CSE216 Programming Abstractions

State University of New York, Korea

Agenda

 Programming paradigms: imperative, procedural, reactive, declarative, object-oriented, functional

Demo

Programming Paradigms

Programming Paradigms

 A programming paradigm is a style or approach to programming that provides a framework for building a software system. It is a set of principles, concepts, and practices that define the way of thinking and organizing the code to solve a specific problem. There are several programming paradigms, and each has its unique way of approaching problem-solving, code organization, and design. Some of the popular programming paradigms include:

- Imperative Programming: This paradigm focuses on the sequence of statements that modify the state of the program, and how to control that sequence using conditional statements, loops, and other control flow constructs.
- Object-Oriented Programming (OOP): This paradigm is based on the idea of organizing software systems as a collection of objects that interact with each other to solve a problem.
- Functional Programming: This paradigm emphasizes the use of functions to solve problems and avoid changing the state of the program.
- **Declarative Programming:** This paradigm focuses on **describing the problem** to be solved, rather than how to solve it.

Choosing a programming paradigm for your work

 The choice of a programming paradigm depends on the problem domain, the team's expertise, and the software requirements. Each paradigm has its strengths and weaknesses, and the choice of the paradigm can have a significant impact on the design, maintainability, and performance of the software system.

Imperative programming

- Imperative programming is a programming paradigm that emphasizes the sequence of statements that **modify the state of the program.** In imperative programming, the program consists of a set of commands or statements that change the program's state.
- Imperative programming is based on the Von Neumann architecture, which describes a computer
 as a machine that stores data and instructions in memory, fetches instructions from memory, and
 executes them in a sequential manner.
- In imperative programming, the programmer specifies how the program should perform its tasks, by **giving a sequence of commands** to be executed by the computer. These commands can include assignments, loops, conditionals, and function calls.
- Examples of imperative programming languages include **C**, **Java**, **Python**, and many others. These languages offer a rich set of control flow constructs and operators that enable the programmer to manipulate the program's state in a variety of ways.
- One of the advantages of imperative programming is that it provides the programmer with a great deal of control over the program's behavior. However, it can be challenging to write, debug, and maintain large imperative programs, as they can quickly become complex and difficult to understand.

Imperative Programming

- A **statement** is a syntactic unit of an imperative programming language that expresses some action to be carried out.
- The program in such a language thus becomes a sequence of statements.

Simple Statements

Imperative Programming

- An assignment statement performs an operation on information located in memory and stores the results in memory for later use.
 - Higher-level imperative languages permit evaluation of complex expressions that may consist of a combination of arithmetic operations and function evaluations.

Imperative Programming

- A <u>conditional statement</u> allows a sequence of statements (known as a *block* or a *code block*) to be executed only if some condition is met.
- Otherwise, the statements are skipped, and the execution sequence continues from the statement following them.

```
if (happy) {
    smile();
} else if (sad) {
    frown();
} else {
    stoic();
}
```

Compound Statements

Imperative Programming

- Looping statements allow a block to be executed multiple times.
- Loops can execute a block a predefined number of times, or they can execute them repeatedly until some condition changes.

```
while ((c = getchar()) != EOF) {
    putchar(c);
}

do {
    computation(i);
    ++i;
} while (i < 10);

for (i = 1; i < n; i *= 2) {
    printf("%d\n", i);
}</pre>
```

Procedural Programming

- Procedural programming is a programming paradigm that is based on the idea of breaking down a program into smaller, reusable procedures. In procedural programming, the program consists of a collection of procedures that operate on data, and the procedures are called in a specified sequence to perform a particular task.
- In procedural programming, the **focus is on the procedures** that are executed, rather than the data that is manipulated. The procedures are defined using a set of instructions that are executed in a sequence, with control structures such as loops and conditionals used to control the flow of the program.
- Procedural programming is particularly useful for developing programs that perform a series of operations on data, as the data can be passed between functions to perform various operations. It is widely used for developing programs that involve input/output operations, data processing, and mathematical calculations.
- One of the advantages of procedural programming is that it is relatively easy to
 understand and debug, as each function is responsible for a specific task. However, it can
 be challenging to write and maintain large procedural programs, as the functions can
 become complex and difficult to manage.

An example in C

```
#include <stdio.h>
// Define a function that prints the Fibonacci sequence up to n
void fibonacci(int n) {
   int a = 0, b = 1, c, i;
   printf("%d %d ", a, b);
   for(i = 2; i < n; i++) {
       c = a + b;
       printf("%d ", c);
       a = b;
       b = c;
// Call the function to print the Fibonacci sequence up to 10
fibonacci(10);
```

An example in Python

```
# Function to calculate the area of a rectangle
def calculate_area(length, width):
    area = length * width
    return area
# Function to calculate the perimeter of a rectangle
def calculate_perimeter(length, width):
    perimeter = 2 * (length + width)
    return perimeter
# Main program
length = float(input("Enter the length of the rectangle: "))
width = float(input("Enter the width of the rectangle: "))
area = calculate_area(length, width)
perimeter = calculate_perimeter(length, width)
print("The area of the rectangle is:", area)
print("The perimeter of the rectangle is:", perimeter)
```

Object-oriented Programming

- Object-oriented programming (OOP) is a programming paradigm or a style of programming that is based on the concept of "objects." An object is a self-contained unit that consists of both data and the methods that operate on that data. In OOP, everything is treated as an object, and the code is organized around these objects.
- OOP is widely used in software development because it provides an advantageous way of creating complex programs, making it easier to write, test, and maintain code. Some of the most popular programming languages that use OOP include Java, C++, Python, and Ruby.

Core concepts in OOP

• Encapsulation: It means that the data and behavior of an object are hidden from the outside world, and only the methods that the object exposes can be used to interact with it.

• **Inheritance**: It allows for the creation of new classes by inheriting properties and methods **from existing ones**.

 Polymorphism: It means that objects can take on different forms and exhibit different behaviors, depending on the context in which they are used.

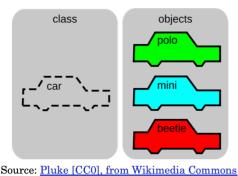
OOP concept 1:

Encapsulation

```
public class BankAccount {
   private int balance;
   public BankAccount(int balance) {
       this.balance = balance;
   }
   public int getBalance() {
        return balance;
   public void deposit(int amount) {
        balance += amount;
   }
   public void withdraw(int amount) {
       if (balance >= amount) {
            balance -= amount;
       } else {
            System.out.println("Insufficient funds");
```

OOP concept 2: Inheritance

- Inheritance is a fundamental concept in Object-Oriented Programming (OOP) that allows a new class to be based on an existing class. In other words, the new class (called a subclass or derived class) inherits the properties and methods of the existing class (called the superclass or base class).
- Inheritance allows for code reuse and makes it easier to create and maintain complex programs. The subclass can add new properties and methods or modify the existing ones of the superclass, while still inheriting the characteristics of the superclass. This allows for more efficient and modular code development.



OOP concept 2: Inheritance

```
1 * class Vehicle {
        protected String make;
        protected String model;
        protected int year;
        public Vehicle(String make, String model, int year) {
            this.make = make;
            this.model = model;
9
            this.year = year;
10
11 }
12
13 - class Car extends Vehicle {
14
        private String country;
15
16 -
        public KoreanCar(String make, String model, int year, String country) {
            super(make, model, year);
17
18
            this.country = country;
19
20
21 -
        public String getCountry() {
22
            return country;
23
24 }
25
26 - class Main {
27 -
        public static void main(String[] args) {
            Car car = new Car("Kia", "Seltos", 2022, "South Korea");
28
29
            System.out.println(car.make); // prints "Kia"
30
            System.out.println(car.model); // prints "Seltos"
31
            System.out.println(car.year); // prints 2022
32
            System.out.println(car.getCountry()); // prints "South Korea"
33
34 }
```

OOP concept 3: Polymorphism

```
// Base class
class Shape {
    public void draw() {
        System.out.println("Drawing a shape...");
}
// Derived class 1
class Circle extends Shape {
    public void draw() {
        System.out.println("Drawing a circle...");
}
// Derived class 2
class Square extends Shape {
    public void draw() {
        System.out.println("Drawing a square...");
// Main class
class Main {
    public static void main(String[] args) {
        // Create an array of shapes
        Shape[] shapes = { new Circle(), new Square() };
        // Draw all the shapes in the array
        for (Shape shape : shapes) {
            shape.draw();
```

 Thanks to polymorphism, we can treat each object in the array as if it were a Shape object, even though they are actually instances of different classes. This allows us to write code that is flexible and reusable — we can easily add new shapes to the array without having to change the for loop or any other code.

Declarative programming languages

- In declarative programming, the program describes the desired result, rather than specifying how to achieve it. This is in contrast to procedural programming, where the program specifies a series of steps to accomplish a task.
- One example of declarative programming is SQL (Structured Query Language), which is used to query relational databases. In SQL, a programmer specifies the criteria for selecting records from a database, and the database management system determines how to retrieve the data.
- Another example is HTML. Programmers specify web structure and content using tags, and the web browser determines how to render the page based on those tags.

Declarative programming in SQL

```
-- Define a table of students
CREATE TABLE students (
  id INT,
  name VARCHAR(255),
 major VARCHAR(255),
  gpa FLOAT
);
-- Select all students with a GPA greater than 3.0
SELECT id, name, major
FROM students
WHERE gpa > 3.0;
```

- Note that we're not specifying how to retrieve the data or iterating over it step-by-step, but rather describing the desired result in a declarative way. SQL takes care of the details of how to retrieve the data and return the desired result.
- So, this is declarative programming we declare what we want to happen, and the programming language takes care of the details.

Declarative programming in HTML

```
<!DOCTYPE html>
<html>
 <head>
   <title>My Web Page</title>
 </head>
 <body>
   <h1>Welcome to My Web Page</h1>
   This is a paragraph of text.
   <l
    Item 1
    Item 2
    Item 3
   </body>
</html>
```

 Note that we're not specifying how the page should be rendered, but rather describing the structure and content of the page in a declarative way. The web browser takes care of the details of how to render the page.

Functional Programming

1

Based on recursive definitions.

• They are inspired by a computational model called **lambda calculus**, developed by Alonzo Church in the 1930s.



A program is viewed as a mathematical function that transforms an input to an output. It is often defined in terms of simpler functions.

• We will see many examples of functional programming in multiple languages (e.g., Java, Python, OCaML).

Functional Programming in Python

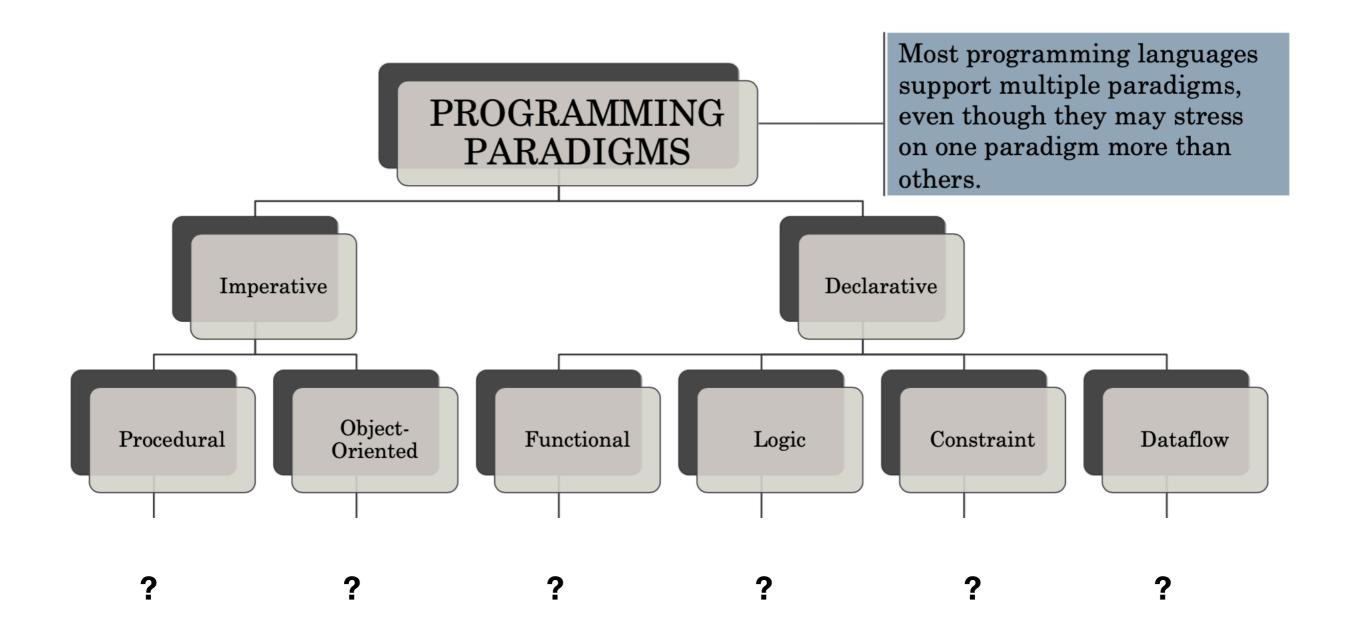
```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = filter(lambda x: x % 2 == 0, numbers)
squared_numbers = map(lambda x: x ** 2, even_numbers)
sum_of_squares = sum(squared_numbers)
```

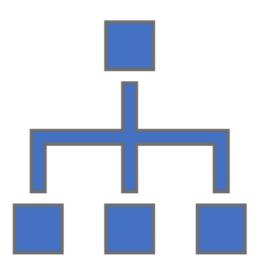
Functional Programming in Java

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
int sumOfSquares = numbers.stream()
    .filter(x -> x % 2 == 0)
    .map(x -> x * x)
    .reduce(0, Integer::sum);
```

Functional Programming in Ocaml

```
let numbers = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10];;
let is_even x = x mod 2 = 0;;
let sum_of_squares =
    List.filter is_even numbers
    |> List.map (fun x -> x * x)
    |> List.fold_left (+) 0;;
```





- One more example. Three paradigms.
- Implementing the greatest common divisor (GCD) solution in different paradigms and different languages.

The GCD problem: pseudocode

Paradigm 1: Imperative Programming in Python

Paradigm 2: Object-Oriented Programming in Java

```
/MyClass.java
   1 public class MyClass {
   2
   3
           static int gcd (int x, int y){
   6 =
               while (x!=y){
                   if (x>y) x = x-y;
                   else y = y - x;
  10
               return x;
  11
  12
  13
  14
  15
           public static void main(String args[]) {
  16 -
  17
             int x=10;
  18
             int y=25;
  19
  20
             System.out.println("GCD of x+y = "+ gcd(x,y));
  21
  22
  23 }
```

Paradigm 3: Functional Programming in Ocaml

```
1  let rec gcd (x, y) = if x> y then gcd (x-y, y) else if x < y then gcd (x, y -x) else x ;;
2  let r= gcd(15, 40);;
4  print_int r;;
6  </pre>
```

Summary

- Imperative, functional, object-oriented paradigms
- Instead of learning languages by languages, it is much more efficient to learn programming language paradigms.