The key differences between decorator functions and generator functions:

1. Purpose:

- Decorator functions are used to modify or enhance the behavior of other functions without changing their source code.

- Generator functions are used to create iterators that generate a sequence of values over time.

2. Syntax and Usage:

- Decorators use the "@" symbol followed by the decorator name above a function definition. They "wrap" the function they're decorating.

- Generators use the "yield" keyword inside the function body to produce values.

3. Execution:

- Decorators are executed immediately when the decorated function is defined.

- Generators are lazy - they only produce values when iterated over.

4. Return Value:

- Decorators typically return a new function that wraps the original one.

- Generators return a generator object that can be iterated over.

5. Use Cases:

- Decorators are often used for cross-cutting concerns like logging, timing, authentication, or caching.

- Generators are used for creating efficient iterables, especially for large data sets or infinite sequences.

6. State Preservation:

- Decorators don't inherently preserve state between calls (though they can be designed to do so).

- Generators automatically preserve their state between yields, allowing them to resume where they left off.

7. Memory Efficiency:

- Decorators don't generally impact memory usage significantly.

- Generators can be very memory-efficient as they produce values on-demand rather than storing an entire sequence in memory.

8. Flexibility:

- Decorators can be applied to any function and even to classes.

- Generators are specifically for creating iterables.

Here's a quick example to illustrate the difference with Python Code:

 # Decorator

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
# Output:
# Something is happening before the function is called.
# Hello!
# Something is happening after the function is called.
```

 # Generator

```
def count_up_to(n):
    i = 1
    while i <= n:
        yield i
        i += 1


for number in count_up_to(5):
    print(number)
# Output:
# 1
# 2
# 3
# 4
# 5
```

In summary, while both are powerful Python features, decorators are used to modify function behavior, while generators are used to create efficient iterables. They serve different purposes and are used in different scenarios.