Functions in Python come with various advantages and some disadvantages. Here are 10 advantages followed by 5 disadvantages of Python functions, each with an example:

Advantages:

1. **Modularity and Reusability**:
   - **Example:** You can create a function to calculate the factorial of a number and reuse it throughout your program.
   ```python
   def factorial(n):
       if n == 0:
           return 1
       else:
           return n * factorial(n - 1)
   ```

2. **Code Organization**:
   - **Example:** Grouping related code into functions makes your codebase more organized and easier to maintain.
   ```python
   def read_data(file_name):
       # Code to read data from a file

   def process_data(data):
       # Code to process data

   def write_data(output_file, processed_data):
       # Code to write processed data to an output file
   ```

3. **Abstraction**:
   - **Example:** You can hide complex implementation details and expose a simplified interface.
   ```python
   def calculate_average(numbers):
       return sum(numbers) / len(numbers)
   ```

4. **Code Reusability**:
   - **Example:** Functions can be reused in different parts of your program or even in other programs.
   ```python
   def find_max(numbers):
       return max(numbers)
   ```

5. **Testing and Debugging**:
   - **Example:** You can isolate and test individual functions, making it easier to identify and fix issues.
   ```python
   def divide(a, b):
       if b == 0:
           raise ValueError("Cannot divide by zero")
       return a / b
   ```

6. **Parameterization**:
   - **Example:** Functions allow you to pass arguments to customize behavior.
   ```python
   def greet(name):
       print(f"Hello, {name}!")

   greet("Alice")
   greet("Bob")
   ```

7. **Scoping**:
   - **Example:** Functions have their own variable scope, preventing variable name conflicts.
   ```python
   def calculate_area(radius):
       pi = 3.14159265359
       return pi * radius * radius
   ```

8. **Encapsulation**:
   - **Example:** You can encapsulate functionality within functions, making it easier to change the implementation without affecting other parts of the code.
   ```python
   def connect_to_database():
       # Code to establish a database connection

   def execute_query(query):
       # Code to execute a database query
   ```

9. **Parameter Validation**:
   - **Example:** Functions can validate input parameters before processing.
   ```python
   ```

```python
def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
```

10. **Readability**:
    - **Example:** Well-named functions make your code more readable and self-explanatory.
    ```python
    def calculate_total_price(price, quantity):
        return price * quantity
    ```

Disadvantages:

1. **Performance Overhead**:
   - Functions can introduce a small performance overhead due to function call and return operations, which can be significant in performance-critical applications.

2. **Complexity**:
   - Using too many functions can make the codebase complex and harder to understand, especially for small scripts.

3. **Global Variables**:
   - Functions can access and modify global variables, potentially leading to unintended side effects.

4. **Memory Usage**:
   - Functions create stack frames, consuming memory, especially if you have many recursive function calls.

5. **Overhead of Function Calls**:
   - In some cases, excessive function calls can result in a performance penalty, as Python has to manage the call stack.

While functions offer numerous advantages in Python development, it's essential to balance their use with the specific requirements of your project and be aware of potential drawbacks.