In [1]:

```python
import time
import logging
from functools import wraps
from datetime import datetime

# Set up
logging.basicConfig(level=logging.INFO, format='%(message)s')

def log_execution(log_errors_only=False, output_destination='console'):
    """
    A decorator to log detailed information about function execution.

    Parameters:
        log_errors_only (bool): If True, only logs errors; otherwise, logs every call.
        output_destination (str): Specifies where to output logs ('console' or file path
    """

    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            func_name = func.__name__
            start_time = time.time()
            timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

            # Initialize
            if output_destination == 'console':
                logger = logging.getLogger(func_name)
            else:
                logger = logging.getLogger(func_name)
                handler = logging.FileHandler(output_destination)
                logger.addHandler(handler)

            # result
            result = None
            exception = None

            try:
                if not log_errors_only:
                    logger.info(f"[{timestamp}] Starting '{func_name}' with arguments: a

                result = func(*args, **kwargs)

            except Exception as e:
                exception = e
                logger.error(f"[{timestamp}] Exception in '{func_name}': {e}")
                raise

            finally:
                end_time = time.time()
                elapsed_time = end_time - start_time
                if exception is None and not log_errors_only:
                    logger.info(f"[{timestamp}] Finished '{func_name}' in {elapsed_time:
                elif exception is not None and log_errors_only:
                    logger.error(f"[{timestamp}] Error in '{func_name}' (elapsed time: {


                if output_destination != 'console':
```

```
                    handler.close()
                    logger.removeHandler(handler)

            return result
        return wrapper
    return decorator
```

In [2]:

```
@log_execution(log_errors_only=False, output_destination='log.txt')
def sample_function(x, y):
    return x / y

try:
    sample_function(10, 5)  # Normal call
    sample_function(10, 0)  # This will raise an exception
except ZeroDivisionError:
    pass
```

```
[2024-11-12 23:17:02] Starting 'sample_function' with arguments: args=(10, 5), kwargs={}
[2024-11-12 23:17:02] Finished 'sample_function' in 0.0020s with result: 2.0
[2024-11-12 23:17:02] Starting 'sample_function' with arguments: args=(10, 0), kwargs={}
[2024-11-12 23:17:02] Exception in 'sample_function': division by zero
```

In [ ]:

```

```