

# *Errors and Visualization:*

So, there are instances when we deal with errors in Programming and bas times we don't really understand what the Error is trying to mean and just sit Clueless.

Now, to deal with that we are going to see a better way of doing the same with some different tools. Which we will cover in this documentation today.

## **Error visualization:**

### **Loguru**

So, to help us with our Error Visualization in Python we have a Python Module called **Loguru**. It helps us finding the Errors where they are and explain it to us in a much better and Understandable way and highlights the portions and also shows the values we have entered for each instance or a variable.

Let us see a Example of what we have just read.

To, Begin with first we will install the **Loguru** module we are talkig about.

**Installation Instructions** → `pip install loguru`

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

Loading personal and system profiles took 1187ms.
@Apurba → Desktop pip install loguru
Collecting loguru
  Downloading loguru-0.5.3-py3-none-any.whl (57 kB)
    | 57 kB 462 kB/s
Requirement already satisfied: colorama≥0.3.4 in c:\users\apurba\appdata\roaming\python\python39\site-packages (from lo
guru) (0.4.4)
Collecting win32-setctime≥1.0.0
  Downloading win32_setctime-1.0.3-py3-none-any.whl (3.5 kB)
Installing collected packages: win32-setctime, loguru
Successfully installed loguru-0.5.3 win32-setctime-1.0.3

```

*Next up, we will see how will it helps us and how do we implement the following module in our code. →*

*Now, what you see below is a Sample code of a division containing Functions, here we pass a list to a function and it divided each and every number with its succeeding number, and every time for doing that it takes help of another Function by Calling it.*

```

from itertools import combinations

def division(num1: int, num2: int):
    return num1/num2

def divide_numbers(num_list: list):
    for comb in combinations(num_list, 2):
        num1, num2 = comb
        res = division(num1, num2)
        print(f"{num1} divided by {num2} is equal to {res}.")

if __name__ == '__main__':
    num_list = [2, 1, 0]
    divide_numbers(num_list)

```

*Now, if we try running this Code, if you guys can notice it will throw a Error as we have passed '0' as an in the list as a Argument in the Function which will eventually gives us ZeroDivisionError. Now we know the error but let's suppose that this would be a long codebase and you can't really understand the bug in your code in that case let's first see the Error which we all will get.*

## Error:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

Loading personal and system profiles took 1426ms.
@Apurba → Desktop & 'C:\Users\Apurba\AppData\Local\Programs\Python\Python39\python.exe'
thonFiles\lib\python\debugpy\launcher' '54542' '--' 'c:\Users\Apurba\Desktop\test.py'
2 divided by 1 is equal to 2.0.
Traceback (most recent call last):
  File "c:\Users\Apurba\Desktop\test.py", line 15, in <module>
    divide_numbers(num_list)
  File "c:\Users\Apurba\Desktop\test.py", line 9, in divide_numbers
    res = division(num1, num2)
  File "c:\Users\Apurba\Desktop\test.py", line 4, in division
    return num1/num2
ZeroDivisionError: division by zero
@Apurba → Desktop
```

Now, this is something we all have expected, but what if we had a Visual Representation, let's see how can we achieve that with **Loguru**

To do that we made a simple change into our code, we just import the **Loguru** Module into our script and then Exactly tracks the part which gives the error, in this case we are getting error from the **Function** itself.

```

1  from itertools import combinations
2  from loguru import logger
3
4  def division(num1: int, num2: int):
5      return num1/num2
6
7  @logger.catch
8  def divide_numbers(num_list: list):
9      for comb in combinations(num_list, 2):
10         num1, num2 = comb
11         res = division(num1, num2)
12         print(f"{num1} divided by {num2} is equal to {res}.")
13
14
15  if __name__ == '__main__':
16      num_list = [2, 1, 0]
17      divide_numbers(num_list)
18

```

*And by doing this, the Error we get here is,*

```

8Apurba → Desktop  c:; cd 'c:\Users\Apurba\Desktop'; & 'C:\Users\Apurba\AppData\Local\Programs\Python\Python39\python.exe' 'c:\Users\Apurba\.vscode\extensions\ms
-python.python-2021.4.765268190\pythonFiles\lib\python\debugpy\launcher' '54686' '--' 'c:\Users\Apurba\Desktop\test.py'
2 divided by 1 is equal to 2.0.
2021-05-08 22:22:52.481 | ERROR    | __main__:<module>:17 - An error has been caught in function '<module>', process 'MainProcess' (11220), thread 'MainThread' (11
828):
Traceback (most recent call last):
> File "c:\Users\Apurba\Desktop\test.py", line 17, in <module>
  divide_numbers(num_list)
  |   |   |   |   |
  |   |   |   |   +-- [2, 1, 0]
  |   |   |   |   <function divide_numbers at 0x000001F5D3821C10>
  |   |   |   |
  |   |   |   +-- File "c:\Users\Apurba\Desktop\test.py", line 11, in divide_numbers
  |   |   |   |   res = division(num1, num2)
  |   |   |   |   |   |
  |   |   |   |   |   +-- 0
  |   |   |   |   |   <function division at 0x000001F5D1497F70>
  |   |   |   |   |
  |   |   |   |   +-- File "c:\Users\Apurba\Desktop\test.py", line 5, in division
  |   |   |   |   |   return num1/num2
  |   |   |   |   |   |
  |   |   |   |   |   +-- 2
  |   |   |   |   |   0
  |   |   |   |   |   2
ZeroDivisionError: division by zero

```

*How about this Visual Representation Errors and Exactly knowing for which values we are getting the Errors?*

# Code Visualization:

## Snoop

What if there is no error in the code, but we want to figure out what is going on in the code? That is when **snoop** comes in handy.

**Snoop** is a Python package that prints the lines of code being executed along with the values of each variable by adding only one decorator.

## Installation Instructions → pip install snoop

```
@Apurba - Desktop pip install snoop
Collecting snoop
  Downloading snoop-0.3.0.tar.gz (33 kB)
Requirement already satisfied: six in c:\users\apurba\appdata\local\programs\python\python39\lib\site-packages (from snoop) (1.15.0)
Collecting cheap_repr<0.4.0
  Downloading cheap_repr-0.4.5.tar.gz (10.0 kB)
Collecting executing
  Downloading executing-0.6.0-py2.py3-none-any.whl (12 kB)
Collecting asttokens
  Downloading asttokens-2.0.5-py2.py3-none-any.whl (20 kB)
Requirement already satisfied: pygments in c:\users\apurba\appdata\roaming\python\python39\site-packages (from snoop) (2.7.3)
Building wheels for collected packages: snoop, cheap_repr
  Building wheel for snoop (setup.py) ... done
  Created wheel for snoop: filename=snoop-0.3.0-py3-none-any.whl size=27304 sha256=9a9c8b7f6d183817f2f5c414b82de5d311f2f2c9184cbfb988455f23a11e811
  Stored in directory: c:\users\apurba\appdata\local\pip\cache\wheels\4a\c1\35\dc7bfb17a749a6dca762acd4af6da81f82aed51cce1b9e413
  Building wheel for cheap_repr (setup.py) ... done
  Created wheel for cheap_repr: filename=cheap_repr-0.4.5-py3-none-any.whl size=7122 sha256=6bcbac71c43a3627cb3d927f8a8c8b817f03a83c231994aecf2b3857ddd28f99
  Stored in directory: c:\users\apurba\appdata\local\pip\cache\wheels\67\9a\86\121312e4c8c28bb5b241aa6559487a51335c6458324e7f13bf
Successfully built snoop cheap_repr
Installing collected packages: executing, cheap_repr, asttokens, snoop
Successfully installed asttokens-2.0.5 cheap_repr-0.4.5 executing-0.6.0 snoop-0.3.0
```

Next up, we will see how will it helps us and how do we implement the following module in our code. →

Now, here we have a Simple Code defining what a Factorial is and helping us to get the Factorial of a Number using the help of a Function.

**Note: We have used Recursive Function in here in this Code.**

```
def factorial(x: int):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))

if __name__ == "__main__":
    num = 5
    print(f"The factorial of {num} is {factorial(num)}")
```

Now, the Output is pretty predictable, it will print the factorial of the Number just like this -

```
Loading personal and system profiles took 1056ms.
@Apurba → Desktop & 'C:\Users\Apurba\AppData\Local\Programs\Python\Python39\python.exe'
thonFiles\lib\python\debugpy\launcher' '54681' '--' 'c:\Users\Apurba\Desktop\test.py'
The factorial of 5 is 120
@Apurba → Desktop
```

But, what if I were to tell you, there is more you can do with it, but not just fetch the Output and understand how the Code Works in the Background and how does the interpreter runs it in Background. Lets' see how can we achieve that using **Snoop**. Also, we will use **Snoop** here as a **Decorator** as we do in **Python**.

After we implement the Snoop Code the Code we have written will look somewhat like this...

```
import snoop

@snoop
def factorial(x: int):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))

if __name__ == "__main__":
    num = 5
    print(f"The factorial of {num} is {factorial(num)}")
```

After this as we see how the Output looks now, much better right?

```
@Apurba → Desktop c:; cd 'c:\Users\Apurba\Desktop'; & 'C:\Users\Apurba\AppData\Local\Programs\Python\Python2021.4.765268190\pythonFiles\lib\python\debugpy\launcher' '54715' '--' 'c:\Users\Apurba\Desktop\test.py'
22:38:31.12 >>> Call to factorial in File "c:\Users\Apurba\Desktop\test.py", line 22
22:38:31.12 ..... x = 5
22:38:31.12 22 | def factorial(x: int):
22:38:31.12 23 |     if x == 1:
22:38:31.12 26 |         return (x * factorial(x-1))
22:38:31.12 >>> Call to factorial in File "c:\Users\Apurba\Desktop\test.py", line 22
22:38:31.12 ..... x = 4
22:38:31.12 22 | def factorial(x: int):
22:38:31.12 23 |     if x == 1:
22:38:31.12 26 |         return (x * factorial(x-1))
22:38:31.13 >>> Call to factorial in File "c:\Users\Apurba\Desktop\test.py", line 22
22:38:31.13 ..... x = 3
22:38:31.13 22 | def factorial(x: int):
22:38:31.13 23 |     if x == 1:
22:38:31.13 26 |         return (x * factorial(x-1))
22:38:31.13 >>> Call to factorial in File "c:\Users\Apurba\Desktop\test.py", line 22
22:38:31.13 ..... x = 2
22:38:31.13 22 | def factorial(x: int):
22:38:31.13 23 |     if x == 1:
22:38:31.13 26 |         return (x * factorial(x-1))
22:38:31.13 >>> Call to factorial in File "c:\Users\Apurba\Desktop\test.py", line 22
22:38:31.13 ..... x = 1
22:38:31.13 22 | def factorial(x: int):
22:38:31.13 23 |     if x == 1:
22:38:31.13 24 |         return 1
22:38:31.13 <<< Return value from factorial: 1
22:38:31.13 26 |         return (x * factorial(x-1))
22:38:31.13 <<< Return value from factorial: 2
22:38:31.14 26 |         return (x * factorial(x-1))
22:38:31.14 <<< Return value from factorial: 6
22:38:31.14 26 |         return (x * factorial(x-1))
22:38:31.14 <<< Return value from factorial: 24
22:38:31.14 26 |         return (x * factorial(x-1))
22:38:31.14 <<< Return value from factorial: 120
The factorial of 5 is 120
```



*Isn't it great? Look, how better it is to understand what's going inside...*

**Note: The Process using Snoop does take some time but it is better at times when you wanna understand what's going inside.**

## Heartrate

Think suppose we have written a piece of code and we wanna see how that is actually working inside as we really don't get what it's happening as the Code is misbehaving i.e it is not giving the Output we are expecting it to give.

In that case, If you want to visualize which lines are executed and how many times they are executed, try **heartrate**. **Heartrate** is also created by the creator of **snoop**.

## Installation Instructions → **pip install heartrate**

```
@Apurba - Desktop pip install heartrate
Collecting heartrate
  Downloading heartrate-0.2.1.tar.gz (238 kB)
    238 kB 731 kB/s
Requirement already satisfied: pygments in c:\users\apurba\appdata\roaming\python\python39\site-packages (from heartrate) (2.7.3)
Collecting Flask
  Downloading Flask-1.1.2-py2.py3-none-any.whl (94 kB)
    94 kB 264 kB/s
Requirement already satisfied: executing in c:\users\apurba\appdata\local\programs\python\python39\lib\site-packages (from heartrate) (0.6.0)
Requirement already satisfied: asttokens in c:\users\apurba\appdata\local\programs\python\python39\lib\site-packages (from heartrate) (2.0.5)
Requirement already satisfied: six in c:\users\apurba\appdata\local\programs\python\python39\lib\site-packages (from asttokens->heartrate) (1.15.0)
Collecting Werkzeug<=0.15
  Downloading Werkzeug-1.0.1-py2.py3-none-any.whl (298 kB)
    298 kB 819 kB/s
Requirement already satisfied: Jinja2<=2.10.1 in c:\users\apurba\appdata\local\programs\python\python39\lib\site-packages (from Flask->heartrate) (2.11.3)
Collecting itsdangerous<=0.24
  Downloading itsdangerous-1.1.0-py2.py3-none-any.whl (16 kB)
Requirement already satisfied: click<=5.1 in c:\users\apurba\appdata\local\programs\python\python39\lib\site-packages (from Flask->heartrate) (7.1.2)
Requirement already satisfied: MarkupSafe<=0.23 in c:\users\apurba\appdata\local\programs\python\python39\lib\site-packages (from Jinja2->Flask->heartrate) (1.1.1)
Building wheels for collected packages: heartrate
  Building wheel for heartrate (setup.py) ... done
  Created wheel for heartrate: filename=heartrate-0.2.1-py3-none-any.whl size=235972 sha256=0206dc7e362df487222b11904d64d5cf741c2a32b8cd3d9fb3e28a08db9685ed
    Stored in directory: c:\users\apurba\appdata\local\pip\cache\wheels\c6\29\98\8c1d4bbb8598c9e1dec2b4149b44cd61fbb3942379fad3ff3
Successfully built heartrate
Installing collected packages: Werkzeug, itsdangerous, Flask, heartrate
Successfully installed Flask-1.1.2 Werkzeug-1.0.1 heartrate-0.2.1 itsdangerous-1.1.0
```

**Next up, we will see how will it helps us and how do we implement the following module in our code. →**

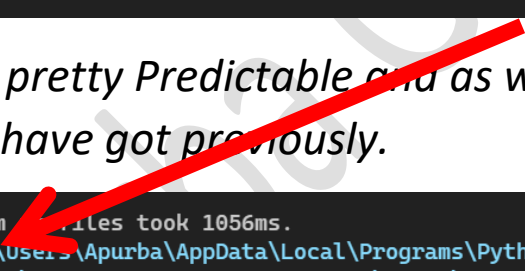


Now, here also we have the same Sample Code defining what a Factorial is and helping us to get the Factorial of a Number using the help of a Function.

**Note: We have used Recursive Function in here in this Code.**

```
def factorial(x: int):  
    if x == 1:  
        return 1  
    else:  
        return (x * factorial(x-1))  
  
if __name__ == "__main__":  
    num = 5  
    print(f"The factorial of {num} is {factorial(num)}")
```

The Output here is pretty Predictable and as we can see here just like this as we have got previously.



```
Loading personal and system files took 1056ms.  
@Apurba → Desktop & 'C:\Users\Apurba\AppData\Local\Programs\Python\Python39\python.exe'  
thonFiles\lib\python\debugpy\launcher' '54681' '--' 'c:\Users\Apurba\Desktop\test.py'  
The factorial of 5 is 120  
@Apurba → Desktop
```

But after implementing **Hearttrate** we get to see as the code as,

```

1
2 import heartrate
3 heartrate.trace(browser=True)
4
5 def factorial(x: int):
6     if x == 1:
7         return 1
8     else:
9         return (x * factorial(x-1))
10
11 if __name__ == "__main__":
12     num = 8
13     print(f"The factorial of {num} is {factorial(num)}")
14
15

```

And the Output is what we have right here,

Note: It would open a tab in your browser in Localhost in your machine in port **localhost:9999**

So, you will see something like this in Your Terminal or run window...

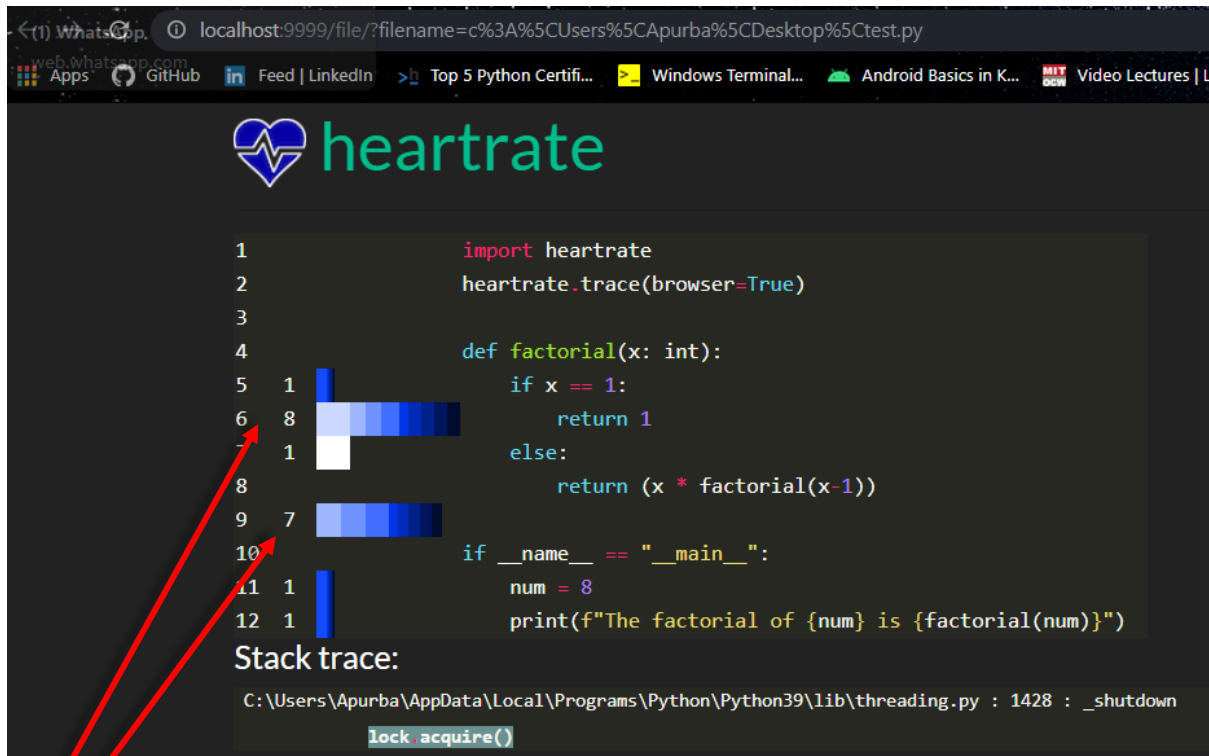
```

@Apurba → Desktop c:; cd 'c:\Users\Apurba\Desktop'; & 'C:\Users\Apurba\AppData\Local\Microsoft\Windows\Common-File\UniversalDataCollection\python.python-2021.4.765268190\pythonFiles\lib\python\debugpy\launcher' '55509' '--
* Serving Flask app "heartrate.core" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
The factorial of 8 is 40320

```

Note: Since it is a Server running at address **localhost:9999**, it won't stop until you stop the program flow.

And you will notice such a window in your browser just like in below with an image of your code and number of times each line is executed...



**Note:** The numbers here shows the **number of times** each line is **executed** and the wheels here shows the visual representation of the same... The **lighter the color** the more **recent** it is in **execution**, the **longer the bars** means the **more number of hits**.

Now let's see what it is like to visualize the execution of a Python program in real-time using heartrate. Let's add `sleep(0.5)` so that the program runs a little bit slower and increase `num` to 20.

```
import heartrate
from time import sleep

heartrate.trace(browser=True)

def factorial(x):
    if x == 1:
        return 1
    else:
        sleep(0.5)
        return (x * factorial(x-1))

if __name__ == "__main__":
    num = 20
    print(f"The factorial of {num} is {factorial(num)}")
```

*So the terminal remains the same, but the browser will show something different, as soon as it pops up look over there and you will see that there is an animation where the wheels are rotation as the the programs flow,*

*Do try it out...*

*That's it guys for the tutorial and Documentation...*

**© iamapurba2003, Stark-Corp**