```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct  1 23:22:47 2023

@author: LAKSHMIPRIYA Anil
"""
from functools import wraps


# Decorator to trace execution of recursive function
def trace(func):

    # cache function name
    recc_func_name = func.__name__

    recc_depth_indicator = '|  '

    # current recursion depth
    trace.recursion_depth = 0

    @wraps(func)        # Decorator to wrapper function to increase readability.
    def traced_wrapper(*args, **kwargs):
        """A wrapper function to extend the capability of passed function by printing its trace."""

        arg_params = ', '.join(map(repr, args))
            # repr() returns a printable representation of an object i.e it returns string with the quotes
        kwarg_params = ', '.join(f'{k}={v!r}' for k, v in kwargs.items())
            # '!r'  is a formatting specifier in an f-string called the representation specifier
                    #with the same functionality as repr()
        if len(kwarg_params) > 0:
            kwarg_params = ', ' + kwarg_params

        # Print the function name and its arguments
        print(f'{recc_depth_indicator * trace.recursion_depth}|-- {recc_func_name}({arg_params}{kwarg_params})')

        # Increment the recursion depth value
        trace.recursion_depth += 1

        # Call the original function
        result = func(*args, **kwargs)

        # Go one step ourside the current recursion depth
        trace.recursion_depth -= 1

        # Print the value being returned
        print(f'{recc_depth_indicator * (trace.recursion_depth + 1)}|-- return {result}')

        return result

    return traced_wrapper



def factorial(n):
    """This function finds the factorial of number n."""
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

@trace
def fibonacci(n):
    """This function finds the fibonacci value till number n."""
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

@trace
def accumufact(n, fact=1):
    if n == 1:
        return fact
    else:
        return accumufact(n-1, n*fact)

@trace
def gcd(p, q):
    if q == 0:
        return p
    else:
        return gcd(q, p%q)
```

```
# Call the traced recursive function
factorial = trace(factorial)
final_result = factorial(7)
print(final_result, '\n\n')

print(fibonacci(3), '\n\n')     # Equivalent syntax to trace(fibonacci)
                                # since @trace decorator name is mentioned above callable fibonacci function
print(accumufact(5), '\n\n')
print(gcd(165,27))
```

```
|-- factorial(7)
|  |-- factorial(6)
|  |  |-- factorial(5)
|  |  |  |-- factorial(4)
|  |  |  |  |-- factorial(3)
|  |  |  |  |  |-- factorial(2)
|  |  |  |  |  |  |-- factorial(1)
|  |  |  |  |  |  |  |-- return 1
|  |  |  |  |  |  |-- return 2
|  |  |  |  |  |-- return 6
|  |  |  |  |-- return 24
|  |  |  |-- return 120
|  |  |-- return 720
|  |-- return 5040
5040


|-- fibonacci(3)
|  |-- fibonacci(2)
|  |  |-- fibonacci(1)
|  |  |  |-- return 1
|  |  |-- fibonacci(0)
|  |  |  |-- return 0
|  |  |-- return 1
|  |-- fibonacci(1)
|  |  |-- return 1
|  |-- return 2
2


|-- accumufact(5)
|  |-- accumufact(4, 5)
|  |  |-- accumufact(3, 20)
|  |  |  |-- accumufact(2, 60)
|  |  |  |  |-- accumufact(1, 120)
|  |  |  |  |  |-- return 120
|  |  |  |  |-- return 120
|  |  |  |-- return 120
|  |  |-- return 120
|  |-- return 120
120


|-- gcd(165, 27)
|  |-- gcd(27, 3)
|  |  |-- gcd(3, 0)
|  |  |  |-- return 3
|  |  |-- return 3
|  |-- return 3
3
```