 Reviewers。

本文因此把「Python 3 修正歷史」歸為 Guido 主導方向與整個 Core community 的共同工程。

D.7 2026 年 Python 治理狀態

截至 2026 年 7 月重新核對 PEP 13：

- Python 仍由五人 Steering Council 治理；
- Council 由 Core team 定期選舉；
- 其權力廣泛，但文件要求盡量少直接使用，優先尋求共識與建立標準程序；
- Council 是其他方法失敗後的 Final appeal；
- PEP 13 包含利益衝突、No-confidence 和 Core-team membership 機制；
- Guido 不再具有 BDFL 制度權力。

目前官方 Python 穩定版主線為 Python 3.14，Python.org 在校對時列出 3.14.6；這是時間敏感資訊，只記於校對附錄，不作人物風格核心證據。

D.8 「一個明顯做法」的解釋邊界

本文將 PEP 20 的格言解釋為：

- 減少核心層級的等價機制競爭；
- 讓常見操作有可辨識慣用法。

它不表示 Python 生態真的只有一個 Framework、Algorithm、Packaging tool 或 Error style。

這是 PLDST 分析推論，信心為中高。

D.9 PLDST 原型邊界

下列名稱是本文分析原型，不是 Guido 自稱的正式學派：

可讀實用主義語言設計者

常用路徑塑形者

BDFL 品味裁決者

可退場的開源治理者

其中最後一項描述的是「從個人最終權力轉至可選舉制度」的歷史行動，不表示 Guido 單人設計了 PEP 13。