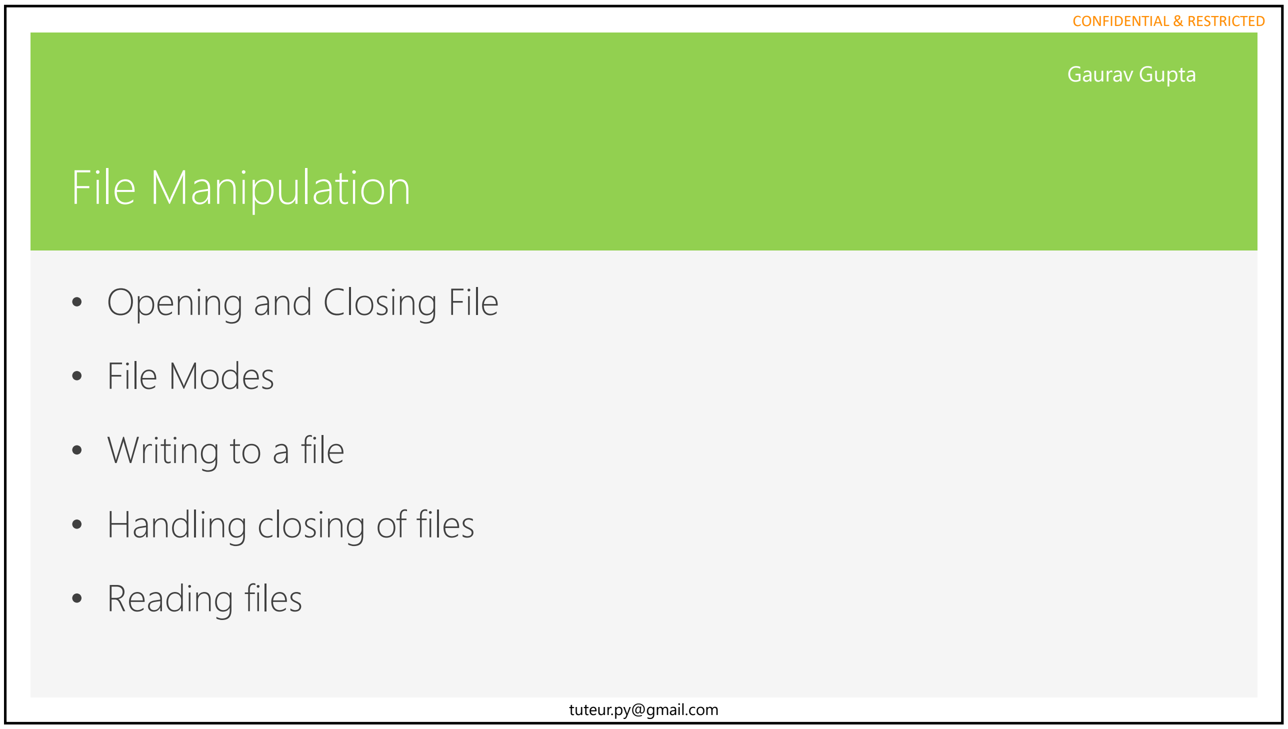
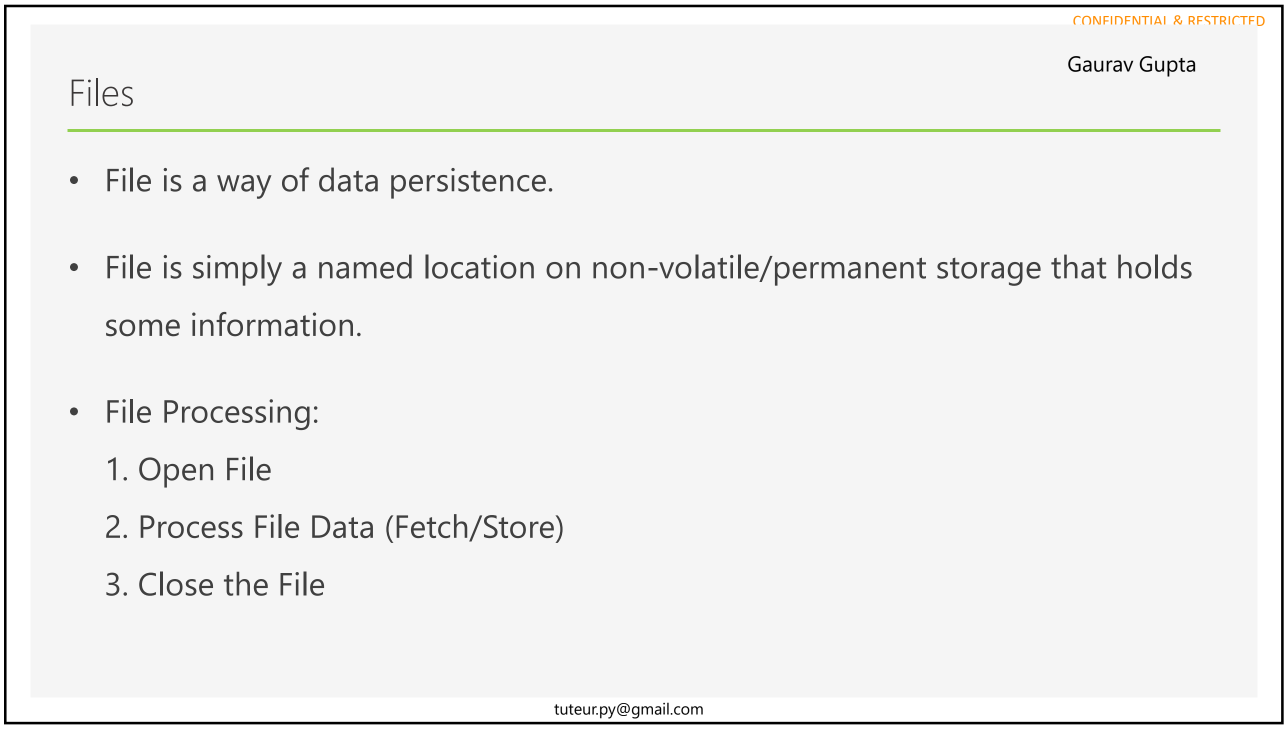


Gaurav Gupta

File Manipulation

- Opening and Closing File
- File Modes
- Writing to a file
- Handling closing of files
- Reading files

tuteur.py@gmail.com

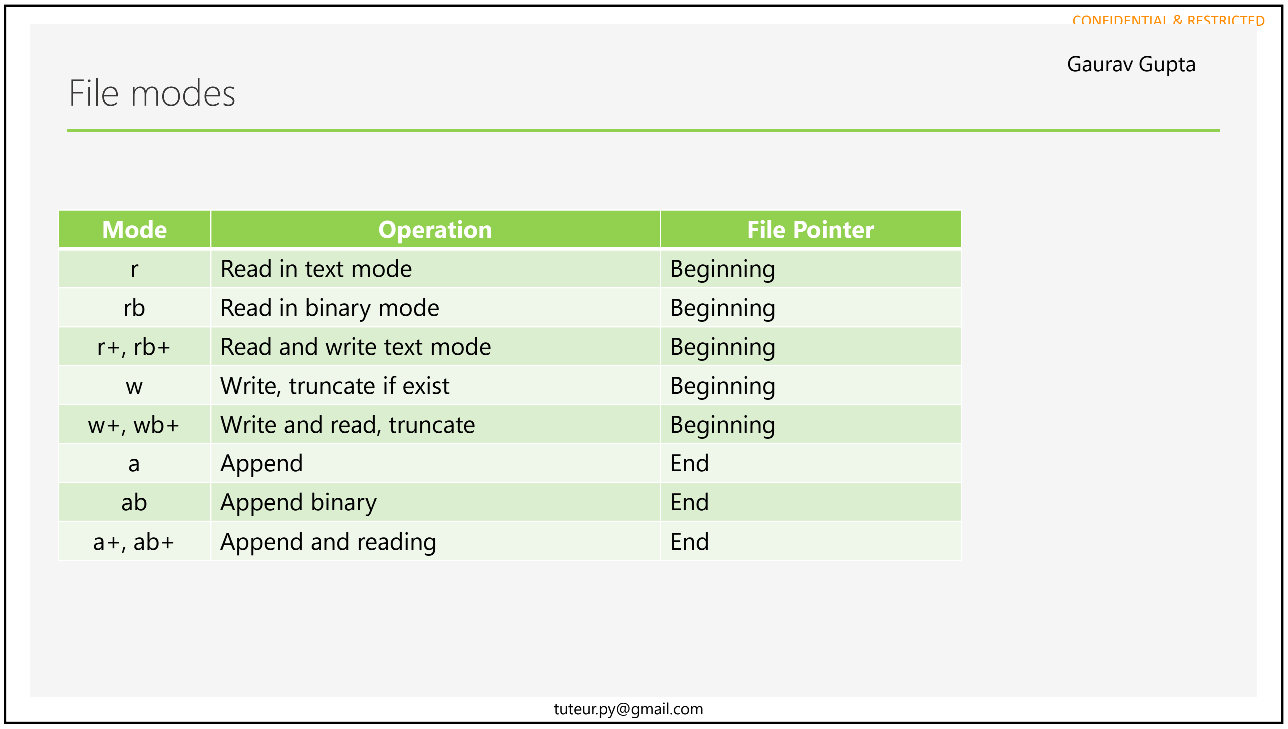


Gaurav Gupta

Files

- File is a way of data persistence.
- File is simply a named location on non-volatile/permanent storage that holds some information.
- File Processing:
  1. Open File
  2. Process File Data (Fetch/Store)
  3. Close the File

tuteur.py@gmail.com

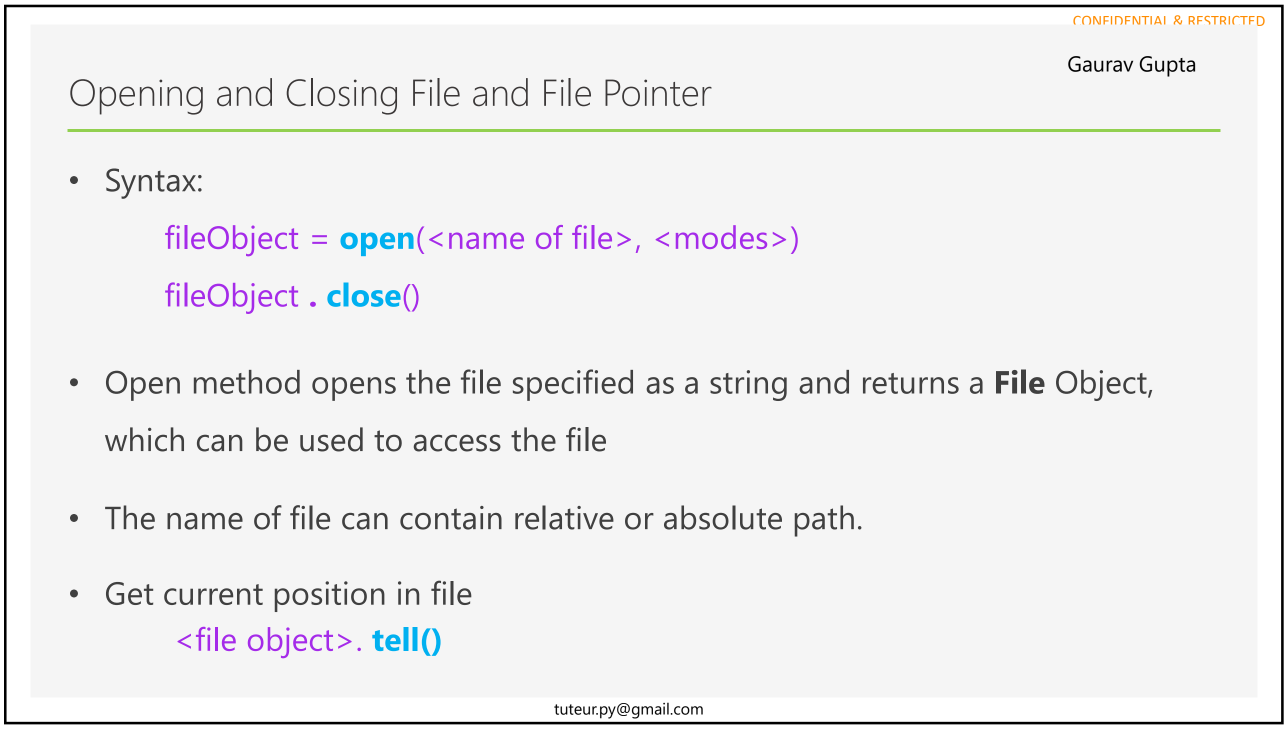


Gaurav Gupta

File modes

Mode	Operation	File Pointer
r	Read in text mode	Beginning
rb	Read in binary mode	Beginning
r+, rb+	Read and write text mode	Beginning
w	Write, truncate if exist	Beginning
w+, wb+	Write and read, truncate	Beginning
a	Append	End
ab	Append binary	End
a+, ab+	Append and reading	End

tuteur.py@gmail.com



Gaurav Gupta

Opening and Closing File and File Pointer

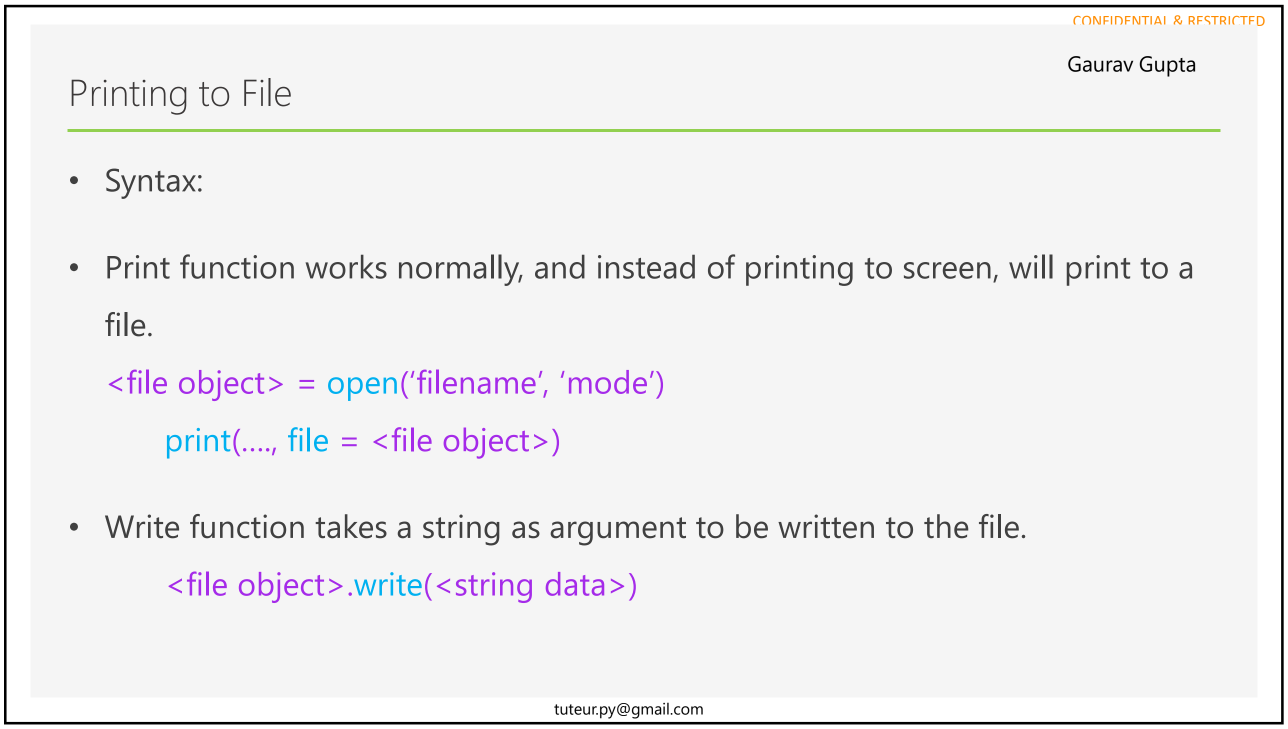
- Syntax:

```
fileObject = open(<name of file>, <modes>)
fileObject . close()
```

- Open method opens the file specified as a string and returns a **File** Object, which can be used to access the file
- The name of file can contain relative or absolute path.
- Get current position in file

```
<file object>. tell()
```

tuteur.py@gmail.com



Gaurav Gupta

Printing to File

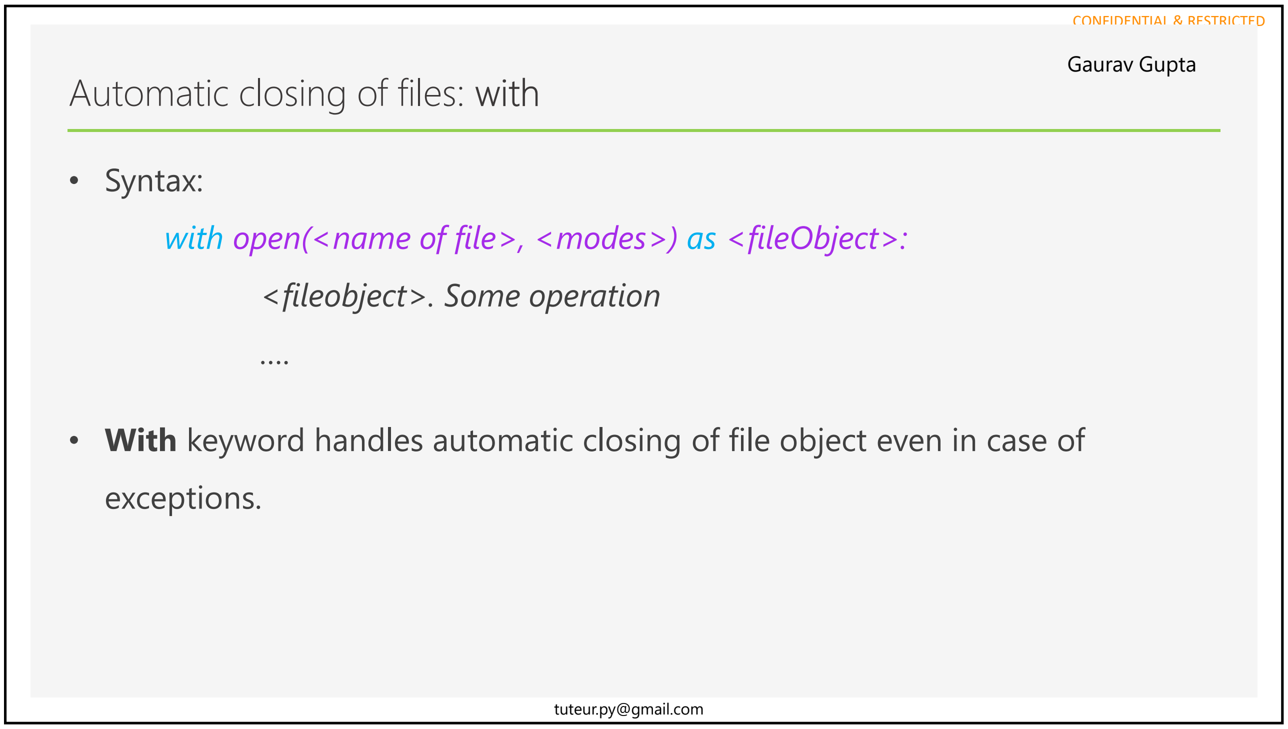
- Syntax:
- Print function works normally, and instead of printing to screen, will print to a file.

```
<file object> = open('filename', 'mode')
    print(...., file = <file object>)
```

- Write function takes a string as argument to be written to the file.

```
    <file object>.write(<string data>)
```

tuteur.py@gmail.com



Gaurav Gupta

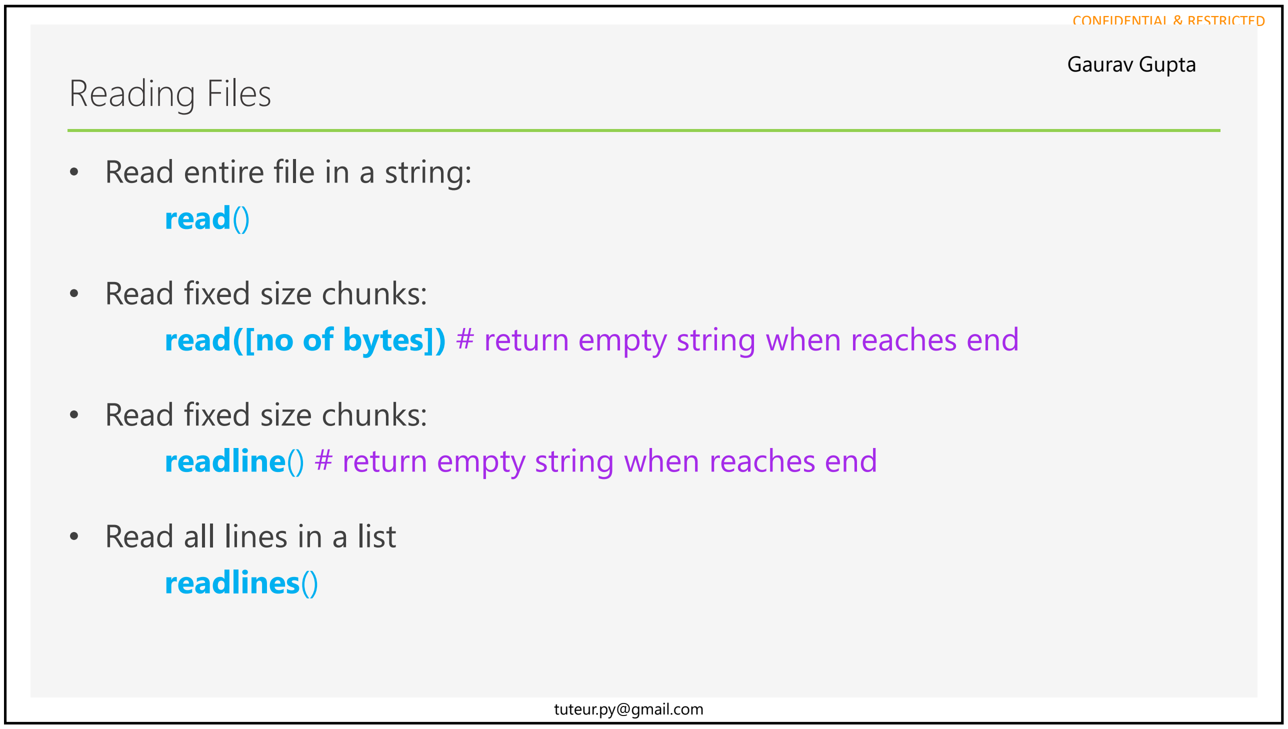
Automatic closing of files: with

- Syntax:

```
    with open(<name of file>, <modes>) as <fileObject>:
            <fileobject>. Some operation
            ....
```

- **With** keyword handles automatic closing of file object even in case of exceptions.

tuteur.py@gmail.com



Gaurav Gupta

Reading Files

- Read entire file in a string:

  read()

- Read fixed size chunks:

  read([no of bytes]) # return empty string when reaches end

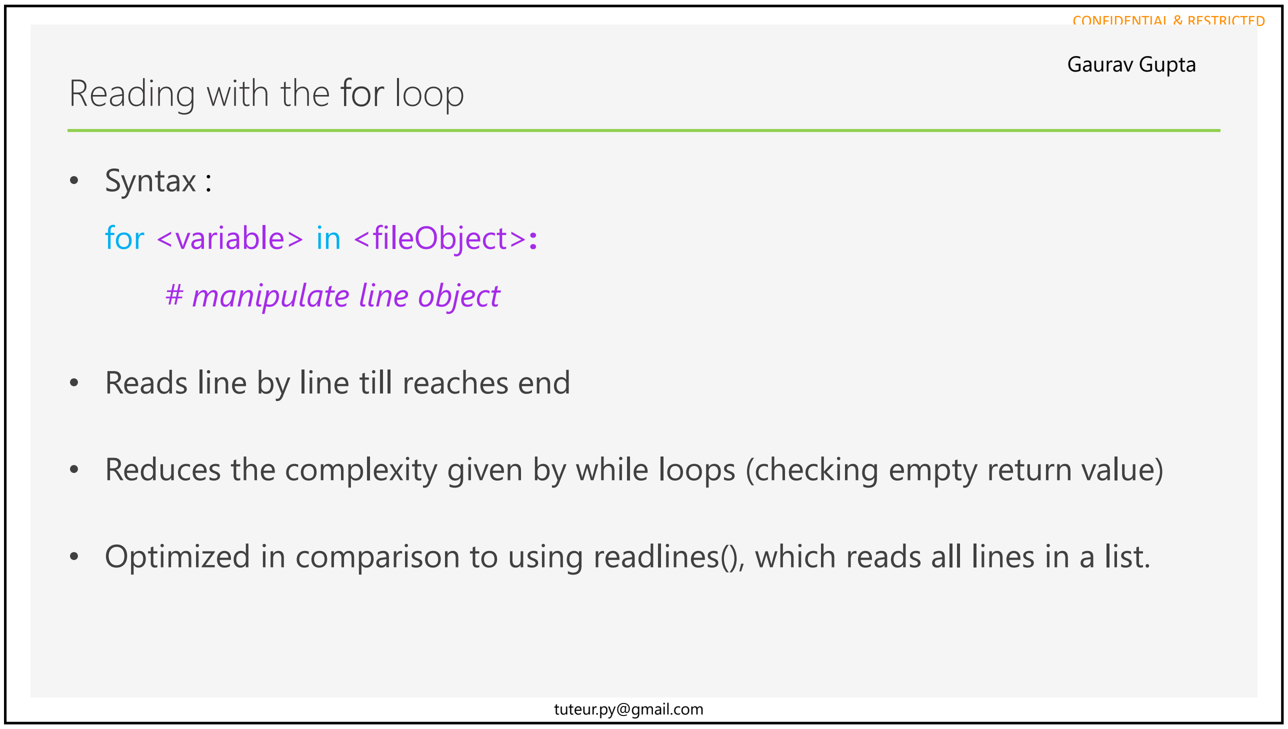
- Read fixed size chunks:

  readline() # return empty string when reaches end

- Read all lines in a list

  readlines()

tuteur.py@gmail.com



Gaurav Gupta

Reading with the for loop

- Syntax :

```
for <variable> in <fileObject>:
    # manipulate line object
```

- Reads line by line till reaches end
- Reduces the complexity given by while loops (checking empty return value)
- Optimized in comparison to using readlines(), which reads all lines in a list.

tuteur.py@gmail.com

Gaurav Gupta

Question

- WAP to dump everything in a file to the screen.
- Time to update our vowel counting skills.

  Writing a method to count vowels from a file.

Gaurav Gupta

File functions

- Flush is used to flush the contents to file forcefully

  <file object>.**flush()**

- Roam around in file

  <file object>. **seek(** <offset>, <pos> **)**

  pos = 0: beginning # this is default
  pos = 1: current
  pos = 2: end

Gaurav Gupta

Some os Operations

- **os** module contains the following functions:
- *getcwd()* : gives current working directory
  chdir(<path>) : changes current working directory
- *mkdir(<name of directory>)* : create folder in current directory or absolute path
  makedirs(< >) : creates multiple folders appearing in the path if they don't already exist
- *rmdir(<path>)* : the directory to be deleted must be empty
  rename(<source>, <dest>) : source and destination should be on same drive

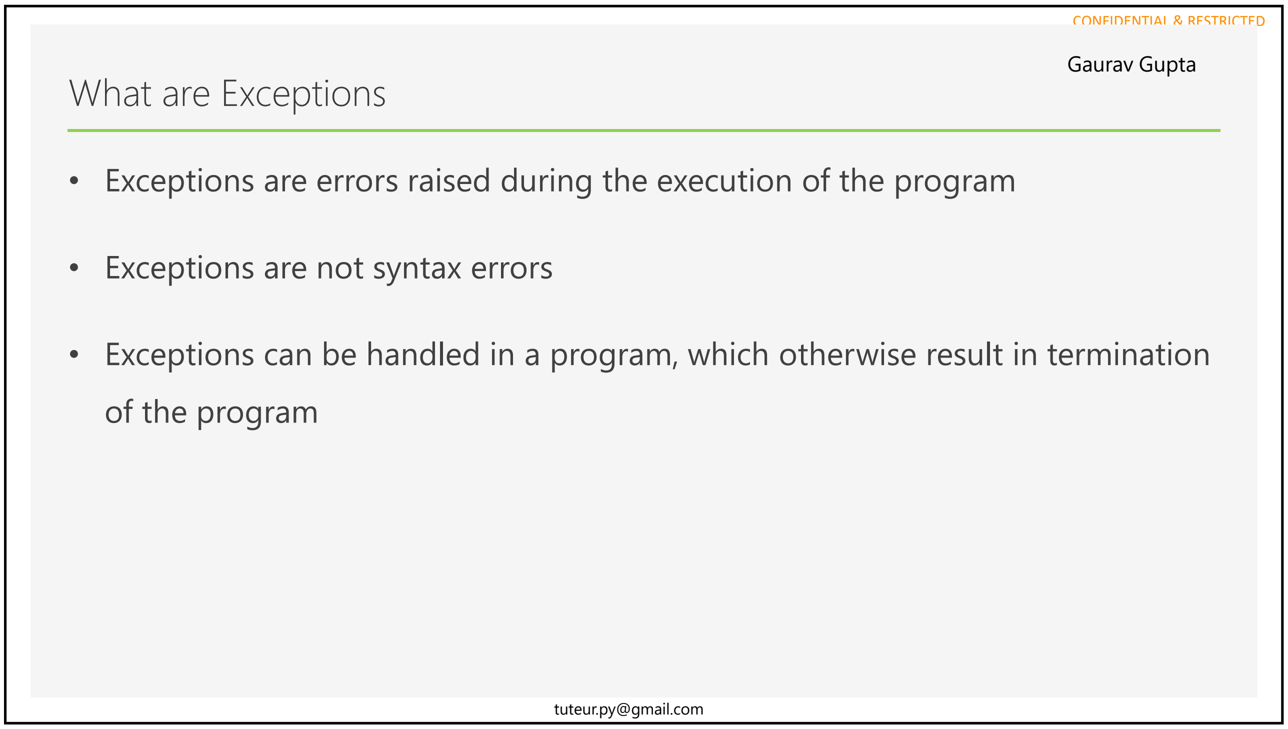
tuteur.py@gmail.com

Gaurav Gupta

Exceptions

- What are Exceptions
- Try Except Syntax
- How it Works
- Multiple Except Statements
- Raising Exceptions
- Complete try – except – else – finally syntax

tuteur.py@gmail.com

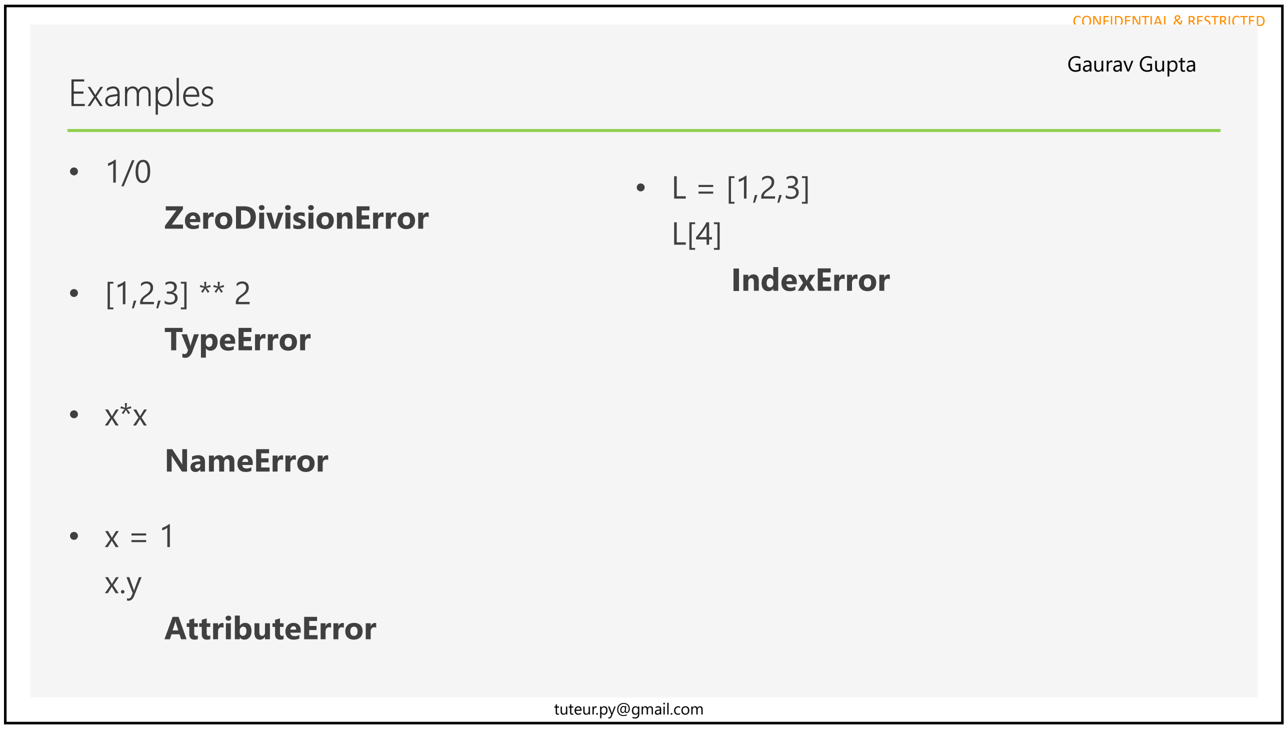


Gaurav Gupta

What are Exceptions

- Exceptions are errors raised during the execution of the program
- Exceptions are not syntax errors
- Exceptions can be handled in a program, which otherwise result in termination of the program

tuteur.py@gmail.com

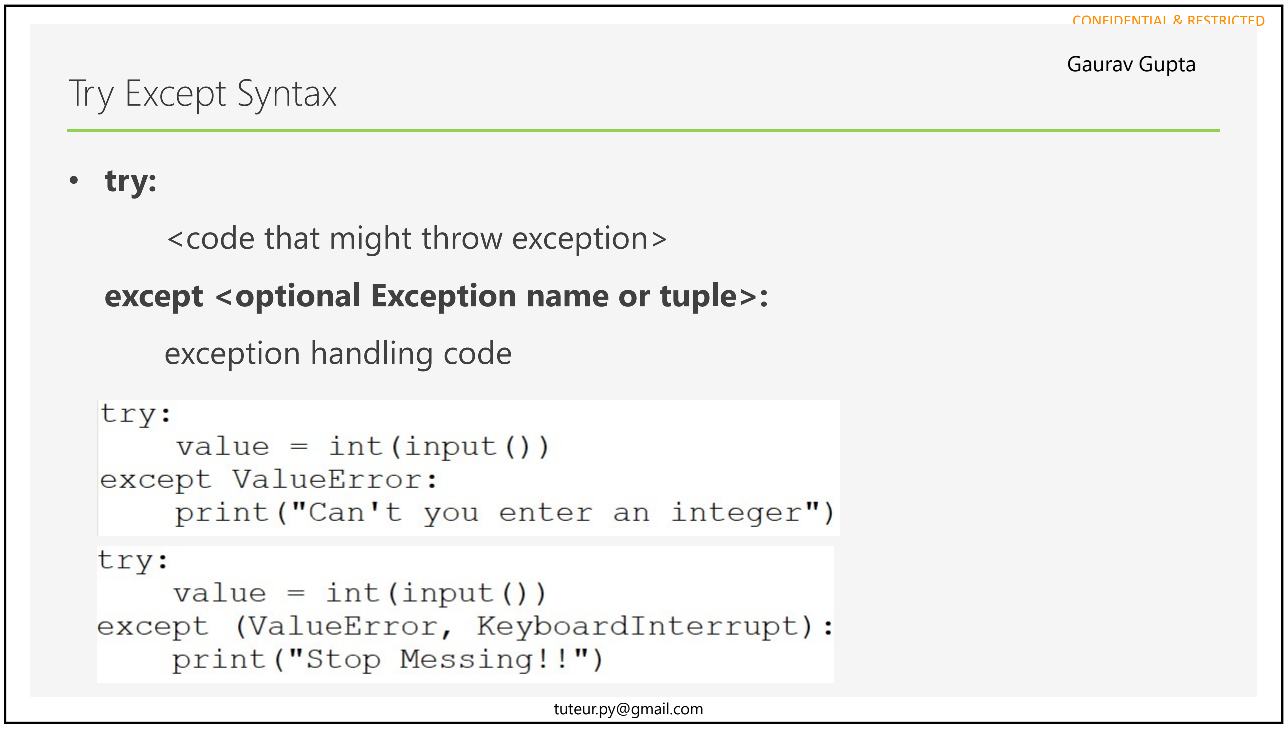


Gaurav Gupta

Examples

- 1/0
  ZeroDivisionError
- [1,2,3] ** 2
  TypeError
- x*x
  NameError
- x = 1
  x.y
  AttributeError
- L = [1,2,3]
  L[4]
  IndexError

tuteur.py@gmail.com



Gaurav Gupta

Try Except Syntax

- **try:**

    <code that might throw exception>

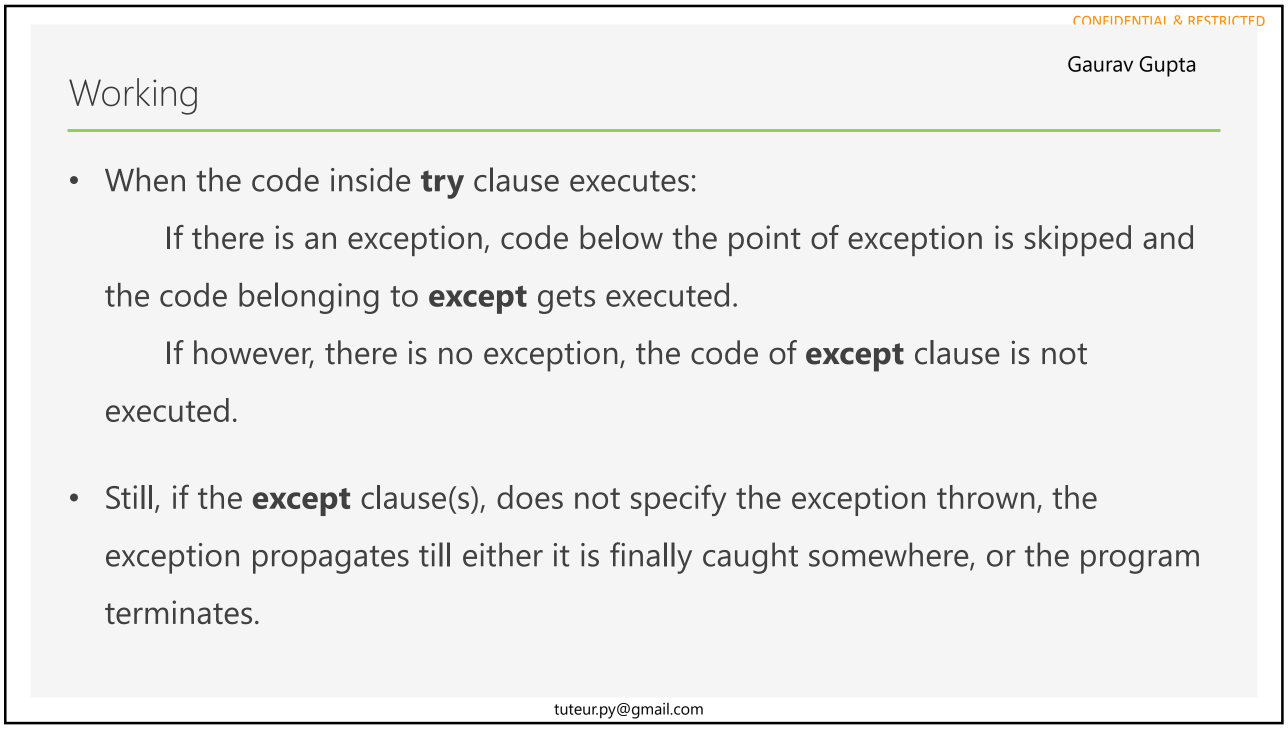
  except <optional Exception name or tuple>:

    exception handling code

```
try:
    value = int(input())
except ValueError:
    print("Can't you enter an integer")
```

```
try:
    value = int(input())
except (ValueError, KeyboardInterrupt):
    print("Stop Messing!!")
```

tuteur.py@gmail.com



Gaurav Gupta

Working

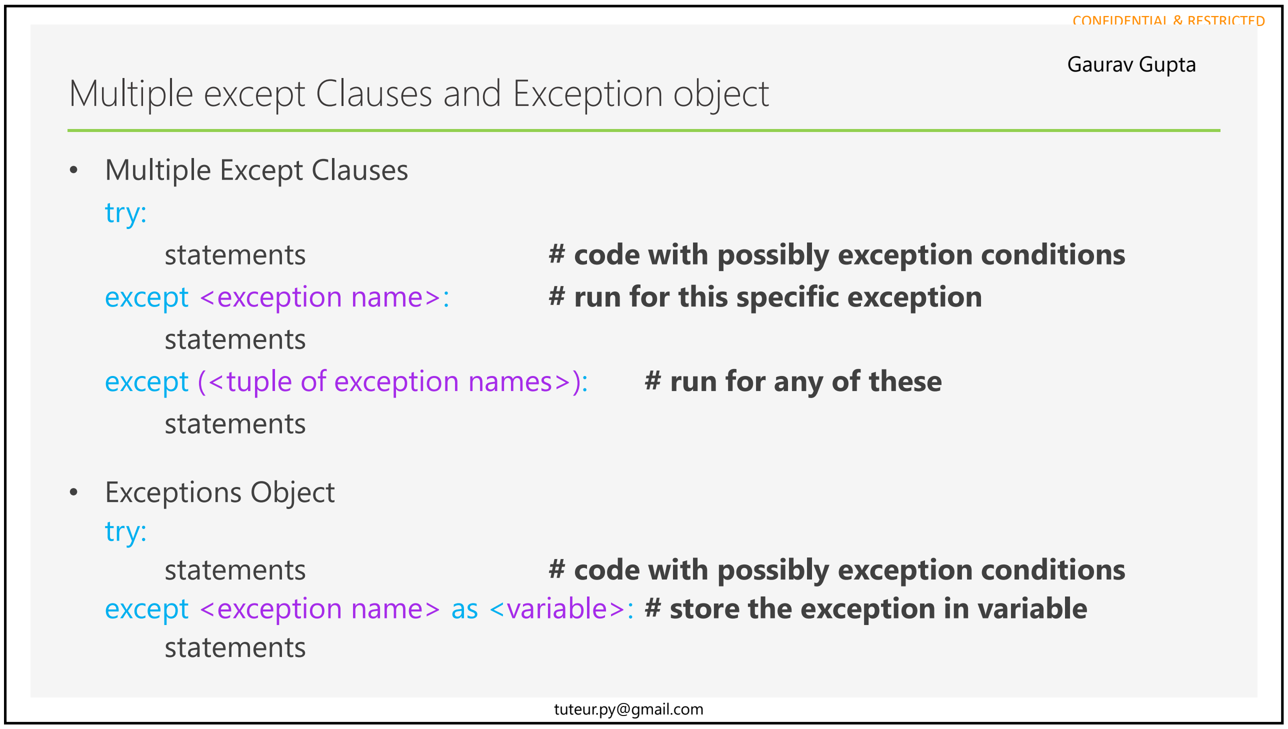
- When the code inside **try** clause executes:

    If there is an exception, code below the point of exception is skipped and the code belonging to **except** gets executed.

    If however, there is no exception, the code of **except** clause is not executed.

- Still, if the **except** clause(s), does not specify the exception thrown, the exception propagates till either it is finally caught somewhere, or the program terminates.

tuteur.py@gmail.com



Gaurav Gupta

Multiple except Clauses and Exception object

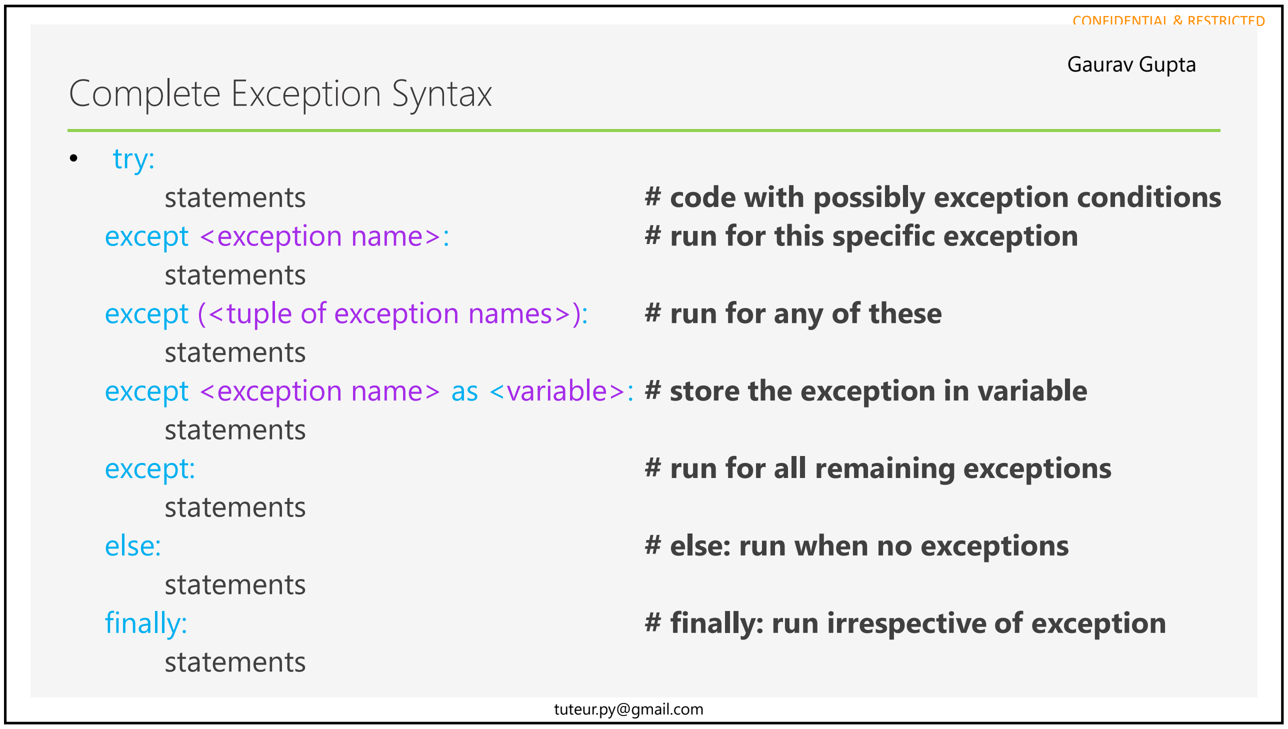
- Multiple Except Clauses

```
try:
        statements                              # code with possibly exception conditions
except <exception name>:                        # run for this specific exception
        statements
except (<tuple of exception names>):            # run for any of these
        statements
```

- Exceptions Object

```
try:
        statements                              # code with possibly exception conditions
except <exception name> as <variable>:          # store the exception in variable
        statements
```

tuteur.py@gmail.com

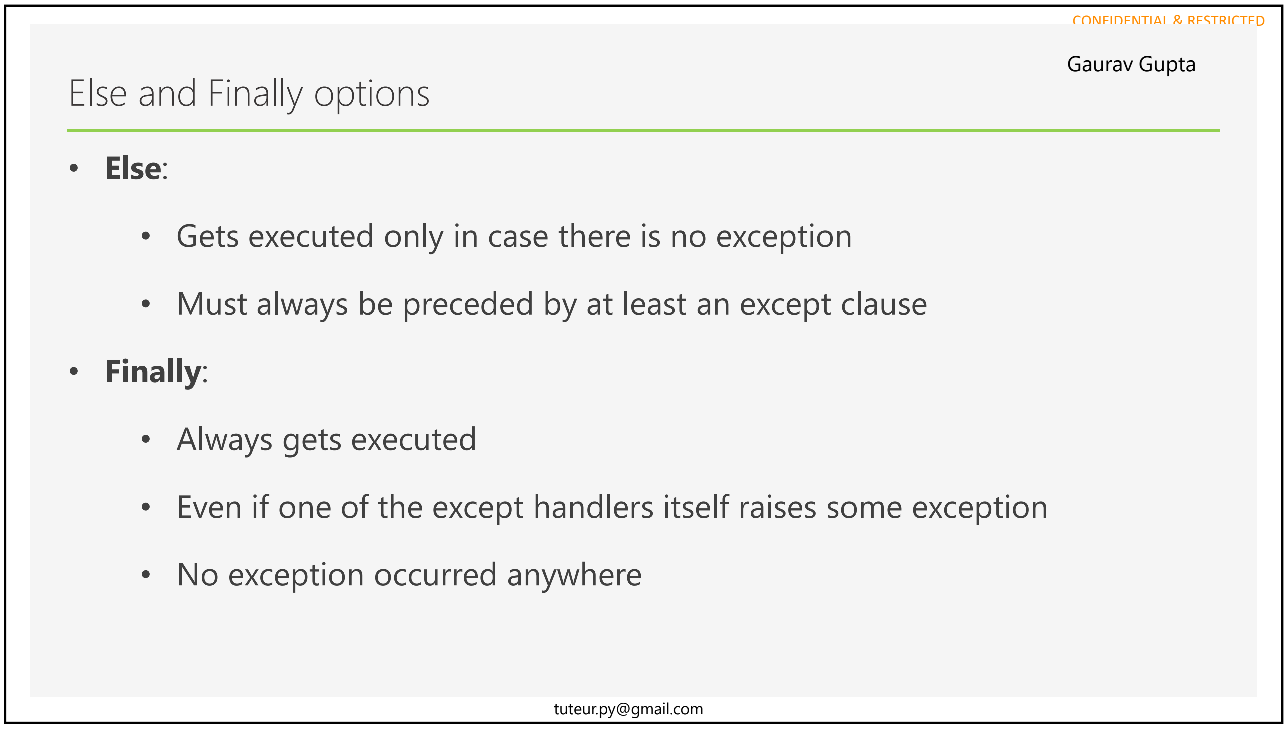


Gaurav Gupta

Complete Exception Syntax

-
  ```
  try:
          statements                              # code with possibly exception conditions
  except <exception name>:                        # run for this specific exception
          statements
  except (<tuple of exception names>):            # run for any of these
          statements
  except <exception name> as <variable>:          # store the exception in variable
          statements
  except:                                         # run for all remaining exceptions
          statements
  else:                                           # else: run when no exceptions
          statements
  finally:                                        # finally: run irrespective of exception
          statements
  ```

tuteur.py@gmail.com

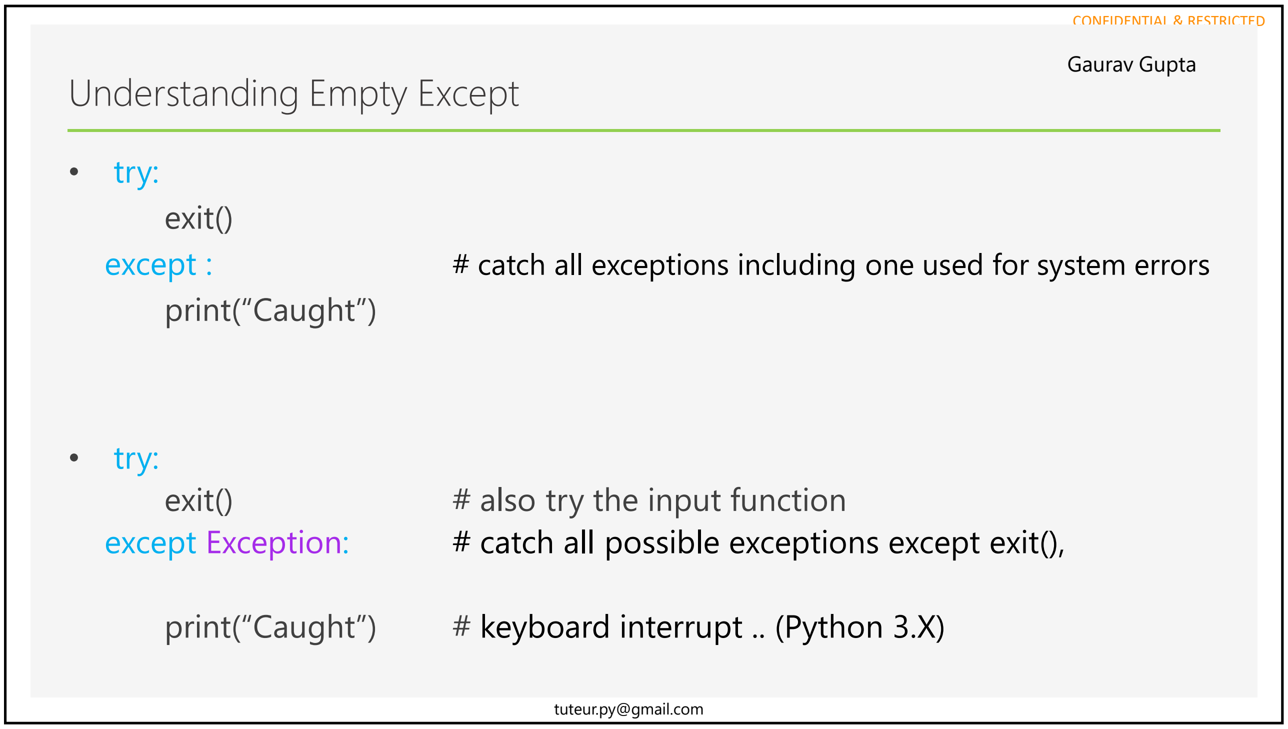


Gaurav Gupta

Else and Finally options

- **Else**:
  - Gets executed only in case there is no exception
  - Must always be preceded by at least an except clause
- **Finally**:
  - Always gets executed
  - Even if one of the except handlers itself raises some exception
  - No exception occurred anywhere

tuteur.py@gmail.com



Gaurav Gupta

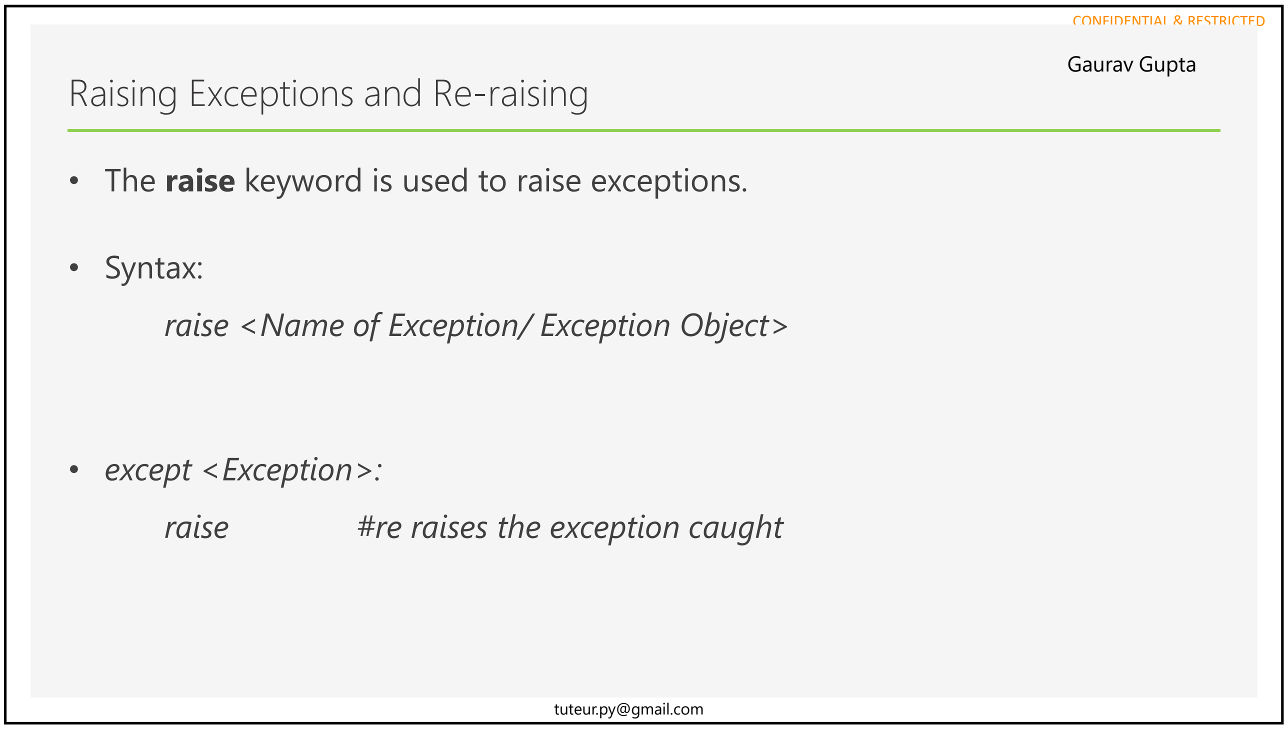
Understanding Empty Except

```
• try:
      exit()
  except :                # catch all exceptions including one used for system errors
      print("Caught")
```

```
• try:
      exit()              # also try the input function
  except Exception:       # catch all possible exceptions except exit(),

      print("Caught")     # keyboard interrupt .. (Python 3.X)
```

tuteur.py@gmail.com



Gaurav Gupta

Raising Exceptions and Re-raising

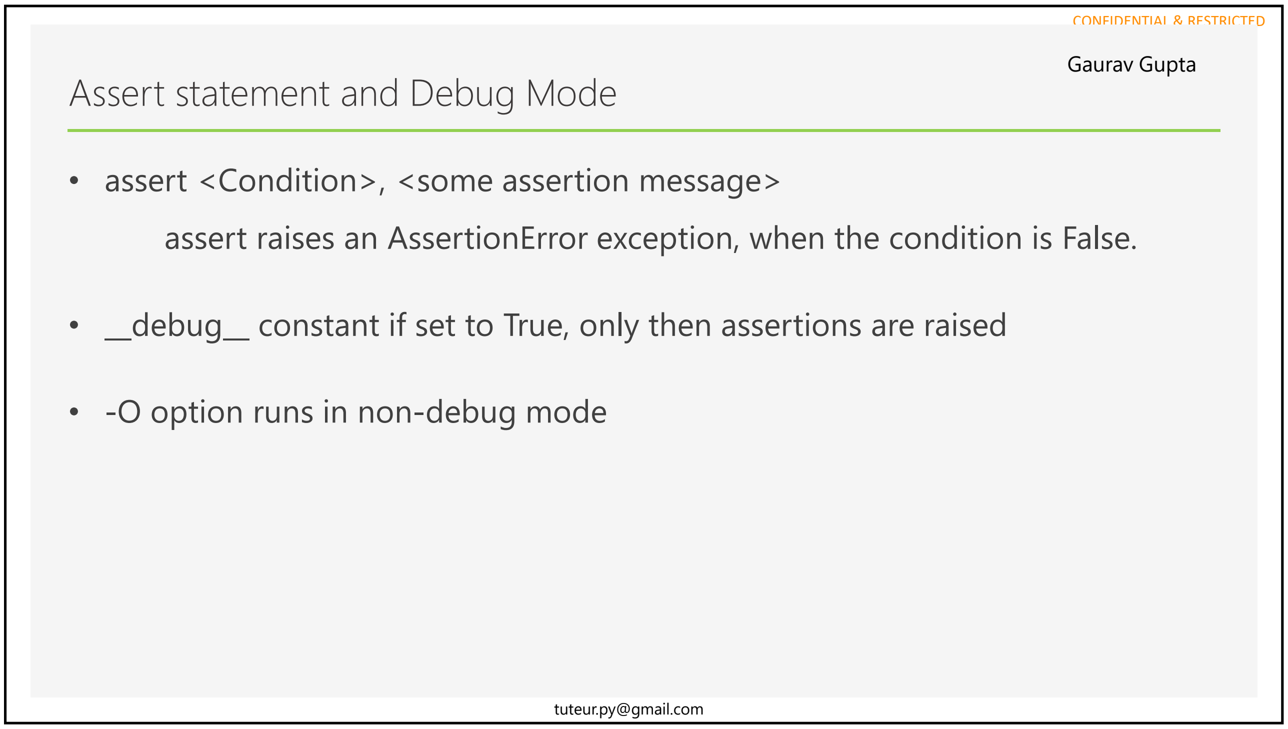
- The **raise** keyword is used to raise exceptions.
- Syntax:

  raise <Name of Exception/ Exception Object>

- *except <Exception>:*

  raise #re raises the exception caught

tuteur.py@gmail.com



Gaurav Gupta

Assert statement and Debug Mode

- assert <Condition>, <some assertion message>

  assert raises an AssertionError exception, when the condition is False.

- __debug__ constant if set to True, only then assertions are raised
- -O option runs in non-debug mode

tuteur.py@gmail.com