Programming Paradigms

Dr Ian Cornelius

Hello

Hello (1)

Learning Outcomes

1. Students are expected to understand the term and meaning behind programming paradigms
2. Students should be able to demonstrate their understanding of the most common programming paradigms in their programming lab activities

Programming Paradigms

Programming Paradigms (1)

- Different styles in which a program (or programming language) can be organised
- It consists of certain structures, features and opinions on how common programming problems can be solved
- There are many paradigms, with each being better suited for a given type of problem
    - different paradigms may be used for different projects etc
- Paradigms are **not** a language or a tool
    - you do not build anything with a paradigm
    - they are more akin to a set of ideals and guidelines
- Paradigms are not mutually exclusive
    - i.e. some languages may use more than one paradigm

Programming Paradigms (2)

Programming Languages and Programming Paradigms

- Programming languages are generally not tied to a specific paradigm
- Some have been built with certain paradigms in mind
  - e.g. Haskel and functional programming
- There are languages with a “multi-paradigm” approach
  - code can be adapted to fit one paradigm or another
  - e.g. JavaScript and Python

Programming Paradigms (3)

Types of Programming Paradigms

- There are two types of paradigms:
  1. Imperative
  2. Declarative
- Each paradigm depends upon the:
  - programming language features
  - style of organising the sourcecode

Imperative Paradigms

Imperative Paradigms (1)

- This is a command-driven paradigm
- The programming language directs the program execution
  - executed as a sequence of statements, one-by-one
- Consists of a set of program statements
  - each statement will direct the computer to perform a specific task
- The programmer elaborates each statement in detail, here each statement directs the following:
  - what needs to be done?
  - how will it be done?
- Examples of imperative paradigm programming languages:
  - e.g. Fortran, Pascal and Basic
- There are several types of imperative paradigms:
  1. Procedural Programming
  2. Object-Oriented Programming
  3. Parallel Processing Approach

```
sum = 0
sum += 1
sum += 2
sum += 3
sum += 4
sum += 5
```

```
The sum is: 15
```

Imperative Paradigms (2)

Procedural Programming

- Allows the splitting of instructions into *procedures*
- Procedures are **not** functions
    - functions return a value, whereas procedures do not
- The primary purpose is to accomplish a given task
    - intended to cause a desired **side effect**
- Examples of procedural paradigm programming languages:
    - e.g. C, C++, Java and Pascal
- Advantages of procedural programming are:
    - code is compact and reusable
    - breaks problems into smaller problems
    - utilises CPU storage effectively

```
sum = 0
for i in range(1, 6):
    sum += i
```

```
The sum is: 15
```

Imperative Paradigms (3)

Object-Oriented Programming

- A common paradigm approach often used in programming
- It consists of unique advantages, such as modularity of the code
- Key characteristics of this approach are:
    - classes, abstraction, encapsulation, inheritance and polymorphism
- An object is an instance of a class
- The object will consist of:
    - **attributes/states**: the data associated with the object
    - **methods/behaviours**: the actions/functions that the object can perform
- Examples of object-oriented paradigm programming languages:
    - e.g. C++, Java, Ruby, and Python

```python
class Calculator:
    def __init__(self, n):
        self.n = n
        self.__sum = 0

    def add_numbers(self):
        for i in range(0, self.n + 1):
            self.__sum += i
        return self.__sum
```

```
The sum is: 15
```

Imperative Paradigms (4)

Parallel Processing Approach

- Concerned with processing of the instructions across multiple processors
- Implies basic assumptions about the fundamental hardware components
  - i.e. how are the processors and memory connected to one another?
- Parallel computing *always* involves multiple processors
  - could also refer to *cores* on a CPU
- Two main methods of parallel programming:
  1. Shared Memory
  2. Distributed Memory
- Refer to examples and demonstrations in the next lecture, Threading in Python

Declarative Paradigms

Declarative Paradigms (1)

- Focuses on the logic of the program and the result
- The control flow is not an important element of the program
- The opposite method of programming compared to imperative programming
  - i.e. the programmer does not give instructions about how a task should be executed
- The JavaScript language consists of some declarative functions
  - i.e. `filter`, `sort`, `reduce` and `map`
- There are a few common types of declarative programming:
  1. Functional Programming
  2. Logical Programming

Declarative Paradigms (2)

Functional Programming

- Concerned about the software/script construction by creating pure functions
- It avoids the concept of shared variables and mutable data
- Has an emphasis on expressions and declarations
  - instead of execution of statements
- There are no local or global variables which in-turn reduce side effects
  - i.e. changes cannot be made to variables outside the functions environment
- Increased modularity of the code and a lack of side effects improves the code maintainability
- Examples of object-oriented paradigm programming languages:
  - e.g. JavaScript, Haskell and Python

```python
def is_prime(n):
    if n < 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

```
is_prime(3) -> True
is_prime(15) -> False
```

Declarative Paradigms (2)

Logical Programming

- A programming paradigm based upon logic and control
- Foundations lay within the mathematical logic
  - programming statements express facts and rules about problems in a system
  - i.e. z is `True` if x and y are also `True`
- It serves as a framework through which computers can produce certain results
- Instead of being explicitly told what to do
  - a computer is given boundaries within which it should be considering
- It can be used to express knowledge that does not depend upon the implementation
  - programs are more flexible, compressed and understandable

Goodbye

Goodbye (1)

Questions and Support

- Questions? Post them on the **Community Page** on Aula
- Additional Support? Visit the Module Support Page
- Contact Details:
  - Dr Ian Cornelius, ab6459@coventry.ac.uk