20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by «S.Nishanth»
«CB.EN.U4CYS21050»
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

AMRITA
VISHWA VIDYAPEETHAM

Outline

Metaprogramming

Metaprogramming paradigm involves writing programs which manipulate other programs as their data.
The simplest example of metaprogramming tool is any compiler, since it converts code written in high-level language into low-level machine language or assembly language.
However, term “metaprogramming” usually implies that one language is used as both metalanguage and object language, and moreover, such usage is provided by language design

Meta programming - Ruby

Ruby, being an object-oriented and dynamic programming language, has built-in support for metaprogramming techniques. Metaprogramming in Ruby can be used in a variety of ways:

1. Dynamically defining methods and classes
2. Modifying existing classes and methods
3. Inspecting and manipulating code objects

Metaprogramming makes Ruby code more powerful, flexible, and expressive. It can also create new features and functionality for the Ruby language itself.

Methods in Metaprogramming (Ruby)

In Ruby, metaprogramming refers to the ability of a program to manipulate or modify itself during runtime. This is often achieved through the use of reflective features, such as introspection and dynamic method creation. Below are some common methods and techniques used in metaprogramming in Ruby:

- `define_method`
- `method_missing`
- `send`
- `class_eval`
- `module_eval`
- `define_singleton_method`
- `const_missing`

Example:

The define_method is a built-in Ruby method that is commonly used in metaprogramming. This method allows developers to define new methods at runtime, which can be incredibly useful for generating dynamic code.

```
class Square end
Square.define_method(:area) do |length|
puts "Area = #length * length" end
Square.new.area(5) # Output: Area = 25
```

Functional programming

Functional programming takes the concept of functions a little bit further. In functional programming, functions are treated as first-class citizens, meaning that they can be assigned to variables, passed as arguments, and returned from other functions.

Functional Programming in F#

F# is a functional-first programming language developed by Microsoft Research. It combines functional programming with object-oriented and imperative programming paradigms and is designed to be a succinct and expressive language that runs on the .NET platform. Here's a concise list of key features and concepts related to functional programming in F#:

- `Immutability`
- `First-class function`
- `Pure Functions`
- `Higher-Order Functions`
- `module_eval`
- `Recursion`
- `List Comprehensions`
- `Pipeline Operator`

Example:

Pure function is a function that, for the same inputs, will always produce the same outputs and has no side effects. This property makes pure functions predictable, testable, and facilitates reasoning about code behavior.
This is a non pure function:

```
let mutable value = 1
let addOneToValue x = x + value
```

This is a pure function:

```
let add x y = x + y
// Example usage
let result = add 3 5
printfn "Result:
```

Comparison and Discussions

Similarities:

- Ruby's metaprogramming and F#'s functional constructs provide dynamic features, allowing flexibility in code execution.
- Both paradigms offer flexibility in different ways — Ruby with dynamic modifications, and F# with functional composition.

Mutability:

- Ruby: Supports mutability and runtime modifications of classes and objects.
- F#: Encourages immutability, aiding in reasoning about code and preventing side effects.

Pattern Matching:

- Ruby: Lacks built-in pattern matching, relying on alternative techniques.
- F#: Emphasizes powerful pattern matching for handling complex data structures.

Tooling and Ecosystem:

- Ruby: Mature ecosystem, particularly strong in web development and scripting.
- F#: Integrated into the .NET ecosystem, suitable for a range of applications, especially in Microsoft environments.

References

1. https://www.freecodecamp.org/news/an-introduction-to-programming-paradigms
2. http://progopedia.com/paradigm/metaprogramming/
3. https://iue.tuwien.ac.at/phd/heinzl/node32.html
4. https://www.scaler.com/topics/metaprogramming-in-ruby
5. https://www.shakacode.com/blog/metaprogramming-in-ruby/
6. https://hackr.io/blog/functional-programming
7. https://books-library.net/files/books-library.net-01301832Oz1H4.pdf