Programming Paradigms & Practices

Mohamed Sweelam

Software Engineer

Outline

Course Objectives

1. Provide good Arabic content for the topic

Course Objectives

1. Provide good Arabic content for the topic
2. Provide comprehensive insights into programming paradigms

Course Objectives

1. Provide good Arabic content for the topic
2. Provide comprehensive insights into programming paradigms
3. Explore advanced programming techniques

Course Objectives

1. Provide good Arabic content for the topic
2. Provide comprehensive insights into programming paradigms
3. Explore advanced programming techniques
4. Discuss best practices for project structure and code quality

Course Objectives

1. Provide good Arabic content for the topic
2. Provide comprehensive insights into programming paradigms
3. Explore advanced programming techniques
4. Discuss best practices for project structure and code quality
5. Overview of deployment models and strategies

Understanding Programming Paradigms

Definition *wikipedia*

Programming paradigms are fundamental styles or approaches to computer programming, offering distinct methodologies for designing and structuring software.

Importance *ChatGPT*

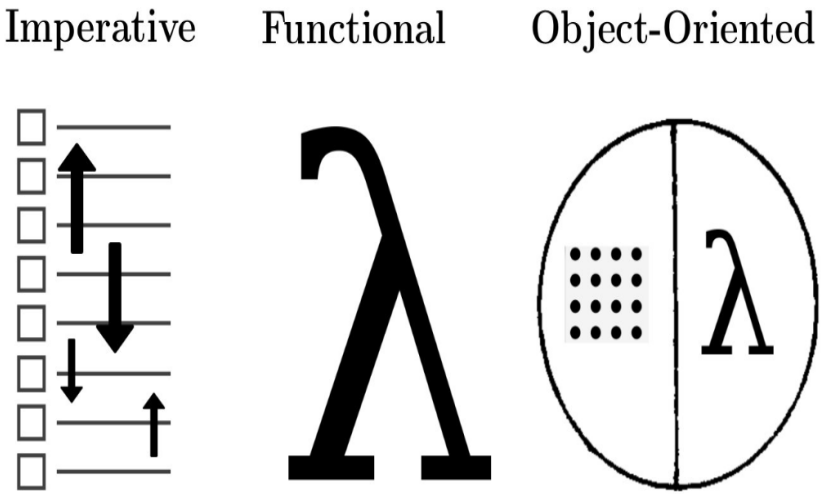
Understanding different programming paradigms is crucial for selecting the right approach to solve specific problems, leading to more efficient and maintainable code.

Historical Context **wikipedia**

Programming paradigms have evolved over time, with significant contributions from various programming languages that introduced unique features and concepts, shaping the way we write software today.

Programming Paradigms Types

- Imperative Programming
- Procedural Programming
- Object-Oriented Programming
- Declarative Programming
- Functional Programming
- Event-Driven Programming
- Aspect-Oriented Programming
- Reactive Programming



Imperative Programming

- In this paradigm, the program is a sequence of instructions that explicitly change the program state. It focuses on how to achieve a task.

Imperative Programming

- In this paradigm, the program is a sequence of instructions that explicitly change the program state. It focuses on how to achieve a task.
- C, Python, Java (most common imperative languages have procedural features as well)

Imperative Programming

- In this paradigm, the program is a sequence of instructions that explicitly change the program state. It focuses on how to achieve a task.
- C, Python, Java (most common imperative languages have procedural features as well)
- **Key Concepts:** Variables, controls (loops,conditionals), and subroutines.

Imperative Programming

- In this paradigm, the program is a sequence of instructions that explicitly change the program state. It focuses on how to achieve a task.
- C, Python, Java (most common imperative languages have procedural features as well)
- **Key Concepts:** Variables, controls (loops,conditionals), and subroutines.

```c
#include <stdio.h>

int main() {
  int Salaries[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int N = sizeof(Salaries) / sizeof(Salaries[0]);
  int result = 0, i;

  for (i = 0; i < N; i++) {
    if (Salaries[i] % 2 == 0) {
      result += Salaries[i];
    }
  }

  printf("Sum of even salaries is %d", result);
  return 0;
}
```

Procedural Programming

- A subset of imperative programming that structures programs as a series of procedures or functions. These procedures perform operations on data and are reusable.

Procedural Programming

- A subset of imperative programming that structures programs as a series of procedures or functions. These procedures perform operations on data and are reusable.
- C, Pascal, Python.

Procedural Programming

- A subset of imperative programming that structures programs as a series of procedures or functions. These procedures perform operations on data and are reusable.
- C, Pascal, Python.
- **Key Concepts:** Procedures, functions, modular programming.

Procedural Programming

- A subset of imperative programming that structures programs as a series of procedures or functions. These procedures perform operations on data and are reusable.
- C, Pascal, Python.
- **Key Concepts:** Procedures, functions, modular programming.

```c
#include <stdio.h>

int SumEven(int arr[], int N) {
    int sum = 0;
    int i;

    for (i = 0; i < N; i++) {
        if (arr[i] % 2 == 0) {
            sum += arr[i];
        }
    }

    return sum;
}

int main() {
    int arr[10] = {1,2,3,4,5,6,7,8,9,10};

    int result = SumEven(arr, 10);

    printf("sumi of even numbers is %d \n" , result);

    return 0;
}
```

Object-Oriented Programming

- Focuses on designing software using objects that represent real-world entities. Objects are instances of classes that encapsulate data and behavior.

Object-Oriented Programming

- Focuses on designing software using objects that represent real-world entities. Objects are instances of classes that encapsulate data and behavior.
- Java, C#, PHP, Ruby.

Object-Oriented Programming

- Focuses on designing software using objects that represent real-world entities. Objects are instances of classes that encapsulate data and behavior.
- Java, C#, PHP, Ruby.
- **Key Concepts:** Classes, objects, polymorphism, encapsulation, abstraction.

Object-Oriented Programming

- Focuses on designing software using objects that represent real-world entities. Objects are instances of classes that encapsulate data and behavior.
- Java, C#, PHP, Ruby.
- **Key Concepts:** Classes, objects, polymorphism, encapsulation, abstraction.
- *let's try an example*

Declarative Programming

- In this paradigm, the programmer specifies what the program should accomplish, rather than how to accomplish it. This is often used in conjunction with other paradigms like logical or functional programming.

Declarative Programming

- In this paradigm, the programmer specifies what the program should accomplish, rather than how to accomplish it. This is often used in conjunction with other paradigms like logical or functional programming.
- SQL, HTML, CSS.

Declarative Programming

- In this paradigm, the programmer specifies what the program should accomplish, rather than how to accomplish it. This is often used in conjunction with other paradigms like logical or functional programming.
- SQL, HTML, CSS.
- **Key Concepts:** Expressions, constraints, high-level abstraction.

Declarative Programming

- In this paradigm, the programmer specifies what the program should accomplish, rather than how to accomplish it. This is often used in conjunction with other paradigms like logical or functional programming.
- SQL, HTML, CSS.
- **Key Concepts:** Expressions, constraints, high-level abstraction.

```sql
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

Functional Programming

- Emphasizes computation using mathematical functions and avoids changing states or mutable data. Functions are first-class citizens and can be passed as arguments and returned as values.

Functional Programming

- Emphasizes computation using mathematical functions and avoids changing states or mutable data. Functions are first-class citizens and can be passed as arguments and returned as values.
- Haskell, Lisp, Scala, Erlang, F#.

Functional Programming

- Emphasizes computation using mathematical functions and avoids changing states or mutable data. Functions are first-class citizens and can be passed as arguments and returned as values.
- Haskell, Lisp, Scala, Erlang, F#.
- **Key Concepts:** Pure functions, immutability, higher-order functions, recursion.

Functional Programming

- Emphasizes computation using mathematical functions and avoids changing states or mutable data. Functions are first-class citizens and can be passed as arguments and returned as values.
- Haskell, Lisp, Scala, Erlang, F#.
- **Key Concepts:** Pure functions, immutability, higher-order functions, recursion.

```haskell
sumList :: [Int] -> Int
sumList = foldl (+) 0                          -- Higher Order Function

main = do
    let salaries = [1,2,3,4,5,6,7,8,9,10]
    print ("result = ", sumList salaries)     -- result = 55
```

Functional Programming

```haskell
sumList :: [Int] -> Int
sumList = foldl (+) 0                          -- Higher Order Function

main = do
    let salaries = [1,2,3,4,5,6,7,8,9,10]
    print ("result = ", sumList salaries)     -- result = 55
```

Functional Programming

```haskell
sumList :: [Int] -> Int
sumList = foldl (+) 0                          -- Higher Order Function

main = do
    let salaries = [1,2,3,4,5,6,7,8,9,10]
    print ("result = ", sumList salaries)     -- result = 55
```

```java
import java.util.Arrays;

class SalarySum {

  public static void main(String[] args) {
    int[] salaries = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    int result = Arrays.stream(salaries)
                       .reduce(0, (a, b) -> a + b);

    System.out.println("result = " + result);   // result = 55
  }
}
```

Typing Categories

- Typing refers to how data and variables are categorized based on their data types like integers, strings, or boolean.

source: Static vs Dynamic Typing: A Detailed Comparison

Typing Categories

- Typing refers to how data and variables are categorized based on their data types like integers, strings, or boolean.
- This categorization is crucial as it determines how a programming language handles and manipulates data

source: Static vs Dynamic Typing: A Detailed Comparison

Typing Categories

- Typing refers to how data and variables are categorized based on their data types like integers, strings, or boolean.
- This categorization is crucial as it determines how a programming language handles and manipulates data
- Typing involves specifying the data type of a variable or allowing the programming language to deduce it automatically.

source: Static vs Dynamic Typing: A Detailed Comparison

Typing Categories

- Typing refers to how data and variables are categorized based on their data types like integers, strings, or boolean.
- This categorization is crucial as it determines how a programming language handles and manipulates data
- Typing involves specifying the data type of a variable or allowing the programming language to deduce it automatically.

Static Typing

Static typing is a typing system where variables are bound to a data type during compilation. Once a variable is assigned a data type it remains unchanged throughout the programs execution.

Dynamic Typing

Dynamic typing allows variables to be bound to data types at runtime instead of during compilation. This flexibility enables concise code and ease of use.

source: Static vs Dynamic Typing: A Detailed Comparison

Static Typing

- One of the advantages of typing is ensuring type safety which reduces the chances of runtime errors caused by mismatches in data types.

source: Static vs Dynamic Typing: A Detailed Comparison

Static Typing

- One of the advantages of typing is ensuring type safety which reduces the chances of runtime errors caused by mismatches in data types.
- Error detection. Since the compiler knows the data types during the development process it can catch errors before runtime resulting in reliable software.

source: Static vs Dynamic Typing: A Detailed Comparison

Static Typing

- One of the advantages of typing is ensuring type safety which reduces the chances of runtime errors caused by mismatches in data types.
- Error detection. Since the compiler knows the data types during the development process it can catch errors before runtime resulting in reliable software.
- Performance benefits. The compiler can optimize code effectively in languages with typing potentially leading to faster execution.

source: Static vs Dynamic Typing: A Detailed Comparison

Static Typing

- One of the advantages of typing is ensuring type safety which reduces the chances of runtime errors caused by mismatches in data types.
- Error detection. Since the compiler knows the data types during the development process it can catch errors before runtime resulting in reliable software.
- Performance benefits. The compiler can optimize code effectively in languages with typing potentially leading to faster execution.
- PLs: C++, Java, and Rust

source: Static vs Dynamic Typing: A Detailed Comparison

Static Typing

- One of the advantages of typing is ensuring type safety which reduces the chances of runtime errors caused by mismatches in data types.
- Error detection. Since the compiler knows the data types during the development process it can catch errors before runtime resulting in reliable software.
- Performance benefits. The compiler can optimize code effectively in languages with typing potentially leading to faster execution.
- PLs: C++, Java, and Rust

```cpp
//      C++ example
bool IsPrime(int number) {
  bool is_prime = true;             // Type "bool" is detected in compile time

  if (number == 0 || number == 1) {
    return false;
  }

  int i;                            // Type "int" is detected in compile time
  for (i = 2; i <= number/2; ++i) {
    if (number % i == 0) {
      is_prime = false;
      break;
    }
  }

  return is_prime;
}
```

source: Static vs Dynamic Typing: A Detailed Comparison

Dynamic Typing

- Flexibility: Variables have the ability to change their data type during runtime.

source: Static vs Dynamic Typing: A Detailed Comparison

Dynamic Typing

- Flexibility: Variables have the ability to change their data type during runtime.
- Simplification: No need to explicitly specify data types when coding.

source: Static vs Dynamic Typing: A Detailed Comparison

Dynamic Typing

- Flexibility: Variables have the ability to change their data type during runtime.
- Simplification: No need to explicitly specify data types when coding.
- PLs: Python, Javascript, and Ruby.

source: Static vs Dynamic Typing: A Detailed Comparison

Dynamic Typing

- Flexibility: Variables have the ability to change their data type during runtime.
- Simplification: No need to explicitly specify data types when coding.
- PLs: Python, Javascript, and Ruby.

```python
# Program to add two matrices using nested loop
def sum_matrix():
    X = [[12,7,3],
        [4 ,5,6],
        [7 ,8,9]]

    Y = [[5,8,1],
        [6,7,3],
        [4,5,9]]

    result = [[0,0,0],
              [0,0,0],
              [0,0,0]]

    # iterate through rows
    for i in range(len(X)):
       # iterate through columns
       for j in range(len(X[0])):
          result[i][j] = X[i][j] + Y[i][j]

    for r in result:
       print(r)
```

source: Static vs Dynamic Typing: A Detailed Comparison

Static vs Dynamic Typing

Aspect	Static Typing	Dynamic Typing
Determination Time	Determined at compile-time.	Determined at runtime.
Error Detection	Errors caught during compilation.	Errors may appear during program execution.
Performance	Typically faster due to compile-time optimizations.	Possible overhead from runtime type-checks.
Coding Verbosity	Requires explicit type declarations.	Concise; types aren't specified explicitly.
Flexibility	Variables bound to one type.	Variable types can change during execution.
Type Safety	High type safety through early error detection.	Some type safety traded off for flexibility.

source: Static vs Dynamic Typing: A Detailed Comparison

Advanced Programming Techniques

- Multithreading and Concurrency

Advanced Programming Techniques

- Multithreading and Concurrency
- Reactive Programming and Asynchronous Streams

Advanced Programming Techniques

- Multithreading and Concurrency
- Reactive Programming and Asynchronous Streams
- Memory Management and Optimization

Advanced Programming Techniques

- Multithreading and Concurrency
- Reactive Programming and Asynchronous Streams
- Memory Management and Optimization
- Effective Error Handling and Debugging

Project Structure and Code Quality

- Organizing Your Codebase
- Implementing Best Practices for Readability and Maintainability
- Writing Clean and Testable Code
- Integrating Continuous Integration and Automated Testing

Deployment Models

- Understanding Different Deployment Strategies
- Containerization and Orchestration with Docker and Kubernetes
- Continuous Deployment and Delivery Pipelines
- Monitoring and Maintaining Production Environments

Conclusion

- Recap of Key Learnings
- Emerging Trends in Software Development