04

Ancient Code: The Indian Origins of Computer Science

Dr. Gnaaneshchandra N. Jani
Faculty, Computer science Department
Vivekanand College of Arts Ahmedabad, Gujarat

Abstract:

History books often tell us that Computer Science is a modern invention from the West, starting in the 20th century. However, if we look closely at ancient Indian Knowledge Systems (IKS), we find something surprising: a "computational" way of thinking that existed thousands of years ago. This paper looks at the striking similarities between ancient Indian methods and modern computer programming. We look at **Panini** (who created the logic for code), **Pingala** (who discovered binary numbers), and mathematicians like **Aryabhata** and **Bhaskara II** (who invented algorithms). We argue that these were not just calculation tools, but the actual ancestors of the software logic we use today.

----

1. Introduction:

The definition of a "computer" lies in the methodology used to solve problems, not in the silicon chip. Long before the advent of electricity, Indian scholars developed rigorous, rule-based systems to manage complex data in linguistics and astronomy.

While ancient Greek mathematics, such as Euclid's geometry, focused primarily on static proofs, ancient Indian mathematics was inherently **algorithmic**. It focused on the process (*karas*) of deriving a result step-by-step (Plofker, 2009). This procedural approach is the very essence of modern software. This paper explores four specific domains where ancient Indian scholarship prefigured modern computational concepts by centuries.

2. Panini: The Father of Code (c. 500 BCE):

The Sanskrit grammar known as the *Aṣṭādhyāyī*, authored by Pāṇini, is arguably the world's first formal system. It is not merely a textbook; it functions as a generative machine. Comprising approximately 4,000 rules (*sutras*), it functions like a computer program capable of generating every grammatically correct sentence in the Sanskrit language (Staal, 1965).

2.1 The Metalanguage (Code in Relation to Data):

Just as modern programmers use strict syntax in languages like Java or Python, Pāṇini established a formal structure:

- **Metalanguage:** Panini created a technical version of Sanskrit specifically to describe the rules of the language. This mirrors the distinction between the **source code** a developer writes and the **application** the user interacts with (Kiparsky, 2002).

- **Rewrite Rules:** His system utilizes logic equivalent to "Replace A with B if context C is present." In modern Computer Science theory, this is known as **Context-Sensitive Grammar**, a concept rediscovered by Noam Chomsky in the 1950s (Kak, 1987).

2.2 Recursion and Compression:

To ensure his rules could be memorized, Panini employed techniques essentially identical to modern memory optimization:

- **Inheritance (*Anuvṛtti*):** If a rule is defined once, it automatically applies to subsequent rules unless explicitly halted. In coding, this concept is known as **inheritance** or **lexical scoping**.

- **Data Compression (*Shiva Sutras*):** Rather than listing every vowel repeatedly, Pāṇini defined short keywords (such as '*ac*') to represent "all vowels." This is conceptually the same as defining an **Array** or a **Set** variable.

Table 1: *Panini* vs. Modern Coding:

Feature	Pāṇini's Sanskrit Grammar	Modern Coding (Java/Python)
Style	**Meta-Sanskrit:** A technical language to describe rules.	**Syntax:** Code is a formal language separate from the output.
Structure	**Sūtras:** Rules that replace one sound with another.	**Production Rules:** Logic used in compiler design.
Compression	**Shiva Sutras:** Grouping letters (e.g., 'ac' = vowels).	**Arrays:** Grouping data under one variable name.
Inheritance	**Anuvṛtti:** A rule applies to the lines below it.	**Scoping:** Variables pass data to sub-routines.

3. Pingala: The Origin of Binary (c. 300 BCE):

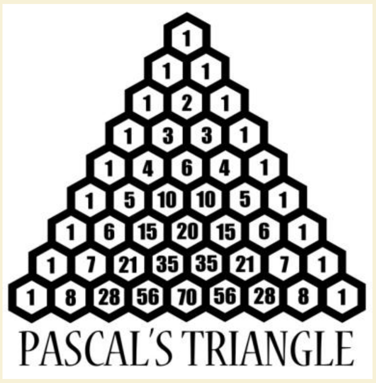
In his treatise *Chanda sastra* (Science of Meters), Acarya Pingala analyzed the rhythm of poetry. Through this study, he inadvertently established the mathematical foundation of modern digital computing: **Binary Code** (Van Nooten, 1993).

3.1 The Binary Mapping:

Computers understand two states: 0 and 1. Piṅgala classified poetic syllables into two states:

- **Laghu (Short beat):** Acts like **1**
- **Guru (Long beat):** Acts like **0**

3.2 Algorithms for Binary Conversion:



Piṅgala developed algorithms to calculate combinatorial patterns:

- **Nashtam:** A method to determine a specific meter's pattern using division by 2. This is mathematically identical to the **Decimal-to-Binary** conversion algorithm used today (Shah, 2013).
- **Meru Prastara:** He constructed a pyramid of numbers to calculate combinations. While known globally as **Pascal's Triangle**, Pingala utilized this structure centuries before Blaise Pascal.

Table 2: Pingala vs Digital Logic:

Concept	**Pingala's Method**	**Modern Computers**
Units	**Short / Long**	**1 (On) / 0 (Off)**
Math	**Prastara:** Calculating all rhythm patterns.	**Truth Tables:** Calculating logic gate inputs.
Conversion	**Nashtam:** Calculating the pattern from a number.	**Decimal-to-Binary:** Converting human numbers to code.
Visuals	**Meru Prastara:** The "Mountain" of numbers.	**Pascal's Triangle:** Used in probability.

4. Navya-Nyaya: Logic for AI (13th Century):

The *Navya-Nyāya* ("New Logic") school developed a linguistic system of absolute precision. In the 1980s, AI researchers noted that while natural languages like English are ambiguous ("messy") for computers, the structured logic of Sanskrit is ideal for **Knowledge Representation** in Artificial Intelligence (Briggs, 1985).

4.1 The Knowledge Triad:

Navya-Nyāya breaks every thought down into three parts, which is exactly how the modern **Semantic Web** (the technology behind Google Search) organizes data:

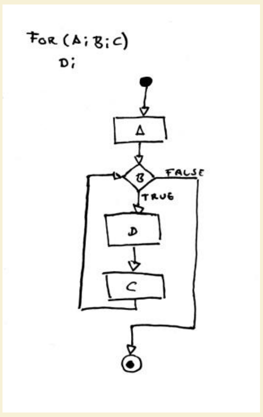
1. **Visheṣya (The Object):** e.g., "The pot"
2. **Prakāra (The Feature):** e.g., "Blue color"
3. **Samsarga (The Relation):** e.g., "Inherences in"
4. This "Triad" structure allows a computational agent to map relationships between entities without ambiguity (Matilal, 1968).

5. Ancient Algorithms: Encryption and Optimization:

5.1 Kuṭṭaka: The Ancestor of Encryption:

Aryabhata (499 CE) invented the *Kuṭṭaka* ("The Pulverizer") method to solve indeterminate linear equations ($ax + by = c$).

- **The Logic:** It breaks numbers down recursively until a solution is found.
- **The Legacy:** This logic is mathematically identical to the **Extended Euclidean Algorithm**. This is the fundamental math used in **RSA Encryption** today—the protocol that secures internet passwords and banking transactions (Datta & Singh, 1962).



Aryabhata (499 CE) invented a method called *Kuṭṭaka* (The Pulverizer) to solve complex algebra equations.

- **The Logic:** It breaks numbers down recursively until a solution is found.
- **The Legacy:** This logic is mathematically identical to the **Extended Euclidean Algorithm**. This is the fundamental math used in **RSA Encryption** today, the protocol that secures internet passwords and banking transactions (Datta & Singh, 1962).

5.2 Chakravakam: Finding the Best Path:

Bhāskara II (12th Century) refined a method called *Chakravakam* (Cyclic method).

- **The Innovation:** Instead of brute-force guessing, this algorithm makes an intelligent estimate to minimize error, then loops back to refine that guess.
- **The Legacy:** This represents an early **Optimization Algorithm**, conceptually similar to the gradient descent methods used in modern Machine Learning to "learn" the optimal solution (Plofker, 2009).

Table 3: Ancient Algorithms vs. Modern Tech:

Ancient System	**The Innovation**	**Modern Parallel**
Navya-Nyāya	**The Knowledge Triad:** Object-Feature-Relation.	**Knowledge Graphs:** How AI understands data connections.
Kuṭṭaka	**The Pulverizer:** A recursive math trick.	**RSA Encryption:** Internet security protocols.
Chakravāla	**Cyclic Method:** Looping to find the best answer.	**Machine Learning:** Algorithms that improve over time.

6. Conclusion:

The history of Computer Science is not a linear progression originating solely in the West; it is a tapestry woven from diverse global threads. India's contributions to generative grammar, binary systems, logic, and algorithms represent the "source code" of computational thought. By understanding these ancient roots, we recognize that Indian scholars were effectively "coding"—designing algorithms and processing information—long before the hardware existed to run them.

References:

1. **Briggs, R.** (1985). "Knowledge Representation in Sanskrit and Artificial Intelligence." *AI Magazine*, 6(1), 32.
2. **Cardona, G.** (1988). *Pāṇini: His Work and Its Traditions*. Motilal Banarsidass.
3. **Datta, B., & Singh, A. N.** (1962). *History of Hindu Mathematics: A Source Book*. Asia Publishing House.
4. **Ingerman, P. Z.** (1967). "Pāṇini-Backus Form Suggested." *Communications of the ACM*, 10(3), 137.
5. **Joseph, G. G.** (2011). *The Crest of the Peacock: Non-European Roots of Mathematics*. Princeton University Press.
6. **Kak, S.** (1987). "The Paninian Approach to Natural Language Processing." *International Journal of Approximate Reasoning*, 1(1), 117-130.
7. **Kiparsky, P.** (2002). "On the Architecture of Pāṇini's Grammar." In *Indo-Iranian Perspectives on Computational Linguistics*.
8. **Knuth, D. E.** (2011). *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley Professional.
9. **Matilal, B. K.** (1968). *The Navya-Nyāya Doctrine of Negation*. Harvard University Press.
10. **Plofker, K.** (2009). *Mathematics in India*. Princeton University Press.
11. **Rao, T. R. N., & Kak, S.** (1998). *Computing Science in Ancient India*. Center for Advanced Computer Studies.
12. **Staal, F.** (1988). *Universals: Studies in Indian Logic and Linguistics*. University of Chicago Press.
13. **Van Nooten, B.** (1993). "Binary numbers in Indian antiquity." *Journal of Indian Philosophy*, 21(1), 31-50.