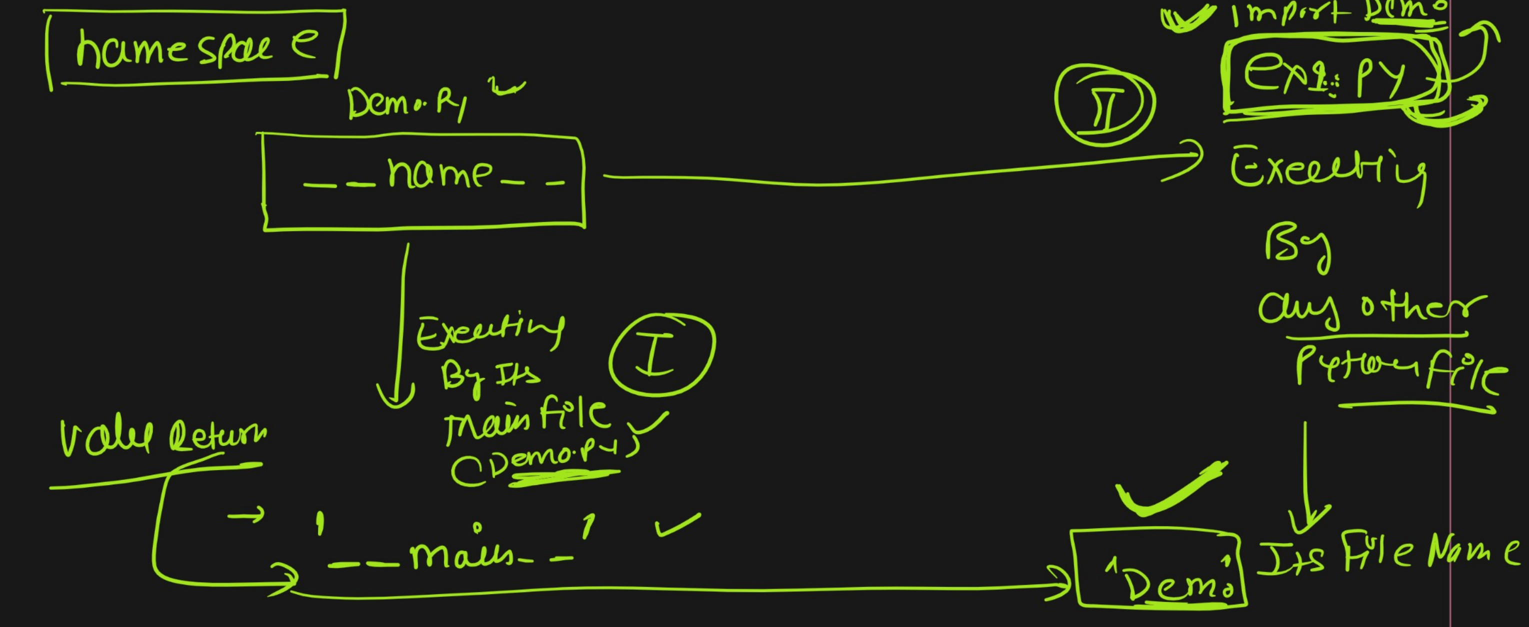
KPMU

Day-5

11-08-2023



namespace

Demo.Py

__name__

Executing By Its main file (Demo.Py) ✓

I

Value Return

→ '__main__' ✓

II

✓ from Demo import *

✓ import Demo

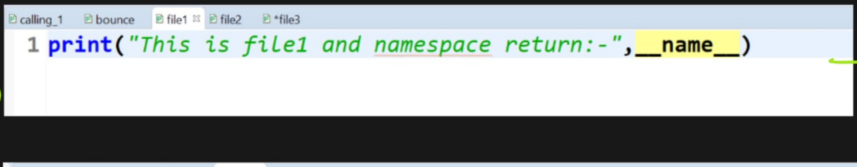
ex1.py

Executing By any other Python file

↓ Its File Name

'Demo' ✓

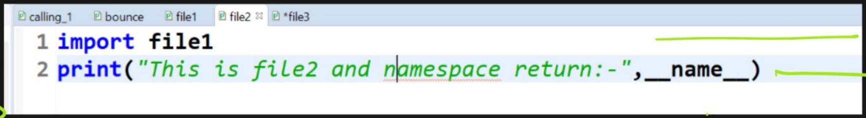
file1



```python
print("This is file1 and namespace return:-",__name__)
```

→ file1

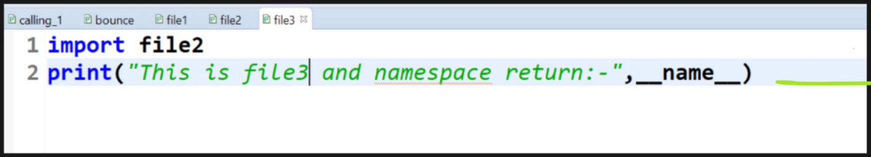
file2



```python
import file1
print("This is file2 and namespace return:-",__name__)
```

→ file2

file3 ✓



```python
import file2
print("This is file3 and namespace return:-",__name__)
```

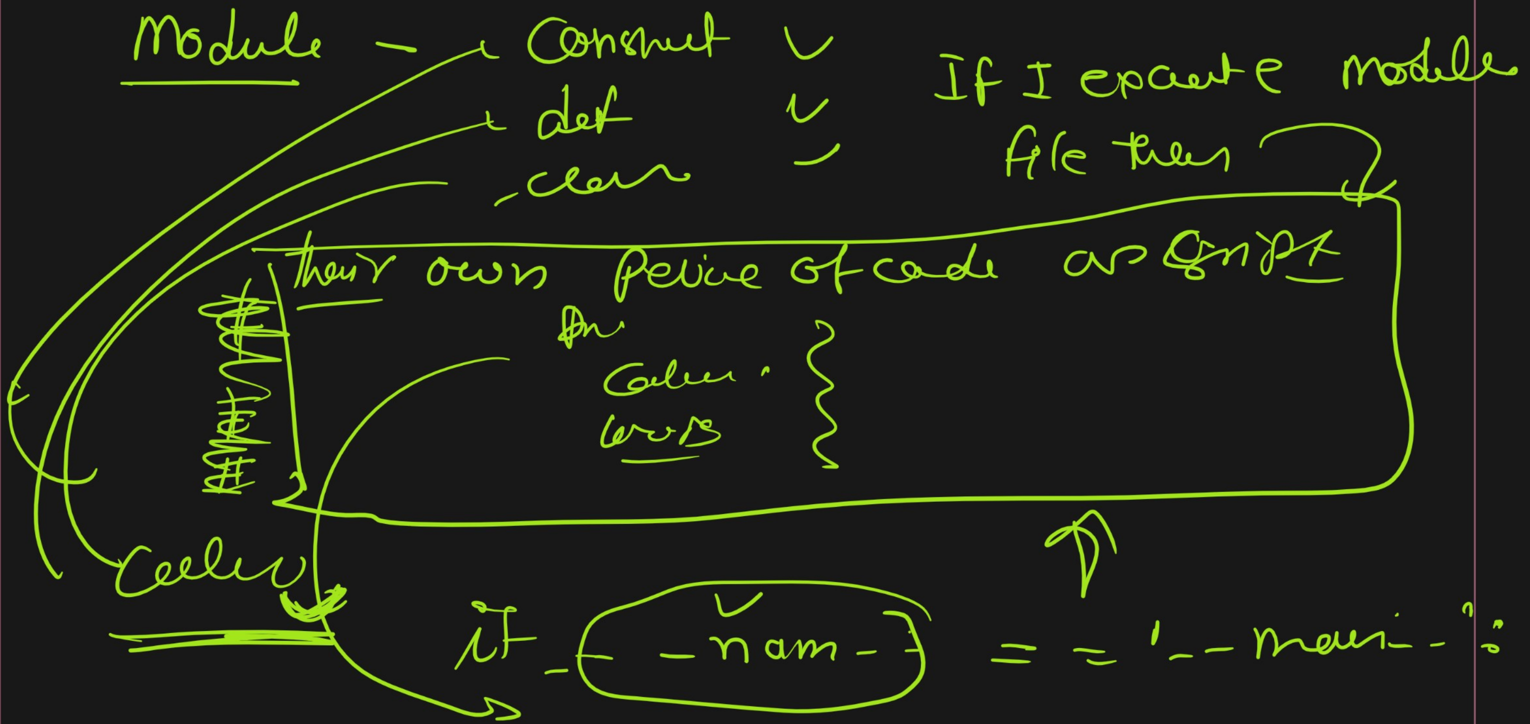
→ __main__

⇒

file1

file2

__main__



Module — constant ✓

def ✓

class ✓

If I execute module file then

their own piece of code as script

fn:

Calc.

Wrds

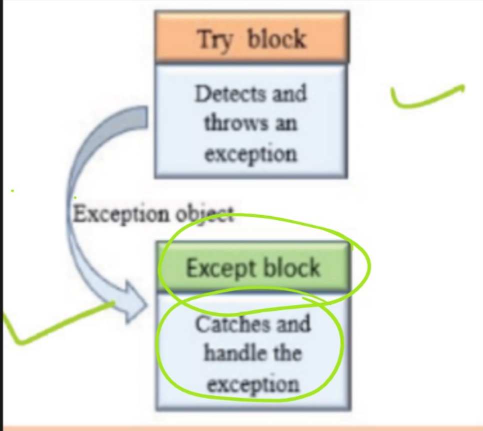
Calc

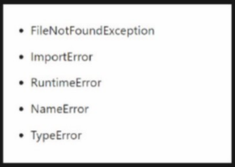
if __name__ == '__main__':

30 3 $\frac{3,10,000}{31} =$

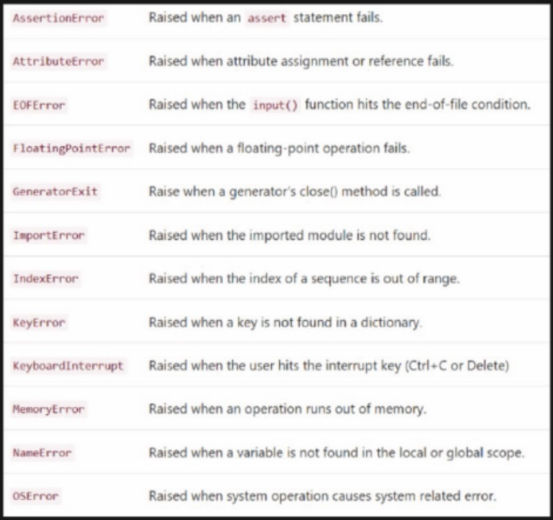
exception handling in python

An exception is an event that occurs during the execution of programs that **disrupt the normal flow of execution**


Try block
Detects and throws an exception
Exception object
Except block
Catches and handle the exception



- FileNotFoundException
- ImportError
- RuntimeError
- NameError
- TypeError



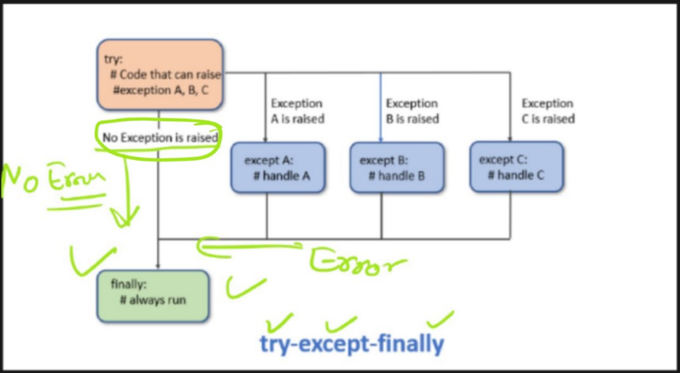
`AssertionError`	Raised when an `assert` statement fails.
`AttributeError`	Raised when attribute assignment or reference fails.
`EOFError`	Raised when the `input()` function hits the end-of-file condition.
`FloatingPointError`	Raised when a floating-point operation fails.
`GeneratorExit`	Raise when a generator's close() method is called.
`ImportError`	Raised when the imported module is not found.
`IndexError`	Raised when the index of a sequence is out of range.
`KeyError`	Raised when a key is not found in a dictionary.
`KeyboardInterrupt`	Raised when the user hits the interrupt key (Ctrl+C or Delete)
`MemoryError`	Raised when an operation runs out of memory.
`NameError`	Raised when a variable is not found in the local or global scope.
`OSError`	Raised when system operation causes system related error.

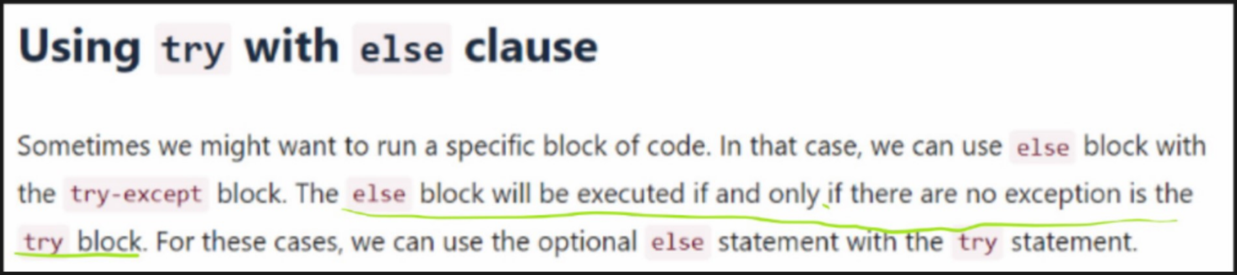
try-except-finally

Python provides the finally block, which is used with the try block statement. The finally block is used to write a block of code that must execute, whether the try block raises an error or not.

Mainly, the finally block is used to release the external resource. This block provides a guarantee of execution.

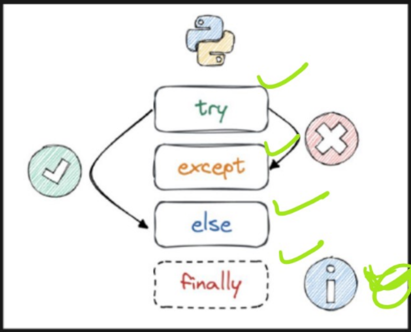
```
try :

except E1 :
except E2 :
   :
except Exam:

finaly :
```



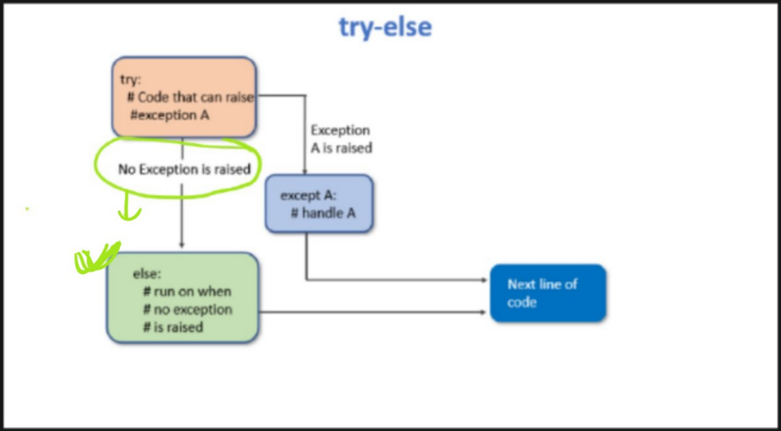


Using `try` with `else` clause

Sometimes we might want to run a specific block of code. In that case, we can use `else` block with the `try-except` block. The `else` block will be executed if and only if there are no exception is the `try` block. For these cases, we can use the optional `else` statement with the `try` statement.



Always



lines.txt ① * If ext → not .txt

→ ValueError

raised

* IncorrNam.Incorrex → ValueErr.

* In name .txt → IOError

A	Name	Ept
NameError	✗	✓
ValuError	✓	✗
IoError	✗	✗