1. Account gitHub
2. Account kaggle
3. Download anaconda : jupter note book
4. All company and startup that use AI in Egypt

|  |  |  |
| --- | --- | --- |
| **Company** | **Website** | **information about the company** |
| AION | [https://aion.ai/ ↗](https://aion.ai/) | AION is a Cairo-based startup that uses AI to help businesses improve their customer service. |
| Cognito | [https://cognito.io/ ↗](https://cognito.io/) | Cognito is a Cairo-based startup that uses AI to help businesses automate their marketing and sales processes. |
| Elmenus | [https://elmenus.com/ ↗](https://elmenus.com/) | Elmenus is a Cairo-based food delivery company that uses AI to help customers find the best restaurants and dishes. |
| Farafina | [https://farafina.ai/ ↗](https://farafina.ai/) | Farafina is a Cairo-based startup that uses AI to help businesses create personalized marketing campaigns. |
| Horus | [https://horus.ai/ ↗](https://horus.ai/) | Horus is a Cairo-based startup that uses AI to help businesses detect fraud. |
| Kalam | [https://kalam.ai/ ↗](https://kalam.ai/) | Kalam is a Cairo-based startup that uses AI to help businesses translate and localize content. |
| Noor | [https://noor.ai/ ↗](https://noor.ai/) | Noor is a Cairo-based startup that uses AI to help businesses improve their customer service. |
| Sada | [https://sada.ai/ ↗](https://sada.ai/) | Sada is a Cairo-based startup that uses AI to help businesses manage their inventory. |
| Talabat | [https://talabat.com/ ↗](https://talabat.com/) | Talabat is a Cairo-based food delivery company that uses AI to help customers find the best restaurants and dishes. |

1. principles of clean code

The principles of clean code refer to a set of guidelines and best practices that help developers write code that is easy to understand, maintain, and modify. Clean code is readable, concise, and efficient, making it easier for developers to collaborate, debug, and enhance the codebase. Here are some key principles of clean code:

* 1. **Simplicity**: Keep the code simple and straightforward. Avoid unnecessary complexity, convoluted logic, and excessive use of design patterns. Write code that is easy to understand and follow.
  2. **Readability**: Make the code easy to read and comprehend. Use meaningful and descriptive names for variables, functions, and classes. Follow a consistent coding style and formatting conventions. Write comments to explain complex or non-obvious parts of the code.
  3. **Single Responsibility Principle (SRP)**: Each class, function, or module should have a single responsibility or do one thing well. Avoid creating large, monolithic functions or classes that handle multiple unrelated tasks. Split complex tasks into smaller, focused functions that are easier to understand and maintain.
  4. **Don't Repeat Yourself (DRY)**: Avoid duplicating code by extracting reusable components. Use functions, classes, and modules to encapsulate common functionality. By eliminating duplication, you improve maintainability and reduce the risk of introducing bugs.
  5. **Small and focused functions**: Functions should be small and focused on a specific task. They should ideally do one thing and do it well. Small functions are easier to test, understand, and reason about. Aim for functions that fit within a single screen or less.
  6. **Avoid long methods**: Long methods are harder to understand and maintain. Aim for shorter methods that have a clear and specific purpose. If a method becomes too long or complex, consider refactoring it into smaller, more manageable pieces.
  7. **Testing**: Write automated tests to verify the correctness of your code. Follow Test-Driven Development (TDD) or similar practices to ensure that your code is testable and that you have good test coverage. Well-tested code is more reliable, maintainable, and easier to refactor.
  8. **Consistent naming and formatting**: Use consistent naming conventions for variables, functions, and classes. Choose descriptive names that convey the purpose and meaning of the code. Use indentation, spacing, and formatting consistently to improve readability.
  9. **Avoid unnecessary comments**: Write code that is self-explanatory and minimize the need for comments. While comments can be useful for explaining complex algorithms or documenting important details, strive to write code that is so clear and expressive that comments are rarely needed.
  10. **Keep dependencies minimal**: Minimize dependencies between modules, classes, and functions. Reduce coupling and strive for loose coupling between components. This improves the flexibility, testability, and maintainability of the code.