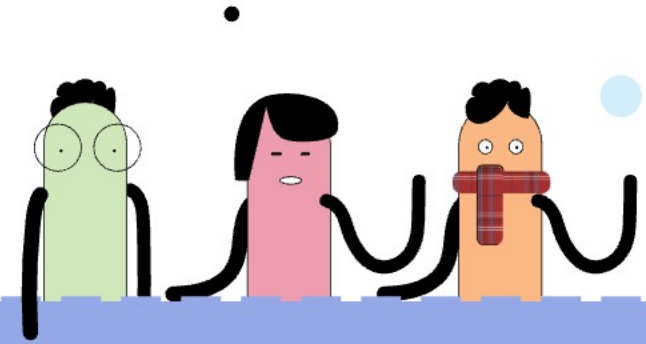


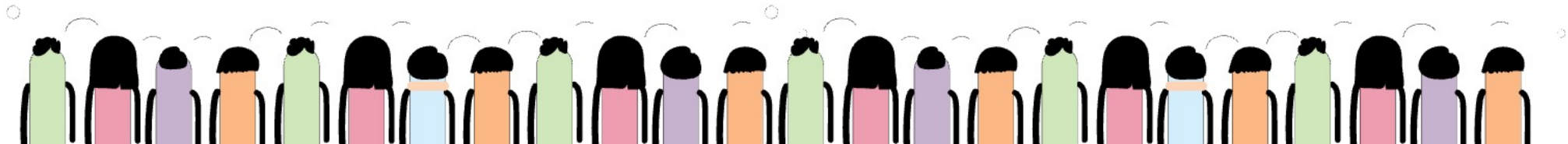
AWS

Coderrange

AWS

🎤 Inspiring Message for Job & Security Aspirants – Powered by Coderrange

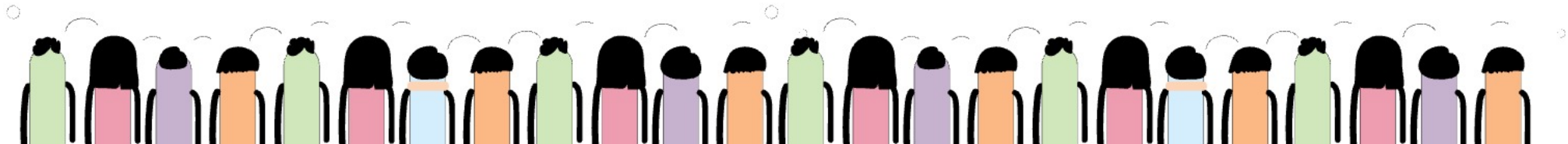
> **"Think about this...**
>
> 🌍 The world's most iconic names—**Netflix, NASA, Pfizer, BMW**—entrust AWS with their most sensitive data. Not just because it's fast, but because it's **secure**, **resilient**, and **smart**.
>
> 📦 Over **100 trillion files** live in AWS S3 today. That's nearly **10,000 files for every human on Earth.**
>
> 🧠 And every **second**, **100,000+ AWS Lambda functions** come alive—silently automating tasks, defending networks, and making life easier... all without a human lifting a finger.
>
> 👨‍💻 But here's the twist:
> **AWS isn't looking for more clouds. It's looking for more Cloud Guardians.**
> People who know how to **secure** it. **Automate** it. **Scale** it.



AWS Prequiste's

✅ 1. Basic IT & Networking Knowledge

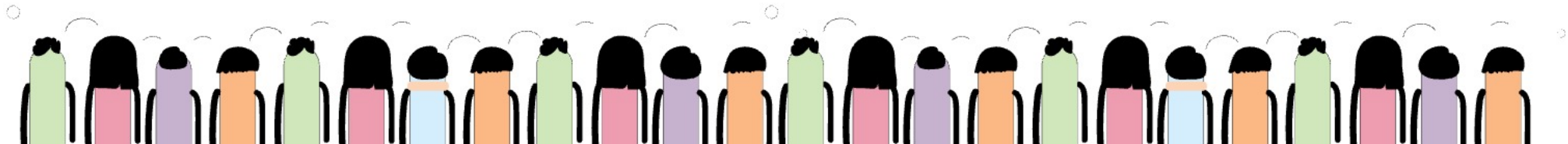
- **Operating Systems**: Linux & Windows basics (file system, terminal, services).
- **Networking Concepts**:
  - IP, DNS, Subnetting
  - HTTP, HTTPS
  - Firewalls, Ports, NAT
- **Command Line Interface (CLI)**: Comfort with basic commands and tools like `curl`, `ping`, etc.



AWS Prequiste's

✅ 2. Programming Basics (Preferred)

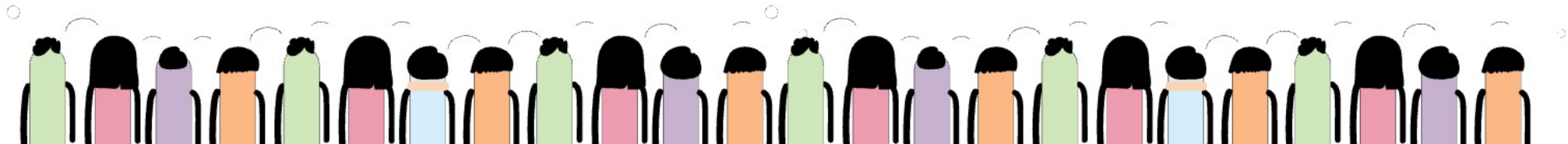
- **Python** or **Node.js** for writing automation, Lambda functions, SDK-based apps.
- Scripting: Bash or PowerShell basics.
- JSON & YAML: Configuration files in AWS often use these formats (e.g., CloudFormation, IAM policies).



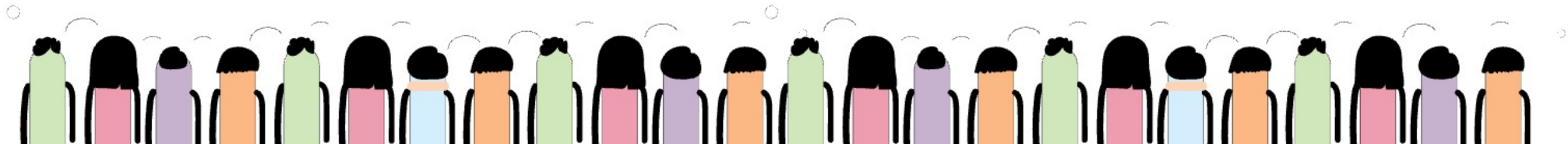
Python Prequiste's

✅ 1. Essential Python Concepts

Category	Must-Know Topics
✅ Basics	Variables, Data Types, Loops, If/Else
✅ Functions	`def`, arguments, `*args`, `**kwargs`, return
✅ File I/O	`open()`, read/write files (useful for logs, scripts)
✅ OOP	Classes, Inheritance, `__init__()`, Encapsulation
✅ Exceptions	`try-except`, custom exceptions (`raise`)
✅ Logging	`logging` module for security/event logs
✅ JSON/XML	Parsing for API responses (`json.loads()`)
✅ CLI	Using `argparse` for command-line automation



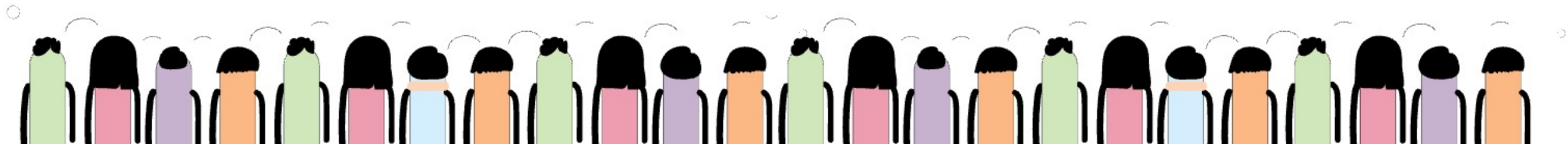
OOPS Coding



Python Prequiste's

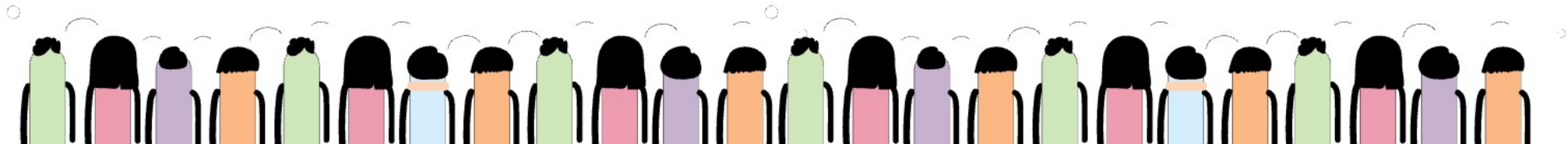
🧠 2. Core Python Packages for AWS Automation

Package	Usage
`boto3`	AWS SDK for Python: Interact with EC2, S3, IAM, RDS, Lambda, etc.
`botocore`	Low-level core of `boto3` (auto-installed)
`requests`	Interact with APIs (external/internal microservices)
`json`	Handle API payloads, AWS Lambda responses
`os` , `sys`	File paths, environment variables, process controls
`argparse`	For building reusable Python CLI tools
`logging`	Writing logs (best practice in AWS Lambda/CloudWatch)
`uuid` , `hashlib`	Secure file names, token generation for S3 uploads/downloads
`concurrent.futures`	ThreadPoolExecutor for parallel Lambda tasks or bulk uploads



Package Coding

coder [range]

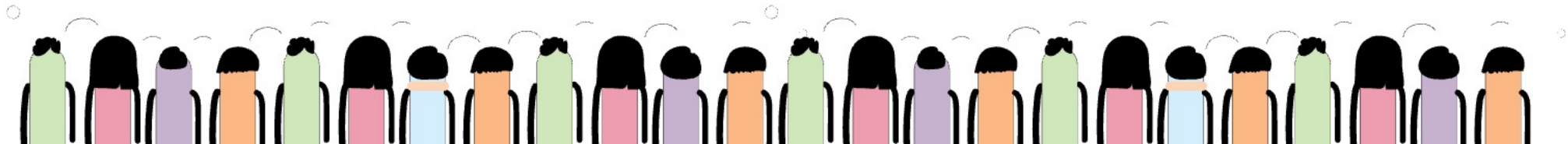


DSA Prequiste's

coder [range]

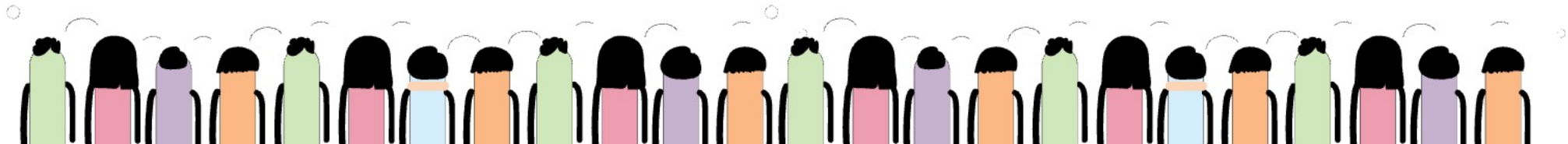
🧱 3. DSA (Data Structures & Algorithms) Relevance

DSA Area	AWS Use Case (Why it's helpful)
✅ Hashing	File integrity check before/after S3 upload (e.g., SHA-256)
✅ Queues	SNS/SQS event-driven systems
✅ Dictionaries	JSON parsing for IAM policies, Lambda configs
✅ Lists/Sorting	Manage bulk EC2 instances or sort log entries
✅ Trees/Graphs	Optional – useful in building custom resource dependency maps



DSA Coding

coder [range]



AWS Basic's

📦 1. Amazon EC2 (Elastic Compute Cloud)

Layman Definition:

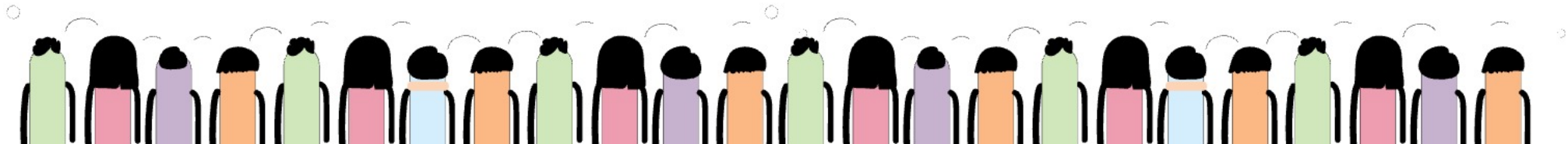
> It’s like renting a virtual computer from Amazon that you can use anytime, anywhere.

Image Concept:

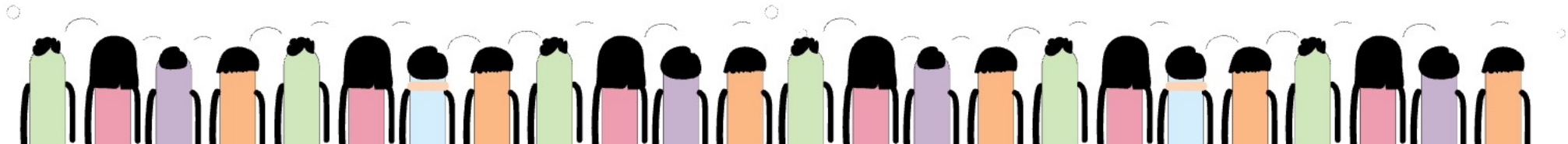
🖥️ A computer inside a cloud with a “power on” button.

Use case:

Hosting a website, running code, or using software just like your home computer—but online.



S3 Coding



S3 Basic's

🗄️ 2. Amazon S3 (Simple Storage Service)

Layman Definition:

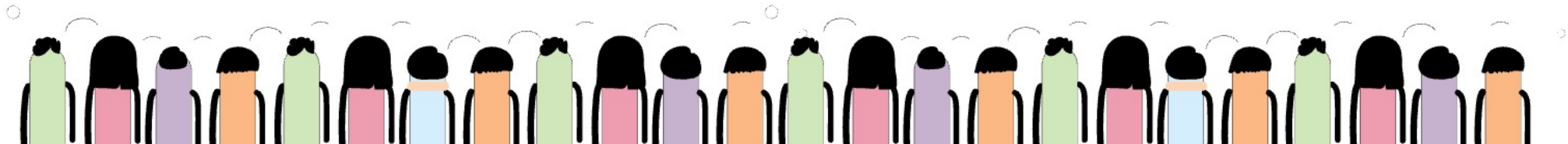
> It’s like a super-secure online **hard drive** to store files like photos, videos, or backups.

Image Concept:

🗂️ Cloud storage box labeled “Photos”, “Videos”, “Docs”.

Use case:

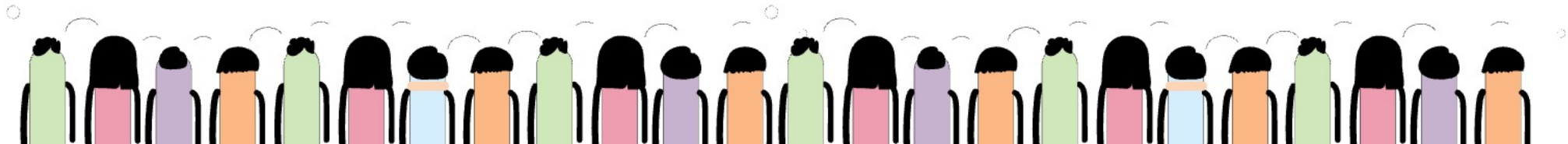
Storing backup files, videos, or serving images for a website.



S3 Internals

🧠 Key Internals Simulated

Feature	Local FS	Simulated S3
Structure	Hierarchical dirs/files	Flat key-value object store
File Write	Direct disk write	Object ID file + metadata in JSON
Metadata	Filesystem metadata (inode)	Manual metadata (JSON DB)
Redundancy	Handled by disk RAID	Simulated parity (CRC32)
Versioning	Not supported	UUID-based version ID
Read Logic	Direct file open	Lookup metadata + verify parity



RDS

🛢️ 3. Amazon RDS (Relational Database Service)

Layman Definition:

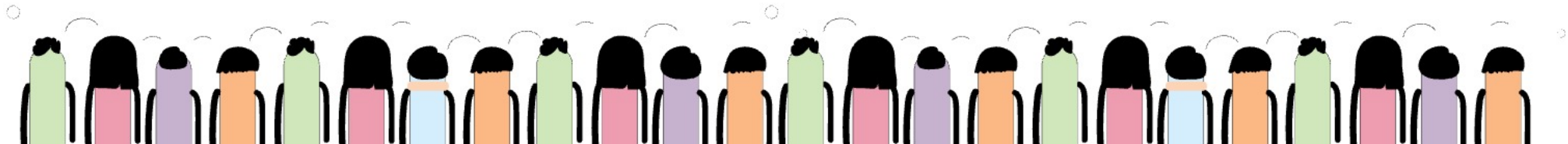
> It's like Amazon runs and manages a **ready-to-use database** for your apps.

Image Concept:

📚 A digital library (database) with a gear icon (auto-managed).

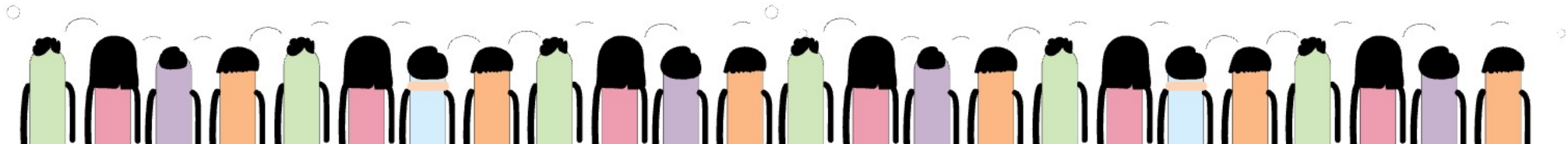
Use case:

Storing customer info, orders, or financial data for an app.



Lambda Coding

coder [range]



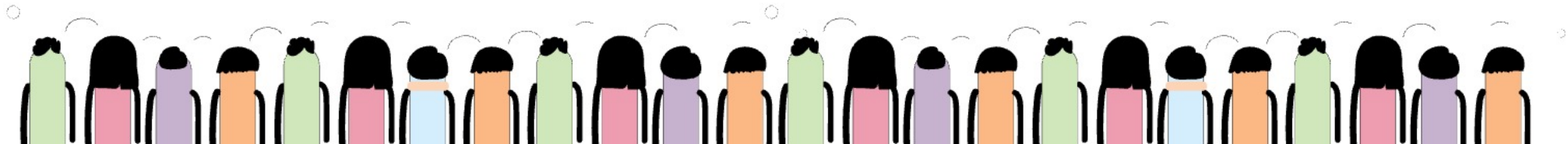
DynamoDB

🧠 Why AWS Built It:

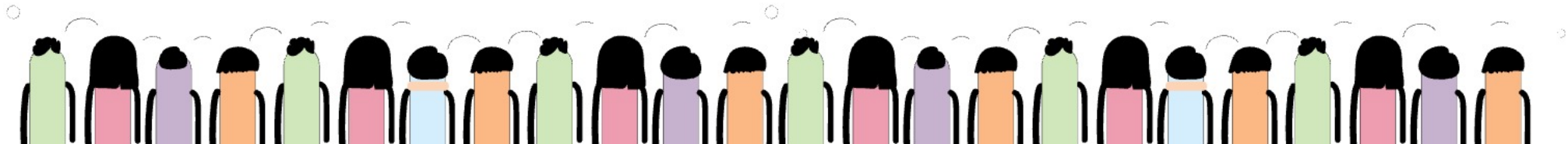
Needed a **highly available, eventually consistent** store. Traditional RDBMS couldn't scale at Amazon retail level.

🧱 Internals:

Layer	Details
Data Structure	**LSM Trees** for fast write throughput
Algorithms	- **Quorum-Based Writes** (N, R, W model) - **Gossip Protocols** for node health - **Vector Clocks** to detect write conflicts
File System	Custom SSTable format (like LevelDB) over NVMe
OS-Level	Thread pool tuning, I/O scheduler using deadline/cfq



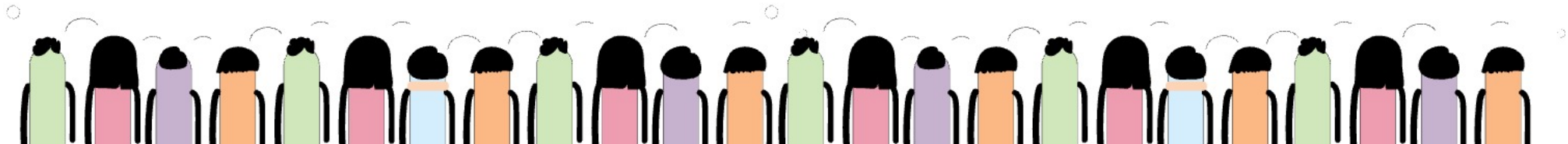
DynamoDB Coding



DynamoDB

✅ What is DynamoDB?

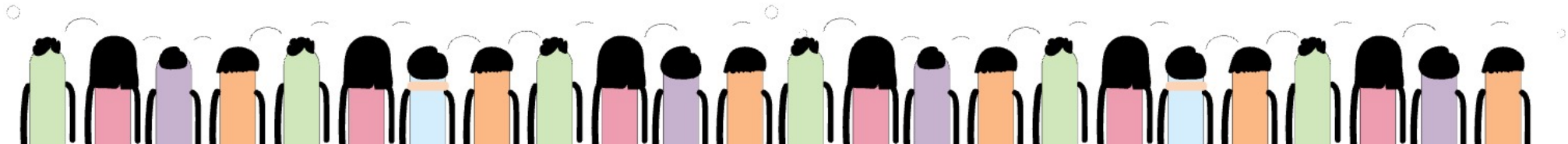
Feature	Description
💳 Type	NoSQL (key-value & document)
🧠 Core Idea	Distributed hash table (DHT) + Quorum consistency
🧩 Data Structure	Partition key (required) + Sort key (optional)
📡 Writes	Eventually consistent or strongly consistent
🔁 Replication	Multi-AZ, quorum-based



DynamoDB

🧠 What It Simulates

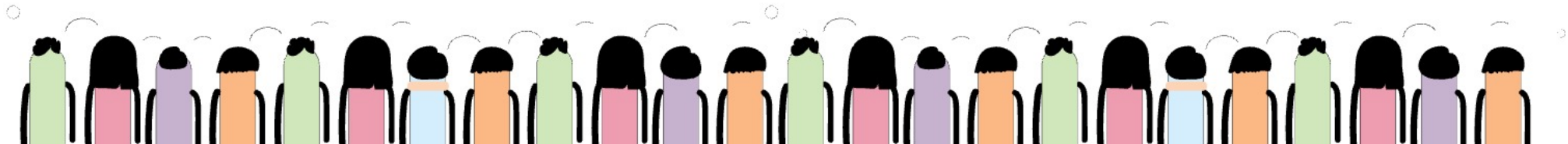
DynamoDB Concept	Python Code Equivalent
Partition Key / Sort Key	2-level dict (`replica[pk][sk]`)
Replication	Writes to 3 in-memory "nodes"
Quorum Read	Majority vote among 3 replicas
Eventual Consistency	No locks across all replicas
Threaded Writes	Simulates async replication latency



DynamoDB

🧠 DynamoDB Real Internals (Compared)

Feature	Real AWS DynamoDB	Simulated Version
💾 Storage	SSD-backed partitions	In-memory dictionary
⚙️ Partitioning	Hash(key) → partition	Dict key-based
🔁 Replication	Quorum + Paxos/RAFT	Threaded replica writes
🕐 TTL	Automatic purge	Not included (can add)
🔐 IAM Control	Per-table policies	Not included
🌍 Global Tables	Multi-region sync	Not included (can simulate)



IAM

👤 4. IAM (Identity and Access Management)

Layman Definition:

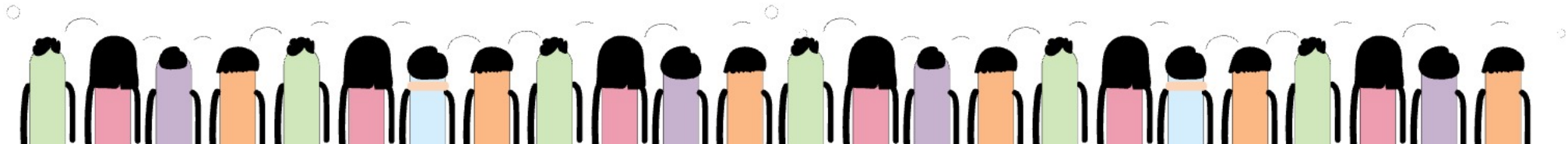
> It's like a **security guard** for your AWS account—decides **who can do what**.

Image Concept:

🛡️ A badge or lock with different user icons.

Use case:

Allowing only certain users to access certain AWS services.



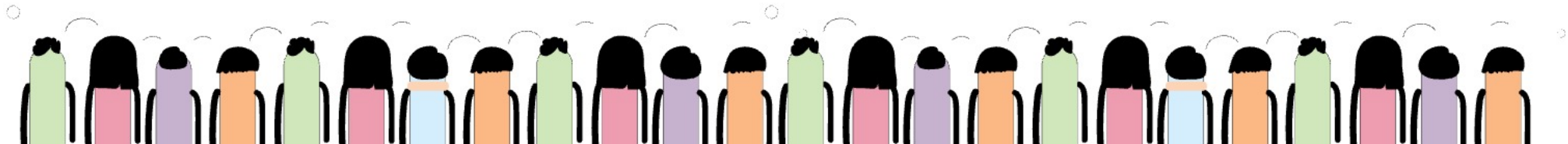
IAM

🧠 Why AWS Built It:

Existing ACL/RBAC systems weren't enough for **fine-grained, multi-service, federated access**.

🧠 Internals:

Layer	Details
Data Model	Directed Graph of Roles → Policies → Actions
Algorithms	- **Graph Evaluation Engine** - **Policy Merge Trees** (identity + resource) - Conflict resolution via Deny > Allow precedence
File System	Stores JSON policy trees in in-memory caches + encrypted blob stores

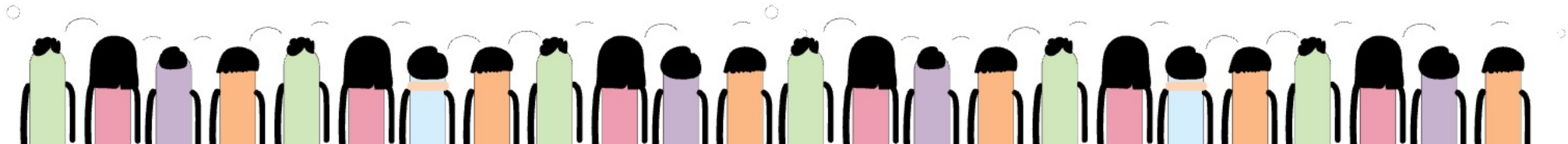


IAM Coding

✅ IAM Simulator in Python (Code Below)

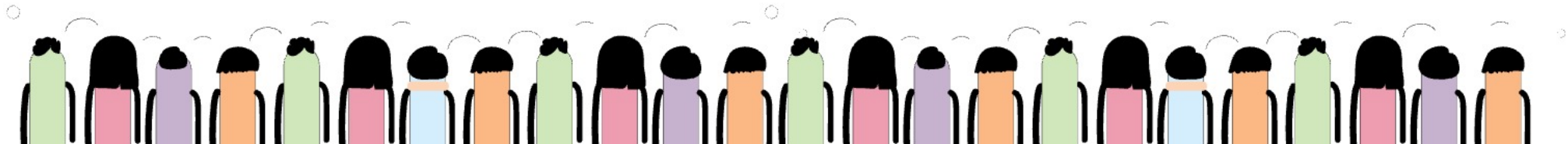
We'll simulate:

- Users
- Roles
- Resources (like `s3:GetObject`)
- Policies
- Permission Evaluation Logic



IAM Coding

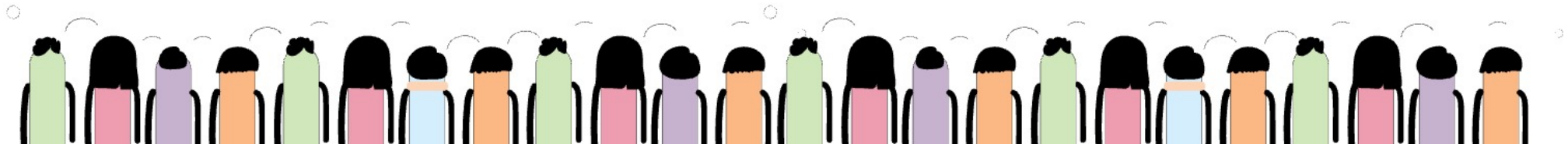
coder [range]



IAM Coding

🧠 Core Concepts Simulated

AWS IAM Concept	Python Simulation
User / Role	Python class
Policy	JSON-like dictionary
Action	String (e.g., `"s3:GetObject"`)
Resource	String (e.g., `"arn:aws:s3:::my-bucket/*"`)
Allow/Deny logic	Explicit logic tree
Evaluation Engine	`evaluate(user, action, resource)` function



Lambda

⚡ 5. AWS Lambda

Layman Definition:

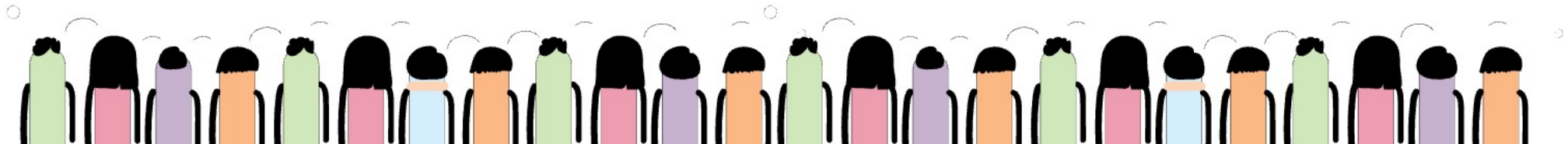
> It's like a **robot that runs your code only when needed**, and then goes to sleep.

Image Concept:

🤖 A robot waking up when a bell rings, runs code, then sleeps again.

Use case:

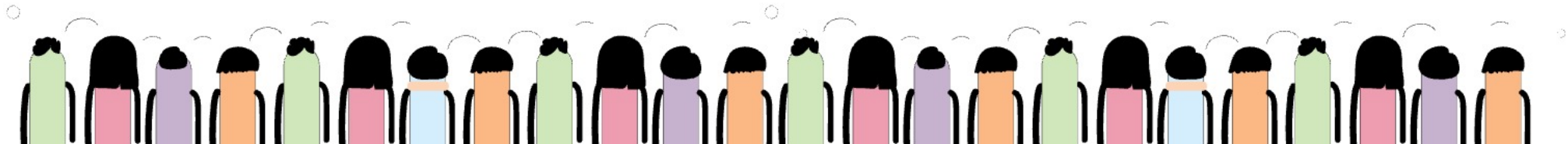
Send an email automatically when a form is submitted.



Core Service's

✅ 5. Core AWS Services You Must Know First

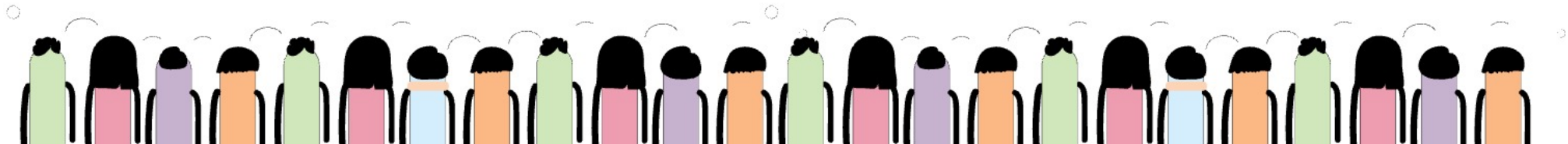
- **Compute**: EC2, Lambda
- **Storage**: S3, EBS
- **Databases**: RDS, DynamoDB
- **Networking**: VPC, Subnets, Security Groups, Route Tables
- **IAM**: Users, Roles, Policies
- **Monitoring**: CloudWatch
- **Infrastructure as Code**: CloudFormation, CDK or Terraform



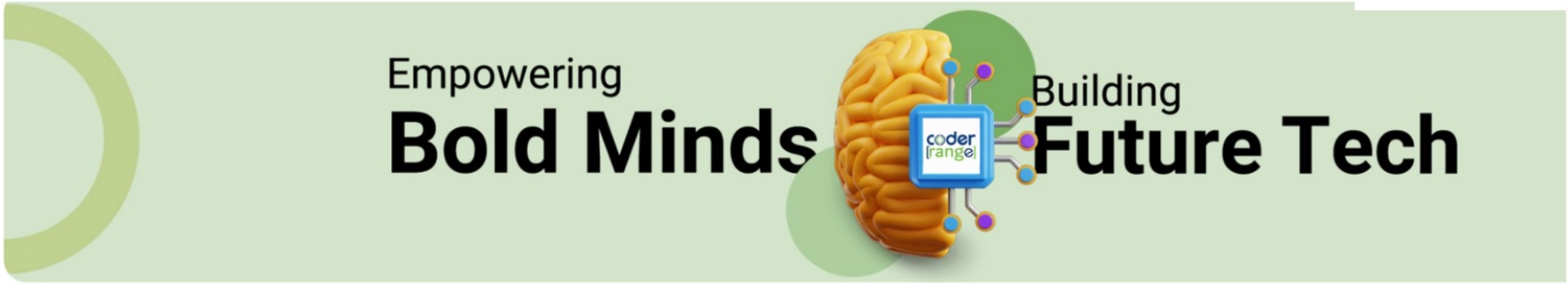
CoderRange

🛡️ That’s where YOU come in. And that’s where Coderrange stands beside you.

> 🏁 At **Coderrange**, we don’t just teach AWS—we **build warriors** of the cloud.
> 🔐 You’ll master Identity & Access Management. You’ll lock down S3 buckets. You’ll build Lambda automations that protect and serve.
>
> 💼 Whether you're applying for your **first job** or switching to **cybersecurity** or **DevOps**, you’ll no longer be lost in theory.
> You’ll **launch real code**. You’ll **face real scenarios**. You’ll **solve real-world problems.**



Utube Live Meetup

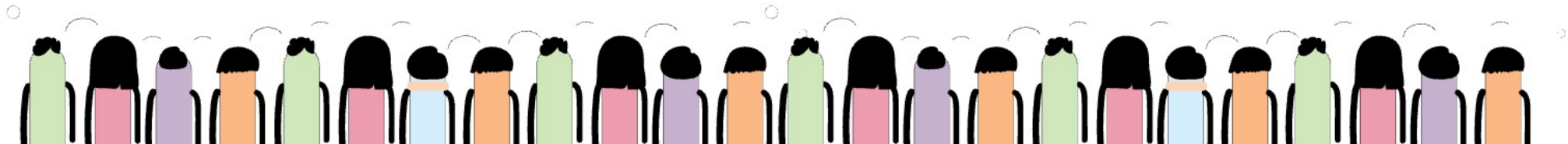


CoderRange

@CoderRange · 105 subscribers · 19 videos

Built in India. Made for the World. 🌍 ...more

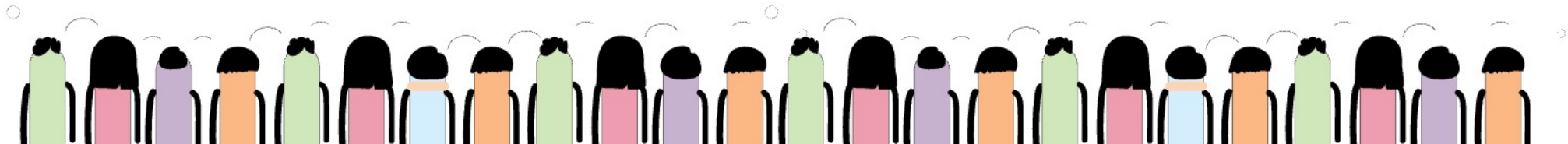
Subscribed



Python Exception's

🛑 4. Security & Exception Handling Best Practices

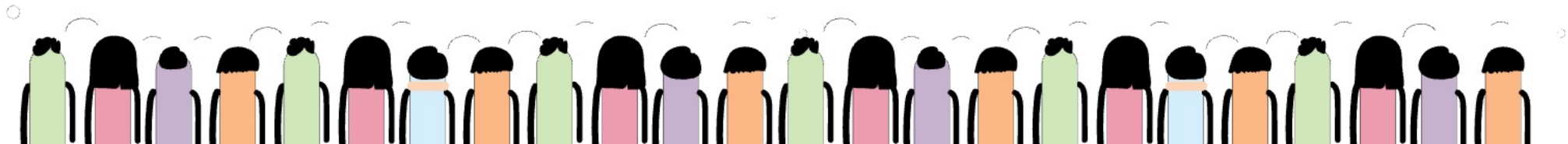
Practice	Example
✅ `try-except` around AWS SDK	Catch `ClientError`, `NoCredentialsError` in `boto3`
✅ Custom exceptions	Raise exceptions for invalid parameters or file formats
✅ Retry logic	Use exponential backoff (`botocore.retryhandler`)
✅ Logging + Alerting	Always log to CloudWatch or local log files
✅ Environment variables	NEVER hardcode secrets; use `.env` or AWS Secrets Manager



Python Real Time Sceanrio

📦 AWS-Specific Use Cases You Should Know in Python

Use Case	What You Should Know
✅ Upload to S3	`s3.upload_file()`, ACLs, exceptions
✅ EC2 Automation	Launch/stop instances via `boto3`
✅ IAM Policy Management	Attach/detach policies, create roles
✅ Lambda Deployment	Upload zip, create function, triggers
✅ CloudWatch Logs	Fetch logs, create alarms
✅ SNS/SES Notifications	Email/alerts from Lambda
✅ RDS Data Operations	Use `pymysql` or `psycopg2` + boto3



Next Smiling Meetup

coder [range]