🧩 Mini Project: Understanding the 5 Essential Shell Scripting Skills for Cloud Computing

My Understanding of the Requirement

This mini project is about connecting **core shell scripting skills** with **practical cloud automation tasks** — particularly using AWS services like EC2 and S3. The goal is to understand how these five scripting concepts (functions, arrays, environment variables, command-line arguments, and error handling) can be applied to automate common cloud operations efficiently and securely.

1️⃣ Functions

Understanding: Functions in shell scripting are reusable blocks of code that perform specific tasks. In this context, they help organize and modularize operations like:

- Creating an EC2 instance
- Configuring or creating an S3 bucket
- Verifying deployment or resource creation status

Example use case:

```
create_ec2() {
    aws ec2 run-instances --image-id $AMI_ID --instance-type $INSTANCE_TYPE
}
```

This helps make the script cleaner and easier to maintain or reuse.

2️⃣ Arrays

Understanding: Arrays in shell scripting allow us to store and manage multiple related values — like a list of AWS resources or instance IDs.

Use case in the project:

- Track all EC2 instances created by the script.
- Keep a list of S3 bucket names created.
- Easily loop through all created resources for verification or cleanup.

Example:

```
EC2_INSTANCES=("i-123456" "i-654321" "i-987654")
for instance in "${EC2_INSTANCES[@]}"; do
    echo "Checking status of $instance"
done
```

3️⃣ Environment Variables

Understanding: Environment variables help store reusable and sensitive information such as:

- AWS credentials (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`)
- Region settings (`AWS_DEFAULT_REGION`)
- Configuration parameters

They make the script **secure**, **portable**, and **environment-independent**.

Use case: Instead of hardcoding sensitive values in the script, they can be exported once and used throughout:

```
export AWS_DEFAULT_REGION=us-east-1
```

4️⃣ Command Line Arguments

Understanding: Command-line arguments make scripts **dynamic** and **flexible** by allowing users to pass parameters when running the script.

Use case in project:

- Specify EC2 instance type (`t2.micro`, `t3.medium`, etc.)
- Specify S3 bucket name or region on the fly

Example:

```
./deploy.sh t2.micro my-bucket
```

Inside the script:

```
INSTANCE_TYPE=$1
BUCKET_NAME=$2
```

5️⃣ Error Handling

Understanding: Error handling ensures the script can **detect**, **report**, and **recover** from failures instead of crashing.

Use case in project:

- Catch AWS CLI errors (e.g., failed instance creation)
- Log meaningful error messages
- Optionally retry failed operations

Example:

```
if ! aws ec2 run-instances --image-id $AMI_ID --instance-type $INSTANCE_TYPE; then
    echo "Error: Failed to create EC2 instance" >&2
    exit 1
fi
```

🧠 Overall Understanding

The goal of this mini project is **not to write the full script yet**, but to understand how these five concepts will be combined to:

- Automate cloud infrastructure setup (EC2, S3, etc.)
- Make scripts modular, flexible, secure, and fault-tolerant
- Build a foundation for DevOps and cloud automation workflows

---

🎯 Summary Table

Concept	Purpose	Example Cloud Use
Functions	Reuse and organize code	Create EC2, configure S3
Arrays	Store multiple resources	Track instance IDs or bucket names
Environment Variables	Store sensitive/config data	AWS credentials, regions
Command Line Arguments	Make script dynamic	Specify instance type or bucket
Error Handling	Make script reliable	Handle AWS CLI errors gracefully