1. **Meaningful Naming:**
    - Using describing names for variables, functions, and classes enhances code readability.

```
# Bad
a = 10
b = calculate_sum(a)

# Good
initial_value = 10
total_sum = calculate_total(initial_value)
```

2. **Single Responsibility Principle:**
    - Each function and class should have a single responsibility, to make the code more modular and maintainable.

```
def process_data(file_path):
    # ... code for data processing and reading ...

# Good
def read_file(file_path):
    # ... code to read file ...

def process_data(data):
    # ... code for data processing ...
```

3. **Formatting:**
    - Using tools to correct spacing and new lines, removing useless empty lines and spaces, such as black, pylint.
4. **Error Handling:**
    - Implementing proper error handling codes to prevent unwanted crashes and provide meaningful error messages to make it easier to debug.
5. **Comments:**
    - Using comments to explain the reasons behind decisions.
6. **Version Control:**
    - Committing regularly over progress with meaningful messages and avoiding large commits.

7. **Encapsulation and Modularity:**
    - Encapsulating methods makes code easier to maintain. When you need to change something it is easier to find method and making change only once saves time and makes you sure that you have not forgot to change others.
8. **Avoid Duplicate Codes:**
    - Refactor repetitive code into functions or modules to reduce repetitions.
9. **Readable Code is Better Than Short Code:**
    - Prefer code readability over complex solutions. Write code that is easy to understand for others (and for you in the future).
10. **Test-Driven Development:**
    - Testing new features, methods, functions immediately makes it easier to locate errors and debug.
11. **Short Functions:**
    - Aim for functions that perform a single, clear task and are ideally no longer than a screenful.
12. **Learn from others:**
    - Actively participate in code reviews, providing constructive feedback and learning from others.
13. **Deep Nesting:**
    - Limit the numbers of nested structures (if statements, loops) to enhance code clarity and performance.
14. **Documentation:**
    - Documentation is important to help users to learn to use your program.