11th

9AY PYTHON PROJECT

LOGGING & DEBUGGING

part 2 → Test 2. PY

* Project

Python Project-123

- main.py
- test.log
- test.py
- test1.log
- test2.py
- External Libraries

test2.py

```
import logging
Logging.basicConfig("test2.log")
```

Here, we have created a new file called name test2.py

test2.py

* import logging

inside test2.py

import logging.

Now, what, I can do is may be I can create a new file, a new log level & multiple things I can create.

* ~~logging.ba~~

* Logging.basicConfig("test2.log")

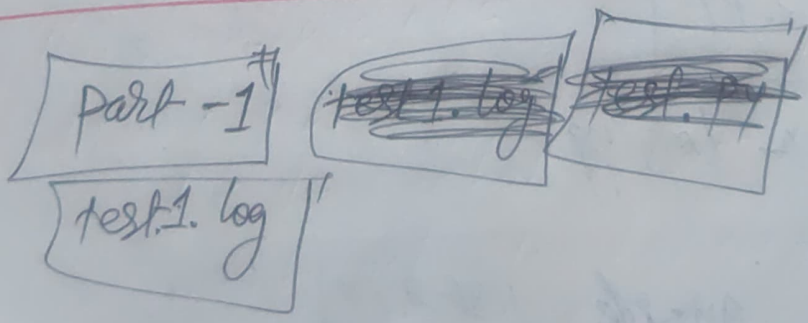
we can try to call ~~logg~~

logging.basicConfig {(basicConfig)} so that, I ~~can~~ can create a ~~function~~ file.

Now here ("test2.log") test2.log file I have created.

---

* Last



*

* Earlier test1.log inside there, we basically trying to log type of logs.

whether its a INFO
warning
Error etc.

everything was written inside test1.log earlier. All the message, I would like to kept, I was able to do that., after 1 year also, if I would like to access those INFO, warning, Error etc. ~~I can do that.~~ If I want to utilized, yes I can do that.

---

* Test2.py :

Here, what I was like to do is

Logging.basicConfig("test2.log")

unexpected argument

Logging

May be I can store time stand, whenever we try to do any kind of activities over a server admin will get to know that.

at what time which user has perform what.

How, when we log into a server, we used to get all the information.

Example:

In neuron we just has introduce a new chatboat system, from where you guys get a support from our team, now in that one.

In that support system what we are trying to do, how we will get to know that which user has asked what kind of question, who has reply from our team etc, what rating that student got an everything we are able to look into it.

After a year also, I am able to see who has done what, what type of conversation, what activities happened entire portal, I am able to see it.

* what they have done is while doing coding, they have logged each & every steps along with time stand as well as with the users with the multiple information.

~~Here~~

* logging.basicConfig("test2.log")

Here, my basic requirement is to show case log along with the time stand.

So, How I will be able to logg time stand let's understand :-

pythonproject123

main.py
test.log
test.py
test1.log
test2.log
test2.py

test2.py

* import logging
* logging.basicConfig(filename="test2.log", level=logging.DEBUG, format='%(asctime)s %(message)s')

Logging.info("this is my log with timestamp")

* logging.basicConfig(filename="test2.log", level=logging.DEBUG, format='%(levelname)s %(name)s %(asctime)s %(message)s')

Logging.info("this is my log with timestamp")

let's understand :-

* logging.basicConfig(filename="test2.log", level=logging.DEBUG, format='%(asctime)s %(message)s')

So let's breakdown code & understand.

* filename="test2.log" ~~test2.log~~ → This will create a log file.

level = logging.DEBUG —

I would like to define level = logging.DEBUG this is the level of logging, the very first hierarchy 10. The score for debugging is equal to 10.

So again after that INFO come to a picture which is score having 20. So, which is like having high priority, as compare to previous one.

* format='%(asctime)s %(message)s')

Then, I can define format, inside format I can define may be time stamp, this is how it is suppose to log any information at any point of a time.

So, may be, I can try to define format that ok along with the messageses ~~or~~ whatever message that we would like to pass along with the message just do one thing try to store a system time as well.

At what time this Error has happened or at what time info has been generate I would like to store that as well.

In case of that I can define these entire thing inside a format.

So what I can do is, I can try to give like % symbol & then try to Call '%(asctime)s' in case of pycharm, it auto Completes all of those things.

Asctime store as a string (str) & then try to store %(message)s.

This is something which, I can try to define %asctime & %message.

It will try to do store not just a messageses, because I am ~~just~~ just trying to give a ~~pla~~ place holder.

Because % percentile always

try to take a place holder.

~~I am not just store a place holder~~

I ~~am~~ am just not trying to store a message, but I am trying to store a system time as well at same point of a time, I am trying to store a system time as ~~well~~ well.

* Logging.info("this is my log with timestamp")

let's suppose,

I am going to call logging.info inside these ("this is my " " . timestamp")

after Execute it Debug 'test2' ← right click

Go and check to the

test2.log

2022-06-29 21:44:26,907 this is my log with ~~sta~~ timestamp.

Now, here if we come & check, I will be able to see that time stamp is mention 2022-06-29 21:44:26 for me its a 2022 Now its 2023 year date time changed.

for me sir did this things in 2022 year.

2022-06-29- timestamp mention.

(this is my log with timestamp) - this is a message which, I am able to see.

So, how from where these able to generate these

→ timestamp & this message ("this is my " " timestamp)

It is able to generate, because, I have define the formal. formal = "%(asctime)s %(message)s")

I have simply said, whenever you try to log something just do one particular operation, try to store a time as well as try to store message.

%asctime

2022-06-29

%message

(this is my log with timestamp)

Now, here outCome these outCome & earlier outCome a little bit difference.

here in this, I have log. some info

logging.info("this is my ... ... time stamp")

But when last time time outCome it shows me different like

test1.log

INFO : root : "

INFO : root : "

"

"

warning : root : "..."

INFO : root :

"

ERROR : root : "

It was trying to show me INFO, 'warning', 'ERROR' etc.

But this time

test2.log

2022-06-29 21:44:26,907 this is my log with timestamp.

But this time go & check, we will not able to find out that.

INFO, warning, error these ~~type~~ type ~~this~~ things. I will not give that one.

why because,

we have just given inside our format

*format = '%(asctime)s %(message)s')

that whenever you to going to log something always try to keep these format of a log.

That is only region, it is not trying to store name of the logg.

let's suppose,
I want to see this thing ~~Same~~
outCome also same as earlier one.
~~For~~ that.

* Logging.basicConfig(filename = "test2.log",
level = logging.DEBUG, format = '%(levelname)s
%(name)s :%(asctime)s %(message)s')

Logging.info("this is my log with timestamp")

right click & Exelute it

test2.log

Come back ~~see~~ if we see in side

test2.log

2022:06:29 21:44:26. this is my log "
" "
" "
" "
" "
" "

INFO root 2022-06-29-21:47:24,573
this is my log with timestamp

Now; we ~~or~~ are able to see same way
outCome like earlier it was.

(levelname)s (name)s (asctime)s (message)s

INFO root 2022-06-29 (this is my timestamp)

So, this ~~the~~ way I can write code & as a outcome, we can see like earlier.. same format, we just need to make change in format section. & as a outcome we can these.

These way I can perform a operation in a best way.

This is what format does.

Python project 123

- main.py
- test.log
- test.py
- test1.log
- test2.log
- test2.py
- test3.py

test3.py

```
import logging

def devide(a, b):
    return a/b

print((devide(3, 4)))
```

* `print((devide(3, 0)))`

Error : zero division

Let's understand :-

Let's understand logg by ~~writing~~ writing couple of code.

Create new file name test3.py

a fresh file, I have just created over here.

test3.py

```
* import logging

def devide(a, b):
    return a/b

~~devide(3,4)~~
print((devide(3,4)))

print((devide(3,0)))
```

So let's import logging. 1st.

Here, I am going to create a very basic function, which is nothing but which will devide 2 integer, that the only function I am going to write.

whatever happens between that function I have to handle it.

So,

Let's suppose, I have written devide function. try to pass (a, b) ~~if~~ then I am trying to return a/b.

Now,

If I am going to call these function I have to pass some information.

* print ((devide (3, 4)))

Now, If I am going to print these entire thing.

yes, 3/4 ~~is~~ 3 by 4 it is able to

devide & it is able to return.

what, if I am going to pass

* print((devide(3,0)))

If I will pass (3,0))) what will happened these case.

Mathematically 3,0 or 3/0 is infinity.

~~In~~ Infinity is not a symbol which is available in a core python.

It's available in Numpy. In numpy its a valid operation.

Because, there is a symbol called as Infinity 'Inf' which exist.

But in core python it is not available.

* def devide(a, b):
    return a/b

print((devide(3,0)))

Error:-

zeroDivisionError:- division by zero.

If I am going to execute this code, it is going to give me an Error. (Division by zero.)

If day after tomorrow if someone would like to check, what kind of mistake, what kind of number or parameter, person has passed.

How person will check, dynamically I am asking person to ~~int~~ insert ~~a/b~~ a & b.

I can take a input, we are aware about the input method, which I will be

take a input, there are 100 of user which will use my code.

Same code if [redacted]

I would like to know how many person has passed value of (b) as a zero, if what is the value of (a). person has passed. How I will be see all short of a. information.

If Because, I am trying to print something in a console that will be gone.

If I am going to hold this program somewhere in cloud platform a server, I will not be able to understand these things.

Because the server going to show case just a current use of it or current execution of it.

server will not going to show case pervious of it.

So, how will I be [redacted] able to persistece this information.

```
Traceback (most recent call last):
  File "", line 1, in <module>
ZeroDivisionError: division by zero
```

These entire information inside some of the log file.

let's try to full fill these objective.

Q: objective wise what we are looking for, what we have to achieve.

I am not looking for these message here.

I am looking for these message

```
C:\anaconda\envs\pythonproject-123\...
Traceback (most recent call last):
  "
  "
ZeroDivisionError: division by zero
```

I am looking for these message or Error message somewhere into a file system with time stamp & everything.

python project123

- main.py
- test1.log
- test.py
- test1.log
- test2.log
- test2.py
- test3.py
- External Libraries

test3.py

```
* import logging
* Logging.basicConfig(filename="test4.log", level=
    logging.INFO, format='%(levelname)s
    %(asctime)s %(name)s %(message)s')

* def devide(a, b):
      logging.info("the number entered by user
      is %s and %s", a, b)
      return a/b

* print((devide(3,6)))
```

Let's understand :-

* import logging

we have import for logging. module.

*

~~As we~~

As we know first of all, we are suppose to create, a physical file, for that what is command.

Logging.basicConfig required unless & until, if we have not created a file, where we are going to store all those information, there is no place to store.

So, ok

filename = "test4.log"

This filename, I have define, then what I will do is.

Level = logging.INFO

level, I have define Logging.INFO.

format = '%(levelname)s %(asctime)s
%(name)s %(message)s')

Now, I would like to ~~store~~ message which I would like to store, I would like to store even a timestamp & the system, I would like to store log level as well.

For that what we will do 'Format' inside these, I can put, ..

level name, System time, root or may be Sub users then messages, I would like to write into it.

These is a basic Configuration which, I have already design.

```
* def devide(a, b):
      Logging.info("the number entered by
                    user is %s and %s", a, b)
      return a/b
print((devide(3, 6)))
```

Now, let's suppose, there is situation, let's suppose, I would like to store, what input user is giving.

user will going to input (a, b).

I would like to store that information that will come under which kind of logg.

INFO, warning, debug, critical or ERROR kind of logg.

Can I say that kind be ~~Simi~~ simple information. It's a basic info.

* logging.info("the number entered by user is %s and %s"; a, b)

Same program, here we are modifying. I am going to say, logging.info(" ..... ") what kind of information (info) I would like to store.

May be, I can store.

(the number entered by user is %s and %s, a, b)

%s	%s
a	b

I am trying store these things as a string.

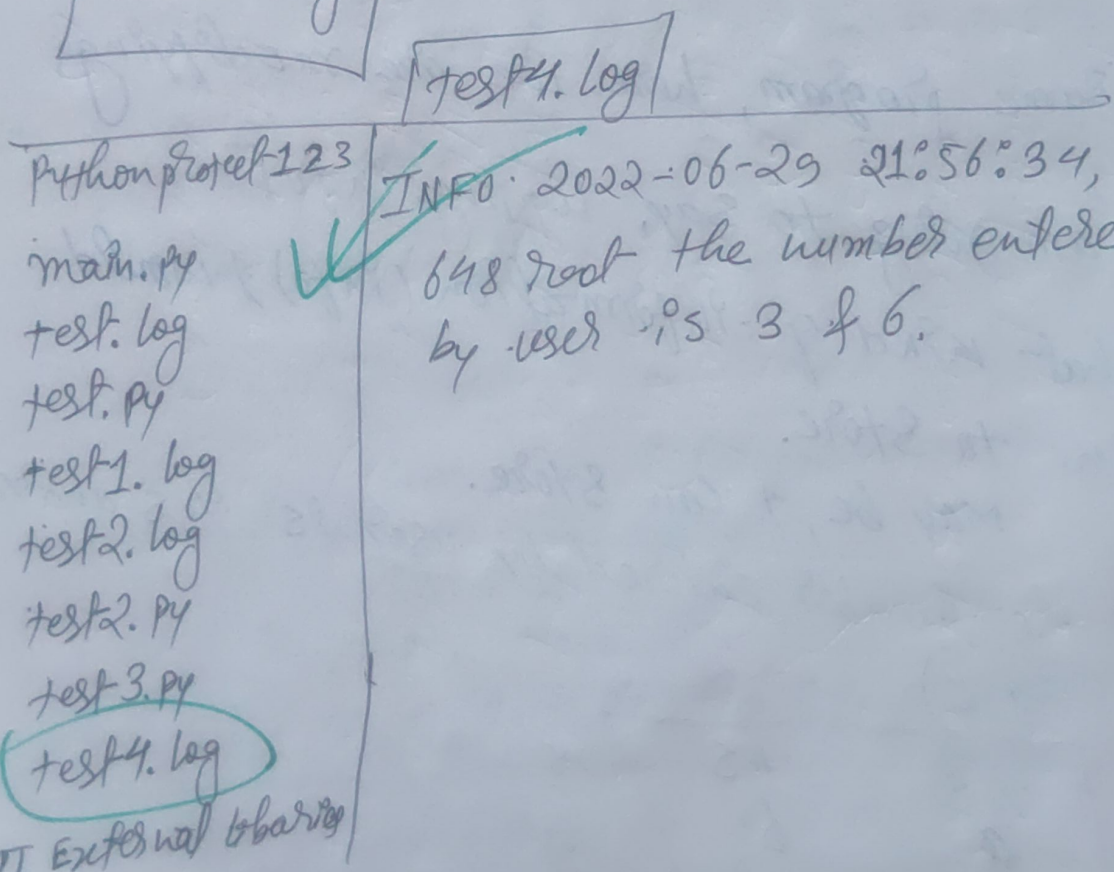
Now,

print((devide(3, 6)))

right click & Run test

once we execute this code. & go to the

test4.log



Now, If I will go & check test4.log, I can see that

INFO | 2022-06-29 21:56:34,648 Read the number entered by user .Is 3 and 6.

we are able to logg the information.

So, here, I had a requirement that logg whatever data which has entered by user, we are able to do it.
we are able to logg a, & b. data value, which user will enter at the time of function call, yes, it will logg automatically.

pythonProject123

- main.py
- test.log
- test.py
- test1.log
- test2.log
- test2.py
- test3.py
- test4.log

III External Libraries

test-3.py

```
import logging
Logging.basicConfig(filename="test4.log", level=logging.INFO,
    format="%(levelname)s %(asctime)s %(name)s
    %(message)s")
def devide(a,b):
    Logging.info("the number entered by user is %s and
                  %s", a, b)
    try:
        Logging.info("we are into function")
        div = a/b
        Logging.info("we have completed a division operation")
        return div
    except Exception as e:
        Logging.exception(e)

(devide(3,6))
```

let's understand :-

Now, There is a possibility, I know in a run time user can pass a '0' zero. as a ~~the~~ denominator.

As a denominator user can pass a value of 'b'. user can pass a zero (0).

To handle those possibility what should, I do.

Can, I say that we can do exception handling. Exception Handling we have discussed,

```
try :
    Logging.info("~~we~~ we are into function")
    div = a/b
    Logging.info("we have completed a division
                  operation")
```

```
try:
    logging.info("we are into function")
    div = a/b
    logging.info("we have completed a division operation")
```

↳ Can try to make 'try' block, ↳ we can try to create a variable may be a

* `div = a/b.`

Now, these a/b ; whatever a/b going to give it to me it will be store ~~in~~ inside `div` variable.

Then, we can logg some

~~logging.info("we are about to do~~

* `logging.info("we are into function")`

Now,

Again, we can try to logg.

* `logging.info("we have completed a division operation")`

* div = a/b

If it will be able to ~~devi~~ devide then only it will come over here.

* logging.info("we have completed a division operation")

* div = a/b

If this is going to fail, Can we Say that it will come to the inside ~~Exception~~ Except statement.

This is what we already discussed in our exception Handling classes.

* div = a/b

If this is going fail then, it will not going to execute this one

* logging.info("we have completed a division operation")

It will come into the 'Except' statement. where we are going to handle a exception.

* div = a/b
  logging.info("we have completed " " ")
✓ return div.

If this is successful (div = a/b)
  logging.info("we have completed " ")

Then in that case, please try to return me 'return div'

division(div)

* except Exception as e:
  logging.exception(e)

* (devide(3, 6))

we have seen in our exception Handling Classes.

If this will fail * div = a/b

It will not even execute this one

* logging.info("we have completed a division operation")

Then, it will come inside the Except statement, where we are going to Handle a exception.

* except exception as e: e (Alias)

~~Exception~~

So, except then exception as 'e' alias that we have created & then we can try to logg basically ~~logging~~

logging.exception(e)

what is my 'e' 'e' is nothing but, whatever exception, it is going to raise, it will be able to logg, each & every thing.

Right click & execute it
& now go to check

test4.log

INFO: 2022-06-29 21:56:34,648 root the number entered by user is 3 & 6.

INFO: " 22:03:52:513 root " "
" "

INFO:2022-06-29 22:03:52,513 root we are into function.

INFO: 2022-06-29 22:03:52;513 root we have completed a division operation.

---

* we are into function outcome because of these line code.

* logging.info("we are into function")

* we have completed a division operation (output or result come)

* logging.info("we have completed " " " operation")

Now, after that, 'return' simple

* (devide (3,6)) it will give you the return. whatever we are looking for.

pythonproject123

main.py
test.log
test.py
test1.log
test2.log
test2.py
test3.py
test4.log

import logging
logging.basicConfig(filename ... '' '' ...

''

''

''

''

Same code

''

''

(devide(3,0))

only last line, I have made
Change instead of
devide(3,6)), I have given
(devide(3,0))

let's understand :-

Here, In this code everything is ok, only change, we made that is in the last.

Instead of (devide(3,6)), I have made it (devide(3,0)). ✗

once, Change (3,0) execute it

go to check

test4.log

INFO: 2022-06-29 21:56:34 by root the number entered by user is 3 & 6.

INFO 22-06-29 22:04:42 root: we are into function.

ERROR 2022-06-29 22:04:42 root division by zero

Traceback (most recent call last):

File "E:/python ...",

ZeroDivisionError: division by zero.

Let's understand :-

* ~~logging exception~~

The number entered by user is 3 & 0.

My program is into the function

logging.info("we are into function")

My program is trying to perform this operation

```
div = a/b
```

But it will fail.

So, it will come direct to these one code.

```
except Exception as e:
    logging.exception(e)
```

& it will log to logging.it.

```
ERROR · 2022-06-29 · 22:04:42,832 · Root
                                  division by zero
Traceback (most recent call last):
  File...
    div 1=
ZeroDivisionERROR: division by zero
```

This has the ERROR, which I have received & it has logged it successfully, inside a particular file.

Can we say we Can achieve Same thing with the help of 'print' statement what I am trying to do is.

This way, If we are going to write program.

we can see that, If I used 'print' statement then my out come reflect in just below down where, I have written code.

But if I will write code one side & instead of reflecting down below, the out come go to next page with others out come it will go sequence wise.

If I will used

```
logging.exception(e)
```

let's suppose, program is very very to complex.

I am not able to understand that control of a program is going from where to where.

If I will write code it just down below reflect. so what I will do is, If I will logg all the program in the next page. we can see my code after years also.

this way, I can maintain code or everything thing line by line.

I can understand that, what is happening whether it was Error, warning. or exception everything will be able to understand. neat & clean way.

* Do not use by mistake also 'print' statemant over here.

Instead of that use this line of code.

```
except Exception as e:
    logging.exception(e)
```

pythonProject123

- main.py
- test.log
- test.py
- test1.log
- test2.log
- test2.py
- test3.py
- test4.log
- test5.py

External Libraries

test5.py

```
import logging
Logging.basicConfig(filename = "test5.log", level = logging.ERROR,
    format = '%(levelname)s " ' ' ' '

def devide(a,b):
    "
    "
    "
    except Exception as e:
        logging.exception(e)

(devide(3, 7))
```

Let's understand :- * Error

Now, I am going to create a new file test5.py

* import logging
* logging.basicConfig(filename = "test5.log",
  level = logging.ERROR, ...

Here, logging of what ERROR. I have mention.

Save it execute it & now this time do not create new folder just clean test4.log & see Can see the outcome here.

~~Go back~~

1st clean test4.log & go back & test5.py Save it. right click execute it.

~~Onte~~ After execution,

If ~~Co~~ we come & check

at test4.log

There is no outcome, we can see:

There is nothing..

pythonproject 123

main.py
test.log
test.py
test 1.log
test 2.log
test 2.py
test 3.py
test 4.log
test 5.py
III External libraries

test5.py

```
import logging
logging.basicConfig(filename="test5.log", level=
  logging.DEBUG, format="%(levelname)s " "
    " , ' .
def devide(a,b):
      "
      ;
      :
      .
*(devide(3,7))
```

Let's understand : **DEBUG**

Now, here let's set level logging as a DEBUG

Here, I have set as a DEBUG

Save it & execute it. test5.py

Now, come to the test4.log & see the outcome. we can see some information.

INFO: 2022-06-29 327 root the number entered by ,, ,, ,,

INFO: 2022- ,, ,,

INFO : =

INFO : ,, ,, ,, ,,

INFO : ,,

python project 123

- main.py
- ~~test~~
- test.log
- test.py
- test1.log
- test2.log
- test2.py
- test3.py
- test4.py
- test5.py

III External libraries

test5.py

```
import logging
logging.basicConfig(filename="test5.log" level=
    logging.CRITICAL, format= '%
    '
    '
    "
```

* (devide(3, 0)) ✓

Let's understand :- * CRITICAL *

Let suppose, if I am going to set may be CRITICAL

* import logging
* logging.basicConf(filename = "test5.log",
level = logging.CRITICAL, format = " "
" " " " " ".
"
"
"
(devide(3,0))

I will use here CRITICAL logging & I will give here devide(3,0).
we know that, if we put devide(3,0).. It will put 'ERROR' for sure.
last time it was showing ERROR in my last ~~thos~~ logg file.

Once we right click save it & execute it.

go & check test4.log

```
INFO: 2022-06-29 . 22:23:09, the number entered by user is 3 & 7.
''
''
''

INFO: 2022-06-29 . . . '' '' ''
```

Can we see any ERROR over here, as a outcome.

'No' right, we know that if we use devide(3,0). we know that it will generate an ERROR.

But, we can't see any ERROR in my test4.log.

But we know that devide(3,0) will give an ERROR outcome.

To check that I will do some modification or change in code.

test3.py

```
import logging
logging.basicConfig(....
                    ,
                    ,
                    ,
                    ,
except Exception as e:
    logging.exception(e)
    print(e)

*(devide(3,0))
```

~~If we add~~

If we add 'print statement'

print(e) just to check & cross verification; if we put print(e) later remove it,

But now to cross verification ~~che~~ keep it & right click execute it.

Once execute we can see out come just down below code.

C:\Users\ana\Conda\envs\

division by zero

process finished with exit code 0.

yes, we can see its printing

division by zero ERROR ← trying to print.

But,

If we go to the

~~test5.py~~

test4.py go & check we can't see any Error in my test4.py inside

& can see

INFO: 2022-11-29 12:11:11

INFO:"

Here, I have done CRITICAL &

devide(3,0)

we know that, If we use (devide(3,0) I will going to give me an Error. But still I am not getting an Error inside test4.log

why,

~~Because,~~

Earlier when we used DEBUG & devide(3,0) it was showing me an ERROR inside ~~test~~ test. or logg file.

But, here when we are trying to do a CRITICAL & devide(3,0) it's not showing me an ERROR inside my test4.log or logg file.

~~why~~ ?

why ?

Because, that is the region sir has given us a 'Numbers'.

'NUMBERS'

Let's rewrite a numbers :-

(1) DEBUG — 10

(2) INFO — 20

(3) WARNING — 30

(4) ERROR — 40

(5) CRITICAL — 50

(1) DEBUG :-

So, when we are going to set

[redacted] level = logging.DEBUG. It [redacted] simply

means that, it will capture

10 and all 20, 30, 40, 50.

(2) INFO :-

when we set level = logging.INFO,

then, it will not capture 'DUBUG',

Even though if we have handle

logging.DEBUG & going to send some message but still it will not able to show those message, because we have set a level over here. (20, 30, 40 & 50)

```
level = logging.INFO
```

(3) WARNING :-

If we are going to set 'warning' it will try to handle warning & below (30, 40 & 50)

(4) Error :-

If we are going to set ERROR. Then it will handle ERROR & below rest. (40, & 50)

(5) CRITICAL :-

If we are going to set 'CRITICAL' it will just handle 'critical'. It will not handle anything else, it will not logg anything else. inside a system. #

pythonProject123

main.py
test.log
test.py
test1.log
test2.log
test2.py
test3.py
test4.py
test5.py
External Libraries

test5.py

```
import logging
logging.basicConfig(filename="test5.log",
                    level=logging.WARNING,)

try:
    ...
    "
    !


except Exception as e:
    logging.exception(e)
    print(e)

(Divide(3,0))
```

let's understand:

* If we are going to set 'CRITICAL', will I able to logg 'ERROR'.
  Answer is 'NO'.

let's suppose,
I am going to set 'WARNING'
my logg level is 'WARNING'
will I able to see 'ERROR' now.
Yes

This one
* logging.exception(e)
whenever, we have a 'ERROR'
will I be able to see 'info'.

* logging.info("the number entered by "
"... %s & %s", a, b)

I will not be able to see 'INFO', because warning having number = 30 &
'info' has having 20.

'warning' is more serious as compare to 'INFO'.

If I have setting a [redacted] log level = warning.

It will show me 'warning', Error & critical.

But, it will not going to show me 'info' & [redacted] 'DEBUG'.

If we ~~Ex~~ right click & Execute it. test3.py . Run test3

Go to the test5.py . or test4 &

If we come & see over here as a outcome.

test5.py

```
Error 2022-06-29 22:17:55,716 root division
                                by zero
Traceback (most recent call last):
File "E:/pythoproject123/ . . .
    div = a/b
ZeroDivisionError: division by zero
```

Once, we executed & as a outcome we can see over here.

[redacted] Are we able to see 'ERROR' over here.

But are we able to see 'INFO' whatever we have logged as a 'INFO' are we able to see it. 'NO'.

Because, level logging. WARNING

This is a meaning of 'level' in case of [redacted] logged.

That is the meaning of levels in case of logg. That is the meaning of [redacted] 'info level', 'critical level', 'ERROR level' 'Debug level'.

pythonProject123

main.py
test.log
test.py
test1.log
test2.log
test2.py
test3.py
test4.py
test5.py

External libraries

test5.py

```
import logging
logging.basicConfig(filename="test5.log",
                    level=logging.DEBUG, format="%(asctime)s %(levelname)s %(message)s")
try:
    logging.info("i am trying to read a file")
    with open("vijay.txt", "r"):
        logging.info("successfully it has read the file")
except Exception as e:
    logging.critical("this is a situation for us")
    logging.error(e)
```

let's understand :- *DEBUG*

Here,

Earlier sir has ~~wri~~ writing code inside test4.py, Now ~~He~~ sir was started writing code in new page. test5.py

But, I was writing code in test5.py I will not going to change, I will just remove all things inside test5.py & I will write Fresh code.

So, here, I will going ~~use~~ call

* import logging,
* logging.basicConfig(filename="test5.log", level=logging.~~DEB~~ DEBUG, format = "%d"' ' " " " " " .

This is my logging ~~confi~~ configuration.

let's write a Fresh code over here.

what we are going to do is, may be I will try to read out ~~some~~ some file, I will try to read ~~a~~ out some files in this particular place.

we know how to read a file.

what is the command for that.

file.open is the command.

TO open up any kind of file.

* try :
with open("vijay.txt", "r"):

So, may be I can try to use "Exception Handling" over here to logg different different kind of a information..

try:

Inside try I am going to write something, with open. So open what "vijay.txt" with my name, I would like to open up & I have given "r" read mode.

Now, this operation fail or going to work.

This operation going to fail, because there is nothing called as a "vijay.txt" & I just open up a file to read mode. In read mode it will not able to read because there is no such kind of a file.

If I have given right mode that case if file has been exist, it will create one. This is what we discussed in our previous classes.

So, here we have just trying to open up this file into a read mode.

I have given "r" over here.

So, it is going to fail.

That is the region, I am using 'try' over here.

~~* logging.info("successfully it has read the file")~~

```
* try:
    logging.info("I am trying to read a file")
    with open("vijay.txt", "r"):
        logging.info("successfully it has read
                      the file")
    except Exception as e:
        ~~logging.error(e)~~
        logging.critical("this is a situation
                          for us")
        logging.error(e)
```

---

let's understand this code.

* logging.info("I am trying ....")

First of all it will try to logg, fine I am trying to read a file.

what is my Level logging.

my Level = logging.DEBUG

Can we say that, bydefault it will be able to logg 'info'

Yes, it is able to logg, some information inside.

logging.info("I am trying to read a file")

Now, I am trying to write with open

* with open("vijay.txt", "r"):

I trying to write "r" read mode.

* logging.info("successfully it has read the file").

logging.info Successfully it read the file.

If it is not able to read a file. Then it will come below code.

* except Exception as e:
    logging.error(e)

It will come to this line code & it will logged this error.

* except Exception as e:
    logging.critical("this is a situation for us")
    logging.error(e)

Here, some short of a message, I am trying to logged as a 'critical'.

'I am trying to logged some short of info, that is again completely fine for me.

Now right click inside test5.py execute it.

Come to the see at

tests.log

INFO : 2022-06-29 22:23:41 root
: I am trying to read a file.

CRITICAL : 2022-06-29 : root - this is a
situation for us.

ERROR : 2022-06-29 : 22:23:43 . root
[Error 2] No such file or
directory : 'vijay.txt'.

As a outcome, we can see over here.

INFO : It has logged.

Critical : It has logged, we can see the
message.

&

Error : whatever 'Error' was occured, so
'yes' it is able to logged that
particular 'Error' as well.

It is able to logged successfully.

* Critical, Error just depend upon me, where, I have to use.

Critical, INFO, ERROR etc all depend upon me where, I have to use it.
In which case, I have to call.

pythonproject123

main.py
test.log
test.py
test1.log
test2.log
test2.py
test3.py
test4.log
test5.log
test5.py

III External libraries

test5.py

```
import logging
logging.basicConfig(filename="test5.log", level=logging.CRITICAL,
                    format= " ' ' ' . . . .
try:
    logging.info("i am trying to read a file")
    with open("vijay.txt", "r"):
        "
        "
        "
        "
    logging.critical("this is a situation for us")
    logging.error(e)
```

Let's understand :- 'CRITICAL'

whether it is suppose to logged something or not suppose to logged something that will be decided by the logged level that we are going to create over here.

This time, I will make change same code, Just I will make change to this line code.

* Earlier last code 'DEBUG' replace it with 'CRITICAL'.

My code written inside test5.py.

once make change right ~~clk~~ click & execute it.

Now go to the test5.log.

before execute clear all the data from inside test5.log.

Now go back & ~~test5.log~~ test5.py execute code right click & Come back & see inside test5.log

~~test.py~~

test.log

we can see over here outcome as a

CRITICAL 2022-06-29 22:24:50,089
this is a situation for us.

Can we see, I am able to see only a CRITICAL ~~as~~ part outcome, I am able to logged.

* In case of exception; logging.exception

last time, I have used its again ~~ex~~ equivalent to the Error.
It has a same degree, degree of freedom. at any point of time.

* Here logging.Error(e), I have done.

pythonproject123

main.py
test.log
test.py
"
"
"
"
"
test5.log
test5.py
External libraries

test5.py

```
import logging
logging.basicConfig(filename="test5.log", level=logging.INFO,
                    format="...")

try:
    logging.info("i am trying ...")
    with open("vijay.txt", "r"):
        ...
        "
        "
        "
    logging.critical("...")
except Exception as e:
    logging.error(e)
```

let's understand :- 'INFO'

let's suppose,

~~level = logging.NI~~

level = logging.INFO

below code

* logging.error(e)

if I will use level = logging.INFO &

down below logging.error(e)

if we use logging.error(e) as a outcome

we can see down below. right click & execute

if & go to check . test5.log

CRITICAL 2022-06-29 . 22:24:50,089 . root
this is a situation for us

INFO 2022-06-29 . 22:25:30 . "I am trying "
"

CRITICAL 2022-06-29

ERROR 2022-06-29 . 22:25:30, [Errno 2] - No such
file or directory: 'vijay.~~txt~~ txt',

There is a small difference, we can see Now here, it is able to logged ERROR.

ERROR: 2022-06-29 22:25:30,494 root

~~[ERROR 2]~~ [Errno 2] No such file or directory: 'vijay.txt'

No such file or directory: this is what, it is done, when I am trying to call ~~logging~~ logging.ERROR.

pythonproject123	test5.py
main.py	import logging
	logging.basicConfig(filename = "test5.log", level= logging.INFO, format=
	" ' ' '
	try:
test5.py	logging.info("I am ' ' '
test5.log	"
	"
	;
	logging.error(e)
*	logging.exception(e)

* INFO *

Let's understand :

Let's suppose, if I am going to call. `logging.exception()` inside `test5.py`

one line added. Now right-click & execute it

Go & check `test5.log`

```
CRITICAL:2022-06-29 22:24:50,089:root:
1:1
ERROR:2022-
INFO:
CRITICAL:2022
ERROR
ERROR
Traceback (most
File "E:/
with
FileNotFoundError:" ....
```

This one line * `logging.exception`

It will going to what, it will going to logg whole exception message for me. not just a main message.
It will execute all messages.

If I would like to see whole exception ERROR ~~through~~ thrown total, ~~then~~ ERROR, then, we can use `logging.exception()`

* Later on we will learn, how to store entire things into DB (database).
with respect to 4 database

(1) SQL
(2) SQL Lite
(3) Mongo DB
(4) Cassandra