```
from functools import wraps
from swarms.tools.py_func_to_openai_func_str import (
    get_openai_function_schema_from_func,
)
from swarms.utils.loguru_logger import logger



def tool(
    name: str = None,
    description: str = None,
    return_dict: bool = True,
    verbose: bool = True,
    return_string: bool = False,
    return_yaml: bool = False,
):
    """
    A decorator function that generates an OpenAI function schema.

    Args:
        name (str, optional): The name of the OpenAI function. Defaults to None.
        description (str, optional): The description of the OpenAI function. Defaults to None.
        *args: Variable length argument list.
        **kwargs: Arbitrary keyword arguments.

    Returns:
        dict: The generated OpenAI function schema.
```

"""

```
def decorator(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        try:
            # Log the function call
            logger.info(f"Creating Tool: {func.__name__}")

            # Assert that the arguments are of the correct type
            assert isinstance(name, str), "name must be a string"
            assert isinstance(
                description, str
            ), "description must be a string"
            assert isinstance(
                return_dict, bool
            ), "return_dict must be a boolean"
            assert isinstance(
                verbose, bool
            ), "verbose must be a boolean"

            # Call the function
            func(*args, **kwargs)

            # Get the openai function schema
```

```
    tool_name = name if not None else func.__name__
    schema = get_openai_function_schema_from_func(
        func, name=tool_name, description=description
    )

    # Return the schema
    if return_dict:
        return schema
    elif return_string is True:
        return str(schema)
    elif return_yaml is True:
        # schema = YamlModel().dict_to_yaml(schema)
        return schema
    else:
        return schema

except AssertionError as e:
    # Log the assertion error
    logger.error(f"Assertion error: {str(e)}")
    raise

except Exception as e:
    # Log the exception
    logger.error(f"Exception occurred: {str(e)}")
    raise
```

```
    return wrapper


return decorator
```