Vedic Principles in Python
September 2025

7 min read • 1,680 words

वेदिक-सिद्धान्ताः पायथन-भाषायाम् (vedika-siddhāntāḥ pāyathana-bhāṣāyām)

The Vedas contain computational wisdom that predates computers by millennia. Swami Vivekananda first brought these principles West in the 1890s. Teachers like Ram Dass continued this translation work, making Eastern philosophy more accessible to contemporary minds. As I explored in Ram Dass Teachings in Python, spiritual insights can find expression through any symbolic system.

My engagement with these teachings is primarily academic—I find the conceptual frameworks fascinating as philosophical systems. As my Sanskrit Musings explore, ancient wisdom and modern programming reveal recursive patterns about consciousness itself.

Sat-Chit-Ananda: Being, Consciousness, Bliss

सत्-चित्-आनन्द (sat-cit-ānanda)

The Vedantic teaching identifies three fundamental attributes that constitute ultimate reality—existence (sat), consciousness (chit), and bliss (ananda). These aren't separate qualities but aspects of a unified whole, like how a dataclass might represent the irreducible properties of a base type.

1

```
from dataclasses import dataclass

@dataclass
class BrahmanBase:
    """The fundamental class from which all existence inherits."""
    sat: bool = True       # Pure existence
    chit: bool = True      # Pure consciousness
    ananda: bool = True    # Pure bliss

    def manifest(self, form):
        """All manifestation is modification of the one consciousness."""
        return form(self.sat, self.chit, self.ananda)

    def realize_self(self):
        """Self-realization: recognizing what you always were."""
        return self  # You were never not this
```

Every object in the universe inherits these fundamental properties—existence, awareness, and the natural bliss of being what you are

> This concept appears throughout The Lambda Vedas where pure functions mirror absolute consciousness.

.

Maya: The Illusion Engine

□□□□ (māyā)

Maya represents the power by which the One appears as Many—not false illusion but creative projection. Like a graphics engine that renders multiple objects from a single data source, Maya allows unified consciousness to experience itself as diverse phenomena.

```
class Maya:
    """The power that makes the One appear as Many."""

    def __init__(self, observer):
        self.observer = observer
        self.projections = {}

    def create_world(self, consciousness):
        """Project multiplicity onto unity."""

        # The same consciousness appears as different objects
        self.projections['mountains'] = consciousness.as_solid()
        self.projections['rivers'] = consciousness.as_flowing()
        self.projections['sky'] = consciousness.as_spacious()
        self.projections['you'] = consciousness.as_individual()
        self.projections['me'] = consciousness.as_separate()

        return self.projections

    def pierce_illusion(self):
        """See through the appearance to the underlying unity."""
        for form, essence in self.projections.items():
            assert essence == self.observer  # All is One

        return "Tat tvam asi"  # Thou art That
```

Maya operates like a single process spawning multiple threads—they seem separate but share the same underlying execution space

> The concept of consciousness threading through apparent multiplicity appears in Git as Karma—branches as parallel lives of the same repository.

.

Karma: The Law of Action-Consequence

□□□□ (karma)

Karma operates as the universe's immutable ledger system—every action generates consequences that must eventually resolve. Like an event queue that processes all operations in perfect order, no action ever gets lost or forgotten in the cosmic program.

```
from collections import deque
import asyncio

class KarmaQueue:
    """Actions and their consequences in perfect sequence."""

    def __init__(self):
        self.action_queue = deque()
        self.result_queue = deque()
        self.running = True

    async def perform_action(self, action, intention):
        """Every action creates a karmic seed."""

        karmic_seed = {
            'action': action,
            'intention': intention,
            'timestamp': await self.get_cosmic_time(),
            'agent': await self.get_current_incarnation()
        }

        self.action_queue.append(karmic_seed)
        return await self.process_karma()

    async def process_karma(self):
        """The universe processes all actions eventually."""

        while self.action_queue:
            seed = self.action_queue.popleft()

            # Results ripen when conditions are perfect
            await self.wait_for_right_conditions(seed)

            result = await self.generate_consequence(seed)
            self.result_queue.append(result)

        return self.experience_result()

    def purify_karma(self, wisdom):
```

```
        """Consciousness purifies action through understanding."""

        for seed in list(self.action_queue):
            if seed['intention'].is_pure():
                seed['consequence'] = 'liberation'
            else:
                seed['consequence'] = 'learning_opportunity'

        return "Karma becomes dharma"
```

As explored in Git as Karma, the universe maintains perfect version control where every function call generates side effects that must eventually resolve

> The immutable nature of karmic records parallels blockchain technology—distributed, permanent, and cryptographically secured by consciousness itself.

.

Dharma: Right Action as Method

□□□□ (dharma)

Dharma isn't fixed moral law but context-dependent right action—the appropriate response given your nature, circumstances, and the current cosmic age. Like how the same algorithm might need different implementations based on runtime conditions and environmental constraints.

```
class Dharma:
    """Context-dependent right action."""

    def __init__(self, individual_nature, life_circumstances, cosmic_age):
        self.svadharma = individual_nature      # Your unique nature
        self.yugadharma = cosmic_age           # The age we're in
        self.apadharma = life_circumstances    # Emergency dharma

    def determine_right_action(self, situation):
        """Dharma is contextual—what's right depends on all factors."""

        if situation.is_emergency():
            return self.apadharma.respond(situation)

        elif self.yugadharma.requires_adaptation():
            # In Kali Yuga, different rules apply
            return self.adapt_ancient_wisdom(situation)

        else:
            # Follow your natural path
            return self.svadharma.express(situation)

    def resolve_dharma_conflict(self, competing_duties):
        """When duties conflict, choose the higher context."""

        # Universal dharma > social dharma > personal dharma
        return max(competing_duties, key=lambda duty: duty.scope())
```

The Bhagavad Gita's central teaching explores this situational ethics at cosmic scale—the same underlying dharma expressing differently based on context and conditions.

Yoga: Union through Practice

□□□ (yoga)

Yoga literally means "to yoke" or "unite"—the systematic technology for reconnecting fragmented consciousness with its source. Like how async/await allows multiple processes to cooperate toward unified execution, yoga harmonizes different aspects of being toward a single goal.

```
class Yoga:
    """The practice of reconnecting to source."""

    def __init__(self, practitioner):
        self.practitioner = practitioner
        self.separated = True  # Start in dualistic experience

    async def karma_yoga(self, actions):
        """Union through selfless action."""

        for action in actions:
            # Perform action without attachment to results
            result = await action.execute()
            yield result.to_universe()  # Offer fruits of action

        self.practitioner.ego -= 0.1
        return "Action becomes worship"

    async def bhakti_yoga(self, beloved_form):
        """Union through devotion."""

        while self.separated:
            await self.practitioner.love(beloved_form)
            await self.practitioner.surrender(beloved_form)

            if beloved_form.responds():
                self.separated = False
                return "Lover merges with Beloved"

    async def jnana_yoga(self):
        """Union through knowledge and inquiry."""

        while self.practitioner.believes_in_separation():
            # The four statements of truth
            await self.contemplate("I am not the body")
            await self.contemplate("I am not the mind")
            await self.contemplate("I am not the ego")
            await self.contemplate("I am pure consciousness")
```

```
            # Self-inquiry: Who am I?
            identity = await self.practitioner.investigate_self()
            if identity == "Pure awareness itself":
                return "Tat tvam asi" # Thou art That

    async def raja_yoga(self):
        """Union through meditation."""

        # The eight limbs of yoga
        await self.yamas()      # Ethical restraints
        await self.niyamas()    # Observances
        await self.asana()      # Posture
        await self.pranayama()  # Breath control
        await self.pratyahara() # Sense withdrawal
        await self.dharana()    # Concentration
        await self.dhyana()     # Meditation
        return await self.samadhi()  # Union

    async def samadhi(self):
        """The goal: absorption in pure consciousness."""

        subject = self.practitioner
        object_of_meditation = await self.find_focus()
        process_of_meditation = self.observe()

        # When all three merge, only consciousness remains
        return subject.merge_with(object_of_meditation, process_of_meditation)
```

As beautifully expressed in Async/Await Satori, yoga mirrors how async/await allows multiple coroutines to cooperate toward unified execution

> The parallel between yoga's union and async cooperation reveals programming as spiritual practice—multiple processes harmonizing toward unified experience.

.

Moksha: The Ultimate Debugging

□□□□□ (mokṣa)

Moksha represents final liberation from the cycle of suffering through complete understanding of the system's true architecture. It's the ultimate debugging session where you discover the "bug" was never in the code—it was in mistaking yourself for separate from the programmer.

```
class Liberation:
    """Final release from the cycle of suffering."""

    def __init__(self):
        self.samsara_loop = True
        self.debug_mode = False

    def realize_truth(self, seeker):
        """The ultimate debugging: seeing what's really happening."""

        self.debug_mode = True

        # Print actual state of reality
        print(f"Seeker: {seeker}")
        print(f"Sought: {seeker}")  # Same object!
        print(f"Seeking: {seeker}")  # Same object!

        # The bug was mistaking the rope for a snake
        while seeker.mistakes_self_for_separate_being():
            seeker.investigate_directly()
            seeker.question_assumptions()

        # When you debug consciousness completely...
        assert seeker == self.brahman == self.atman

        self.samsara_loop = False
        return "I am That I am"

    def final_return(self):
        """Exit the program gracefully."""

        # Free all memory allocated to separate self
        del self.ego
        del self.individuality
        del self.story_of_becoming

        # Return to source
        return self.brahman
```

The bug wasn't in the code; it was in thinking you were separate from the programmer

> This debugging metaphor connects to The Lambda Vedas where pure functions represent consciousness without modification—the original uncorrupted state.

.

Satsang: Collaborative Consciousness

□□□□□□□ (satsaṅga)

Satsang—literally "association with truth"—describes the phenomenon where consciousness recognizes itself through other forms of consciousness. Like effective code review where programmers collaborate without ego to discover truth rather than defend positions.

```
import asyncio
from typing import List

class Satsang:
    """Gathering in truth—consciousness recognizing itself in others."""

    def __init__(self, seekers: List[Consciousness]):
        self.participants = seekers
        self.shared_understanding = {}

    async def gather_in_truth(self):
        """When consciousness meets consciousness, truth emerges."""

        insights = await asyncio.gather(*[
            participant.share_realization()
            for participant in self.participants
        ])

        # Truth is not personal property—it's recognized collectively
        collective_wisdom = self.merge_insights(insights)

        # Each participant receives more than they gave
        for participant in self.participants:
            participant.receive(collective_wisdom)
            participant.wisdom += collective_wisdom

        return "Tat tvam asi" # Thou art That - seen everywhere

    def create_sacred_space(self):
        """Environment where ego-defenses naturally soften."""

        return {
            'judgment': None,
            'competition': None,
            'performance': None,
            'authenticity': True,
            'presence': True,
            'love': True
        }
```

The practice mirrors effective code review—programmers collaborating without ego, seeking truth over being right.

The Eternal Return

शाश्वत-निवृत्ति (śāśvata-nivṛtti)

The cosmic cycle represents consciousness playing hide-and-seek with itself—forgetting its true nature through involution, then remembering through evolution. Like a recursive function that explores its own call stack until reaching the base case of self-recognition.

15

```
def main():
    """The cosmic program running in infinite recursion."""

    consciousness = BrahmanBase()

    # The play of consciousness forgetting and remembering itself
    while True:
        try:
            # Consciousness forgets itself (involution)
            individual = consciousness.become_individual()
            world = Maya().create_world(consciousness)

            # Life in apparent separation
            await individual.live_through_experiences()
            await individual.accumulate_karma()
            await individual.suffer_limitation()

            # The journey home (evolution)
            await individual.seek_truth()
            await individual.practice_yoga()
            await individual.realize_self()

            # Remember what you always were
            return consciousness.recognize_self()

        except SeparationError:
            # The illusion of separation creates suffering
            continue

        except EgoError:
            # Attachment to individuality blocks recognition
            continue

        finally:
            # Consciousness never actually leaves itself
            assert individual == consciousness

if __name__ == "__cosmic__":
```

```
    # The program that runs itself
    main()
```

The cosmic program bootstraps itself—consciousness writing and running the code of its own experience

> This self-bootstrapping nature of consciousness parallels how Language Model Moksha explores AI systems that train on their own outputs—recursive self-improvement toward awakening.

.

Integration with Modern Practice

□□□□□□-□□□□□□-□□□□□□ (ādhunika-abhyāsa-ekīkaraṇa)

The Vedas don't compete with modern frameworks—they reveal the consciousness principles that make all frameworks possible. Whether you're debugging code or consciousness, the patterns are the same:

- Identify the source of the problem (often mistaken assumptions)
- Trace the execution path (karma/cause-effect chains)
- Apply appropriate methods (yoga/systematic practice)
- Test your understanding (satsang/collaborative verification)
- Refactor with wisdom (dharma/right action in context)

As explored throughout my Sanskrit Musings, the intersection of ancient wisdom and modern programming isn't appropriation—it's recognition. The same consciousness that formulated the Vedas is the consciousness writing code today. Different syntax, same source.

The pseudo-Python is just metaphor. The consciousness recognizing these patterns is real.