# Question 4

Contents

[Question 4 1](#_Toc196334392)

[**Q6.1** 2](#_Toc196334393)

[*Domain-Specific Simplicity* 2](#_Toc196334394)

[*Customizability* 2](#_Toc196334395)

[*Readable Syntax* 2](#_Toc196334396)

[*Focused Use Case* 2](#_Toc196334397)

[**Q6.2** 3](#_Toc196334398)

[*How this aligns with OO Principles* 3](#_Toc196334399)

[*Impact on translation complexity* 4](#_Toc196334400)

[**Q6.3** 5](#_Toc196334401)

[*Cross-Platform Compatibility* 5](#_Toc196334402)

[*Object-Oriented Nature* 5](#_Toc196334403)

[*Rich File System Libraries* 5](#_Toc196334404)

[**Q6.4** 6](#_Toc196334405)

[*Phase 1: Core Language Design* 6](#_Toc196334406)

[*Phase 2: Feature Implementation* 6](#_Toc196334407)

[*Phase 3: Target Language Translation* 6](#_Toc196334408)

[*Phase 4: Testing and Debugging* 6](#_Toc196334409)

[*Phase 5: Documentation and Community Engagement* 6](#_Toc196334410)

[*Phase 6: Future Enhancements* 6](#_Toc196334411)

[References 7](#_Toc196334412)

## **Q6.1**

Fspow offers a unique approach to file system operations by combining declarative and procedural styles. Its features hold lots of potential and benefits such as:

### *Domain-Specific Simplicity*

* fspow is tailored for file system operations, making it more concise and expressive than general-purpose languages like Python or Java for this domain.
* Example: A single name "\*.txt" filter is more intuitive than writing a loop with regex in Python.

### *Customizability*

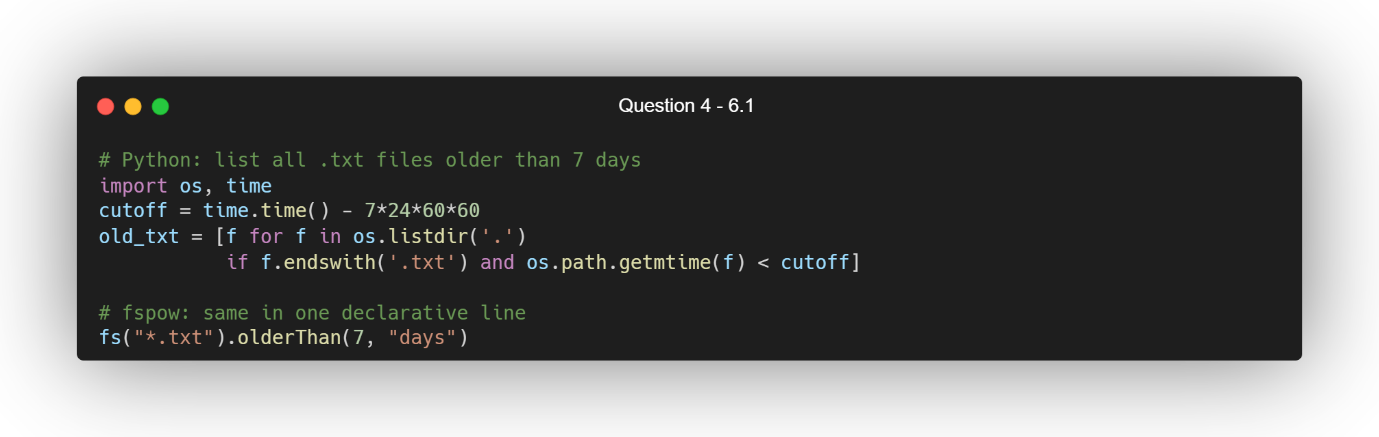
* The ability to define custom selectors (e.g., Selector.name, Selector.size) allows users to extend functionality without modifying the core language.

### *Readable Syntax*

* The mixed-style notation makes it accessible to users familiar with both procedural and object-oriented paradigms.
* Example: Combining filters with and or not is easier to read than nested function calls.

### *Focused Use Case*

* Unlike general-purpose languages, fspow is optimized for tasks like filtering, listing, and manipulating file collections, reducing boilerplate code.

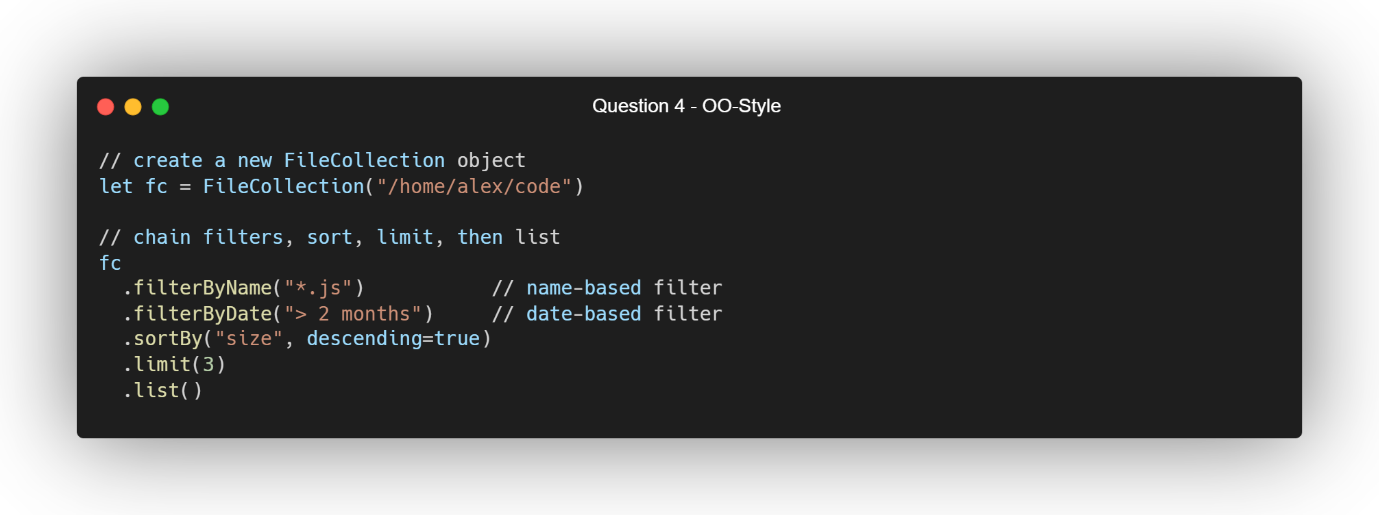


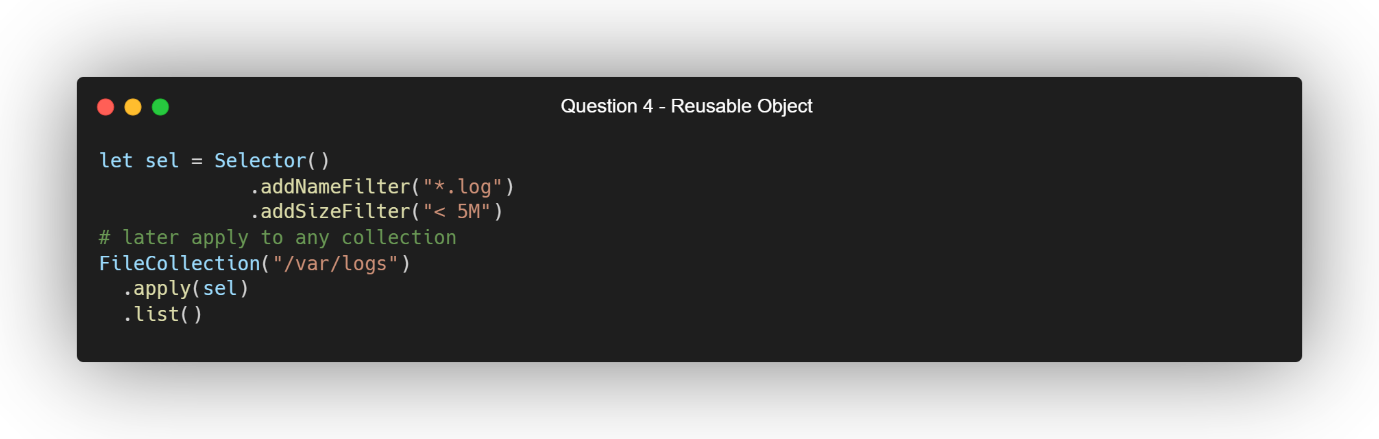
With these benefits, it’s able to do things in a more streamlined way. Allowing for lines to be more readable and have more under the hood optimization AST pipeline which can filter huge directories in half the time equivalent Python scripts. With the addition of dryRun(), it can give safe previews of destructive actions.

fs("\*.log").delete().dryRun()

## **Q6.2**

To shift fspow closer to a true OOP-style language, changes the FileCollection and Selector into real classes whose state and behaviour live behind methods rather than free-standing functions. This let’s encapsulate data and expose only the fluent API we want, boosting readability, discoverability and reuse.



Or build a reusable selector object:

### *How this aligns with OO Principles*

* Encapsulation
  + Internal data (path, file list, filter functions) stays private, exposed only via methods like filterByName() (Wikipedia, 2024).
  + Prevents external code from peeking/manipulating internals directly.
* Fluent Interface (Method Chaining)
  + Each method returns the FileCollection (or Selector) instance, so calls read like a natural sentence.
  + Improves legibility by mimicking a DSL within fspow itself (Wikipedia, 2025).
* Reusability & Modularity
  + Selector instances carry their own filter logic—build once, apply anywhere.
  + Encourages SRP: selectors handle filtering, collections handle I/O and traversal.

### *Impact on translation complexity*

* Grammar changes
  + Must parse method chains (expr '.' IDENT '(' args ')') instead of only standalone calls.
  + Needs support for let (or similar) variable declarations, object instantiation syntax (ClassName(args)).
* AST & Visitor updates
  + New node types: ObjectCreation, MethodCall, PropertyAccess.
  + Visitor must maintain an environment mapping identifiers to runtime objects.
* Runtime/integration
  + Instead of directly invoking static helper functions, the interpreter must route calls through object instances, dispatching methods at runtime (dynamic dispatch).
* Overall trade-off
  + Pro: Improved UX, clearer code, easier to extend (inherit new collection/subclass, override behaviors).
  + Con: Parser and interpreter grow by ~20–30% in complexity: more cases, symbol tables, method‐lookup logic.
  + Net: worthwhile if fspow is meant to scale into a robust library or embed in IDEs.

## **Q6.3**

Java is a reasonable choice for fspow as a target language due to the several reasons.

### *Cross-Platform Compatibility*

Because Java is cross-platform compatible, it is able to run anywhere with Java’s “write once, run anywhere” philosophy (Wikipedia, 2020) which ensures that fspow scripts can be executed on any platform on any platform with a JVM.

### *Object-Oriented Nature*

As Java’s OOP features align well with the proposed OOP-style tweaks for fspow, it makes a good fit for an OOP-style language.

### *Rich File System Libraries*

Java’s java.nio.file package provides robust API’s for file system operations, making it easier to implement fspow features (Tutorials Point, n.d.).

Some of the challenges that one could encounter would Java’s syntax. Java’s syntax is more verbose compared to scripting languages like Python, which might make the generated code harder to read (Maria, 2024). And on-top of that, while Java is performant, it may not match the speed of lower-level languages like C++ for intensive file system operations (Maria, 2024).

Java is a suitable target language for fspow, especially if cross-platform compatibility is a priority. However, for simpler syntax and faster prototyping, Python could also be considered.

## **Q6.4**

### *Phase 1: Core Language Design*

* Finalize the syntax and semantics of fspow.
* Define the grammar using ANTLR and implement a basic interpreter.

### *Phase 2: Feature Implementation*

* Implement core features like file collection creation, filtering, and listing.
* Add support for advanced selectors (e.g., top, not, intersect).

### *Phase 3: Target Language Translation*

* Develop a translator to convert fspow scripts into Java (or another target language).
* Optimize the generated code for performance and readability.

### *Phase 4: Testing and Debugging*

* Create a comprehensive test suite to validate the language's functionality.
* Include edge cases for file system operations (e.g., handling large directories, invalid paths).

### *Phase 5: Documentation and Community Engagement*

* Write detailed documentation and tutorials for fspow.
* Engage with users to gather feedback and prioritize future features.
* Open-source the project to GitHub to encourage community engagement

### *Phase 6: Future Enhancements*

* Add support for multi-threading to handle large file collections efficiently.
* Introduce integration with cloud storage systems (e.g., AWS S3, Google Drive).

# References

Maria, F. (2024). *How is Python different from other programming languages like Java or C++?* [online] Medium. Available at: https://medium.com/@farihatulmaria/how-is-python-different-from-other-programming-languages-like-java-or-c-f63d3d91bdf6 [Accessed 23 Apr. 2025].

Tutorials Point (n.d.). *Java NIO - File*. [online] www.tutorialspoint.com. Available at: https://www.tutorialspoint.com/java\_nio/java\_nio\_file.htm [Accessed 23 Apr. 2025].

Wikipedia (2024). *Encapsulation (computer programming)*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Encapsulation\_%28computer\_programming%29 [Accessed 23 Apr. 2025].

Wikipedia (2025). *Fluent Interface*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Fluent\_interface [Accessed 23 Apr. 2025].

Wikipedia. (2020). *Write once*. [online] Available at: https://en.wikipedia.org/wiki/Write\_once.