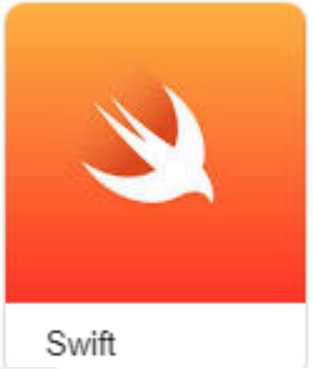

# DECLARATIVE AND IMPERATIVE PROGRAMMING LANGUAGES

Compare declarative and imperative programming languages







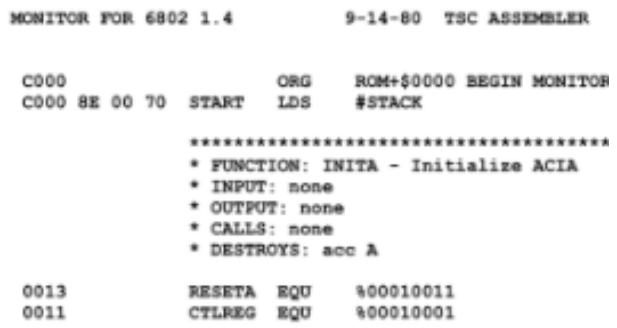


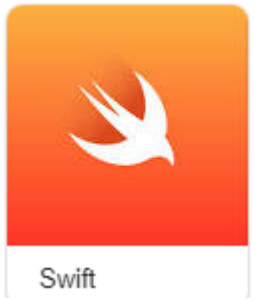


# What do this images show?







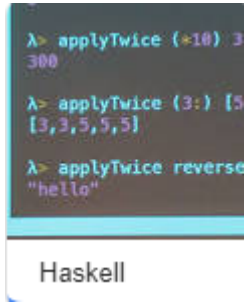








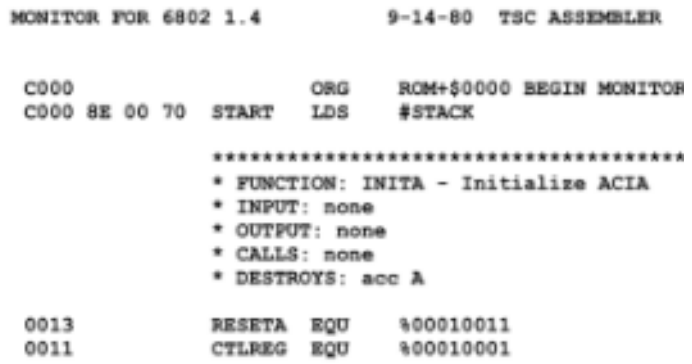

# How can we classify programming languages?









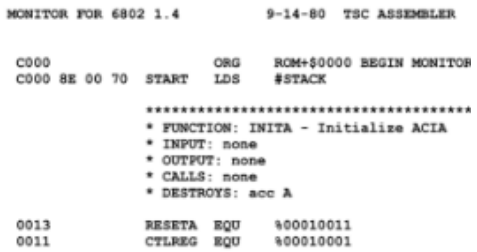
# Low level language


```
MONITOR FOR 6802 1.4           9-14-80   TSC ASSEMBLER


C000                    ORG     ROM+$0000 BEGIN MONITOR
C000 8E 00 70   START   LDS     #STACK

                ****************************************
                * FUNCTION: INITA - Initialize ACIA
                * INPUT: none
                * OUTPUT: none
                * CALLS: none
                * DESTROYS: acc A

0013            RESETA  EQU     %00010011
0011            CTLREG  EQU     %00010001
```

# High level language

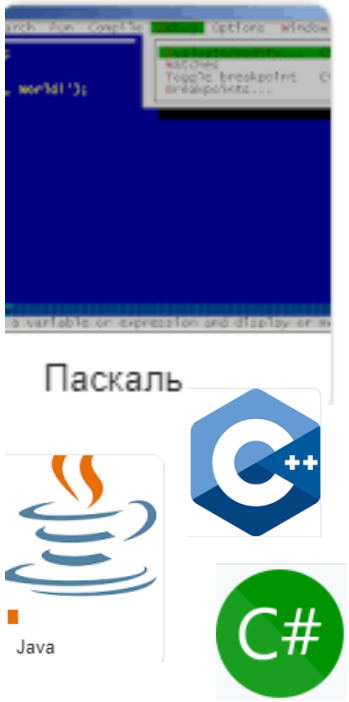
## 1GL


## 2GL




## 3GL

Procedural-Oriented languages

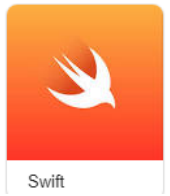



Cobol
Fortran

## 4GL

Problem-Oriented languages






SQL

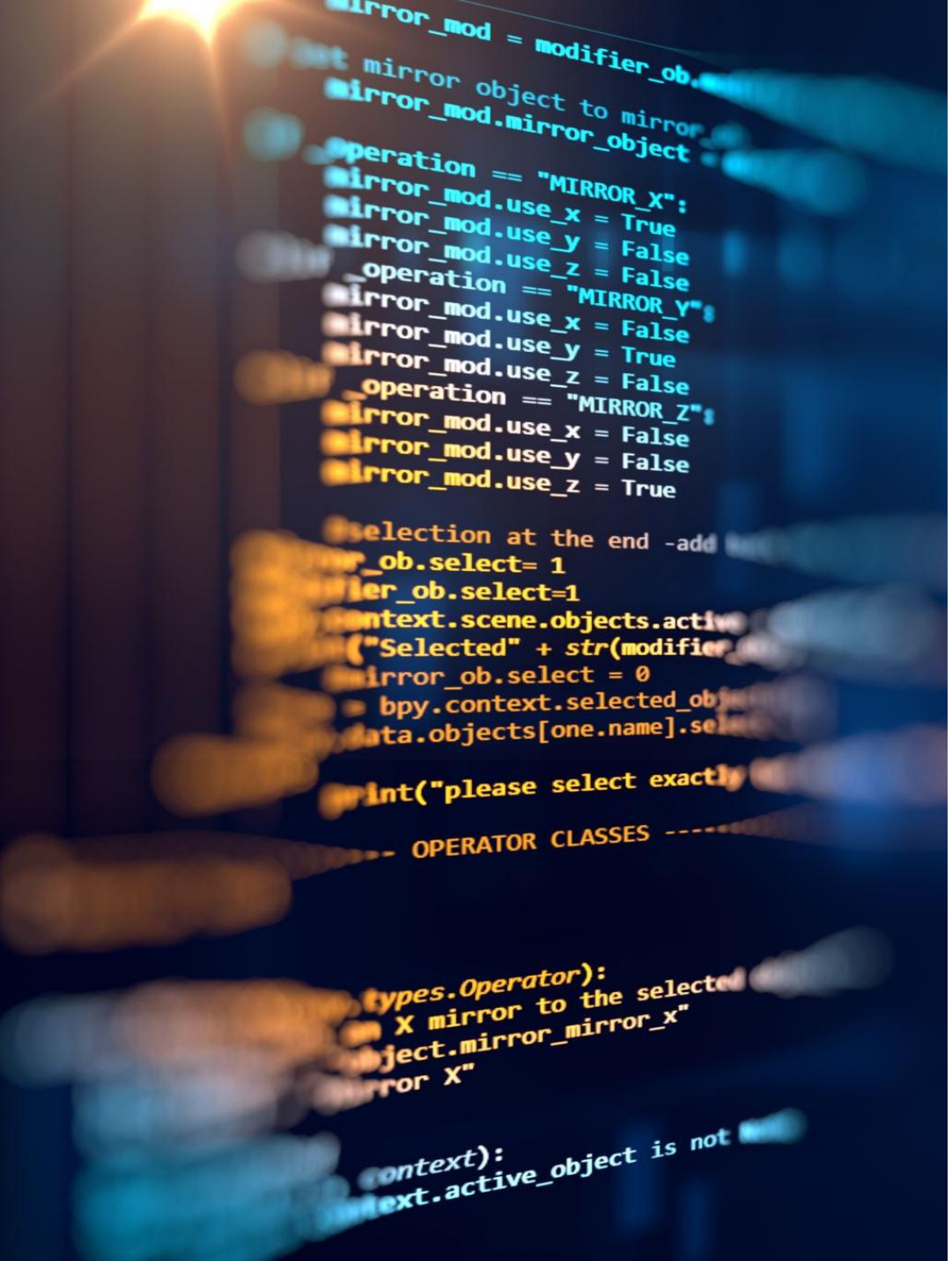
## 5GL

Natural language

# Programming Pradigms

Different fundamental approaches or styles to structuring and writing computer programs, guiding how programmers think about and solve problems.

- Imperative focuses on step-by-step instructions to change program state.
- Declarative describes what to achieve rather than how.




# What is Imperative Programming?

A programming paradigm that uses statements to change a program's state. It focuses on "**how**" to perform tasks.

Characteristics:

- Code describes the step-by-step instructions.

- Involves control structures like loops, conditionals, and assignments.

- The program state is managed explicitly.

**Examples:** C, C++, Java, Python.

Analogy: Like giving someone a recipe with detailed steps to follow.

# What is Declarative Programming?

A programming paradigm that expresses the **logic** of computation without describing its control flow. It focuses on **"what"** to achieve.

Characteristics:

- Code describes **desired outcomes** rather than detailed steps.
- Often relies on expressions and declarations.
- Handles the execution automatically, without needing direct control of the steps

**Examples**: SQL, Prolog.

Analogy: Like telling someone to "bake a cake" without specifying the exact steps.

# Key differences

| Feature | Imperative Programming | Declarative Programming |
|---|---|---|
| **Focus** | How to achieve a task | What the task is |
| **Code Style** | Step-by-step instructions | Describes desired result/outcome |
| **Control Flow** | Explicitly managed by the programmer | Implicitly handled by the language |
| **Examples** | C, Java, Python | SQL, HTML, Prolog |
| **State Management** | Explicit state manipulation | State management abstracted away |
| **Readability** | Can be less readable for complex logic | Generally more readable and concise |

# Imperative

```
# Calculate the sum of numbers from 1 to 5

total = 0
for i in range(1, 6):
    total += i
print(total)

# Output: 15
```

Uses loops and explicit state changes (the total variable)

```
n = 5
factorial = 1

for i in range(1, n + 1):
    factorial *= i

print(f"The factorial of {n} is {factorial}")
```

# Imperative

**Benefits:**

- Direct control over each step of execution.
- Programmers can optimize for specific hardware or constraints.
- Can handle a wide range of problems and domains

**Drawbacks:**

- Can become complex and hard to maintain, especially for large programs.
- Managing state and control flow explicitly can lead to errors.
- Focuses more on how things are done rather than what is achieved.

# Declarative

```
SELECT SUM(value)
FROM numbers
WHERE value BETWEEN
1 AND 5;
```

```
SELECT *
FROM users
WHERE age > 30;
```

Specifies **what** result is desired (sum of values), not **how** to compute it.

# Declarative

**Benefits:**

- Code is often easier to read and understand.
- Often fewer lines of code are required for the same task.
- Easier to write programs that take advantage of multiple cores.

**Drawbacks:**

- Abstracts away control flow, which may not allow for fine-grained optimization.
- Best suited for specific types of tasks (e.g., data querying, configuration).
- May not be as performant in low-level, resource-constrained environments like embedded systems

# Individual work

COMPARE IMPERATIVE AND DECLARATIVE PROGRAMMING PARADIGM

| | DECLARATIVE | IMPERATIVE |
|---|---|---|
| Definition | | |
| Advantages | | |
| Drawbacks | | |
| Example | | |