

Python Fundamentals

Day 66

Today's Agenda

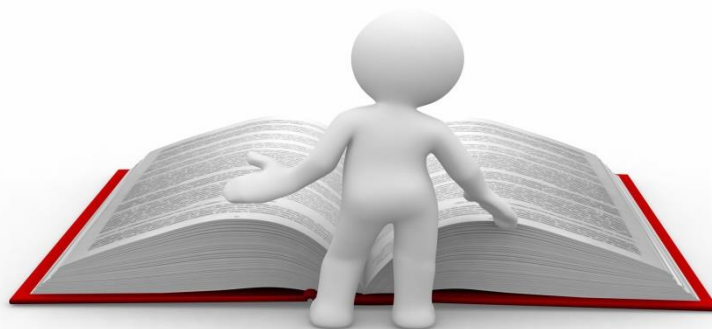
- Loggers
- Levels of logging
- Examples



Loggers

In simple words logger is a piece of code which the user will be attaching in the program. This logger will capture some information from your program and store it in a log file.

What is the information it is going to capture? Why should it be stored in a file? What is the advantage of doing this? Is what we shall see one by one. Let's start with one example that takes list of integers from user, the program will call a function which will take only even elements present in the list add them and return the sum.



```
def sum_even(lst):
    sum=0
    for i in lst:
        if i%2==0:
            sum=sum+i
    return sum

def main():
    lst=list(map(int,input().split()))
    res=sum_even(lst)
    print(res)
main()
```



Output:

```
In [2]: runfile('C:/Users/rooman/OneDrive/Desktop/python/test1.py',
wdir='C:/Users/rooman/OneDrive/Desktop/python')
```

```
5 6 7 8 9 10
24
```

There's nothing new in the above code, but let's trace it once

```
def sum_even(lst):
    print('sum_even() started execution')
    sum=0
    for i in lst:
        if i%2==0:
            sum=sum+i
    print('sum_even() finished execution')
    return sum

def main():
    print('main() started execution')
    lst=list(map(int,input().split()))
    print('input taken from the users')
    print('calling sum_even()')
    res=sum_even(lst)
    print('result of sum_even() collected')
    print(res)
    print('main() finished execution')

main()
```

Output:

```
In [3]: runfile('C:/Users/rooman/OneDrive/Desktop/python/test1.py',  
wdir='C:/Users/rooman/OneDrive/Desktop/python')  
main() started execution  
  
5 6 7 8 9 10  
input taken from the users  
calling sum_even()  
sum_even() started execution  
sum_even() finished execution  
result of sum_even() collected  
24  
main() finished execution
```

We can see the above code has too much of information which is not important for the user or the client. Only if the programmer know the code and it's tracing it's enough. And using print statements everywhere is not a good habit of experienced programmer. To overcome this we have loggers where we can store the tracing of program and refer it whenever needed. Let us see how to achieve it

- Create a logger - let's start with basic one i.e. root logger.
- Connect the logger to a file (log file).
- Tell what information to be collected and stored in log file.



```

import logging

def sum_even(lst):
    logging.info('sum_even() started execution')
    sum=0
    for i in lst:
        if i%2==0:
            sum=sum+i
    logging.info('sum_even() finished execution')
    return sum

def main():
    logging.info('main() started execution')
    lst=list(map(int,input().split()))
    logging.info('input taken from the users')
    logging.info('calling sum_even()')
    res=sum_even(lst)
    logging.info('result of sum_even() collected')
    print(res)
    logging.info('main() finished execution')

main()

```

Output:

```

In [4]: runfile('C:/Users/rooman/OneDrive/Desktop/python/test1.py',
wdir='C:/Users/rooman/OneDrive/Desktop/python')

```

```

5 6 7 8 9 10
24

```

Now we can see no print statements is visible on output screen. Wondering how to access that information? Let us see

The information is stored in a certain order i.e.
<level> : <logger> : <message>

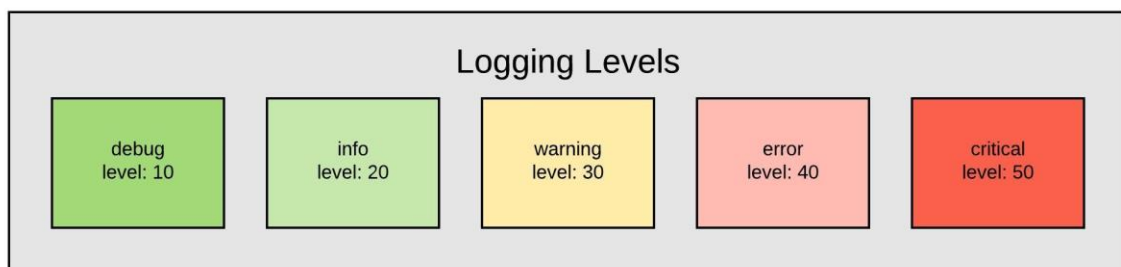
This information will be present in your hard disk where the program is saved.



```
INFO:root:main() started execution
INFO:root:Input taken from user
INFO:root:calling sum_even()
INFO:root:sum_even() started execution
INFO:root:sum_even() finished execution
INFO:root:result of sum_even() collected
INFO:root:main() finished execution
```

Levels of logging

We have several levels in logging which will define what data is being stored in the log file.



Debug: It is used when we want to store debugging related information in log file.

Info: It is used for tracing of program.

Warning: It is used to store warning related information.

Error: It is used whenever exception related information is to be stored.

Critical: It is used to capture information which results in critical failure of application.

Let us start exploring with examples

Example for **debugging**:

```
def add(x,y):  
    return x+y  
  
def sub(x,y):  
    return x-y  
  
def mul(x,y):  
    return x*y  
  
def div(x,y):  
    return x/y  
  
def main():  
    a=10  
    b=5  
    print(f'a={a}')  
    print(f'b={b}')  
    res1=add(a,b)  
    print(f'res1={res1}')  
    res2=sub(a,b)  
    print(f'res2={res2}')  
    res3=mul(a,b)  
    print(f'res3={res3}')  
    res4=div(a,b)  
    print(f'res4={res4}')  
  
main()
```



Output:

```
In [6]: runfile('C:/Users/rooman/OneDrive/Desktop/python/test1.py',  
wdir='C:/Users/rooman/OneDrive/Desktop/python')  
a=10  
b=5  
res1=15  
res2=5  
res3=50  
res4=2.0
```

Activate Windows

Above is the normal code. Let us make changes and add logging features to it.

```

import logging

def add(x,y):
    return x+y

def sub(x,y):
    return x-y

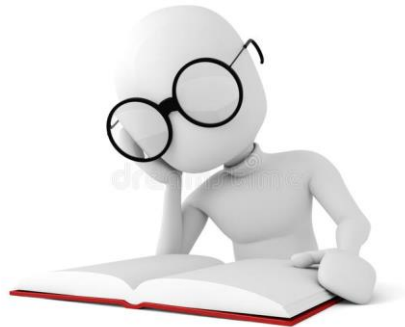
def mul(x,y):
    return x*y

def div(x,y):
    return x/y

def main():
    logging.basicConfig(filename='log.txt',level=logging.DEBUG)
    a=int(input())
    b=int(input())
    logging.debug(f'a={a}')
    logging.debug(f'b={b}')
    res1=add(a,b)
    logging.debug(f'res1={res1}')
    res2=sub(a,b)
    logging.debug(f'res2={res2}')
    res3=mul(a,b)
    logging.debug(f'res3={res3}')
    res4=div(a,b)
    logging.debug(f'res4={res4}')

main()

```



Output:

```
In [7]: runfile('C:/Users/rooman/OneDrive/Desktop/python/test1.py',
wdir='C:/Users/rooman/OneDrive/Desktop/python')
```

10

5

Activate Windows

Definitely all the results will be stored in log file. As shown below

log - Notepad

File Edit Format View Help

```
DEBUG:root:a=10
DEBUG:root:b=20
DEBUG:root:res1=30
DEBUG:root:res2=-10
DEBUG:root:res3=200
DEBUG:root:res4=0.5
```

Example for Warning:

```
def validate(num):
    if len(num)==10:
        print("Valid mobile number")
    else:
        print("Invalid mobile number")

def main():
    num=input("Enter mobile number:")
    validate(num)

main()
```



Output:

```
In [4]: runfile('C:/Users/rooman/OneDrive/Desktop/code/test1.py',
wdir='C:/Users/rooman/OneDrive/Desktop/code')
```

```
Enter mobile number:9988665577
Valid mobile number
```

```
In [5]: runfile('C:/Users/rooman/OneDrive/Desktop/code/test1.py',
wdir='C:/Users/rooman/OneDrive/Desktop/code')
```

```
Enter mobile number:559968709
Invalid mobile number
```

Activate Windows

Let us see how to log the **warning** message into the log file


```

import logging

logging.basicConfig(filename='log.txt',level=logging.WARNING)

def validate(num):
    if len(num)==10:
        print("Valid mobile number")
    else:
        logging.warning("Mobile number validation failed")
        print("Invalid mobile number")

def main():
    num=input("Enter mobile number:")
    validate(num)

main()

```

Output:

```
In [6]: runfile('C:/Users/rooman/OneDrive/Desktop/code/test1.py',
wdir='C:/Users/rooman/OneDrive/Desktop/code')
```


```
Enter mobile number:9988776655
Valid mobile number
```

```
In [7]: runfile('C:/Users/rooman/OneDrive/Desktop/code/test1.py',
wdir='C:/Users/rooman/OneDrive/Desktop/code')
```

```
Enter mobile number:998876544
Invalid mobile number
```

Activate Windows

Let us see if the log file has got updated log message

 log - Notepad

File Edit Format View Help

```

DEBUG:root:a=10
DEBUG:root:b=20
DEBUG:root:res1=30
DEBUG:root:res2=-10
DEBUG:root:res3=200
DEBUG:root:res4=0.5
WARNING:root:Mobile number validation failed

```

Certainly log file has got updated as expected.

Now let us see next example with **error** level

```
def div():
    num=int(input('Enter the numerator: '))
    den=int(input('Enter the denominator: '))
    q=num/den
    print(q)

def main():
    div()

main()
```



Output:

```
In [8]: runfile('C:/Users/rooman/OneDrive/Desktop/code/test1.py',
wdir='C:/Users/rooman/OneDrive/Desktop/code')

Enter the numerator: 100

Enter the denominator: 2
50.0

File "C:/Users/rooman/OneDrive/Desktop/code/test1.py", line 8, in
main
    div()

File "C:/Users/rooman/OneDrive/Desktop/code/test1.py", line 4, in
div
    q=num/den

ZeroDivisionError: division by zero
```

Let us modify according with the concept of loggers

```
import logging

logging.basicConfig(filename='log.txt',level=logging.ERROR)

def div():
    try:
        num=int(input('Enter the numerator: '))
        den=int(input('Enter the denominator: '))
        q=num/den
        print(q)
    except:
        logging.error('Exception Occured')

def main():
    div()

main()
```

Output:

```
In [10]: runfile('C:/Users/rooman/OneDrive/Desktop/code/test1.py',  
wdir='C:/Users/rooman/OneDrive/Desktop/code')
```

```
Enter the numerator: 100
```

```
Enter the denominator: 0
```

Activate Windows

We can see that no error is displayed neither is the respective message. Let us check the log file to get the respective message.

log - Notepad

File Edit Format View Help

```
DEBUG:root:a=10  
DEBUG:root:b=20  
DEBUG:root:res1=30  
DEBUG:root:res2=-10  
DEBUG:root:res3=200  
DEBUG:root:res4=0.5  
WARNING:root:Mobile number validation failed  
ERROR:root:Exception Occured
```

Great!! We can see the message now, but is it conveying entire message? No. Let us see how to get the trace back of the exception occurred

```
import logging  
  
logging.basicConfig(filename='log.txt',level=logging.ERROR)  
  
def div():  
    try:  
        num=int(input('Enter the numerator: '))  
        den=int(input('Enter the denominator: '))  
        q=num/den  
        print(q)  
    except:  
        logging.error('Exception Occured',exc_info=True)  
  
def main():  
    div()  
  
main()
```

Output:

```
In [11]: runfile('C:/Users/rooman/OneDrive/Desktop/code/test1.py',  
wdir='C:/Users/rooman/OneDrive/Desktop/code')
```

```
Enter the numerator: 100
```

```
Enter the denominator: 0
```

Let us check the log file and see what is the message we'll get after including exception info argument

```
log - Notepad  
File Edit Format View Help  
DEBUG:root:a=10  
DEBUG:root:b=20  
DEBUG:root:res1=30  
DEBUG:root:res2=-10  
DEBUG:root:res3=200  
DEBUG:root:res4=0.5  
WARNING:root:Mobile number validation failed  
ERROR:root:Exception Occured  
ERROR:root:Exception Occured  
ERROR:root:Exception Occured  
Traceback (most recent call last):  
  File "test1.py", line 9, in div  
    q=num/den  
ZeroDivisionError: division by zero
```

Great! We have received the trace back of the exception generated. But every single time the log message gets appended making it hard to know which message is appropriate one. Let us see how to change this, and get only the log message of that particular file executing.



```

import logging

logging.basicConfig(filename='log.txt',level=logging.ERROR,filemode='w')

def div():
    try:
        num=int(input('Enter the numerator: '))
        den=int(input('Enter the denominator: '))
        q=num/den
        print(q)
    except:
        logging.error('Exception Occured',exc_info=True)

def main():
    div()

main()

```

Output:

Output will be same as previous ones. Let us directly take a look at log file

```

*log - Notepad
File Edit Format View Help
ERROR:root:Exception Occured
Traceback (most recent call last):
  File "test1.py", line 9, in div
    q=num/den
ZeroDivisionError: division by zero

```

Awesome! We got it as expected. Here by default the filemode will be append(a), but we have changed it to write mode(w) and executed.

