Gaurav Gupta

Iterators

- Iterator vs Iterable
- Understanding with list example
- Iterable Requirements
- Iterator Requirements

tuteur.py@gmail.com

Gaurav Gupta

Iterator vs Iterable

- An iterator is an object that allows the next method to be called upon it and returns values.
- In iterable is an object that has the __iter__ method, which returns an iterator.
- Ex: **list** is an **iterable**

  calling the __**iter**__ method return an iterator

tuteur.py@gmail.com

Gaurav Gupta

Iterable Requirements

- Should support an __**iter**__ method which returns an iterator object upon calling.
- Example:

```
l = [1, 2, 3]
dir(l)
it1 = l.__iter__()
it2 = iter(l)
```

tuteur.py@gmail.com

Gaurav Gupta

Iterator requirements

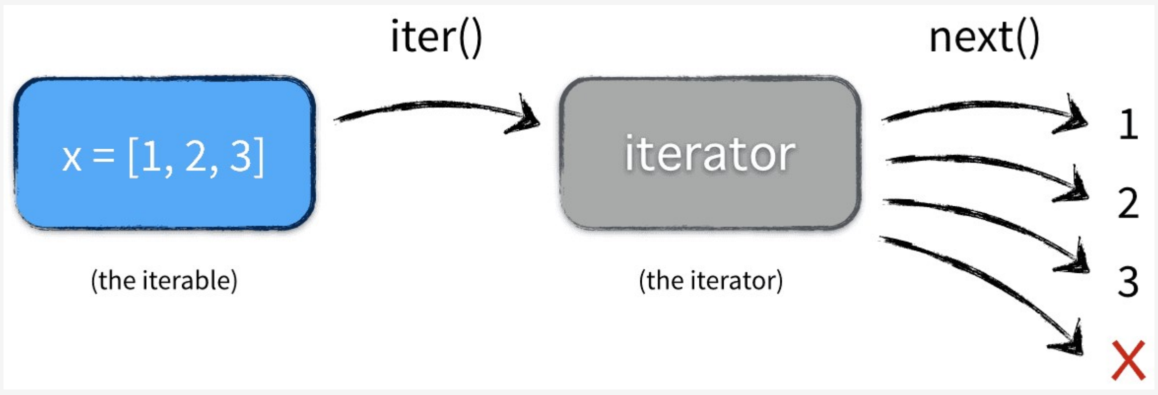
- An iterator should support the __**next**__ method.
- Should raise a **StopIteration** exception upon reaching the last element to be iterated.
- Example:

```
l = [1, 2, 3]
itr = iter(l)
itr.__next__()
next(itr)
```

tuteur.py@gmail.com

Gaurav Gupta

Understanding with list example

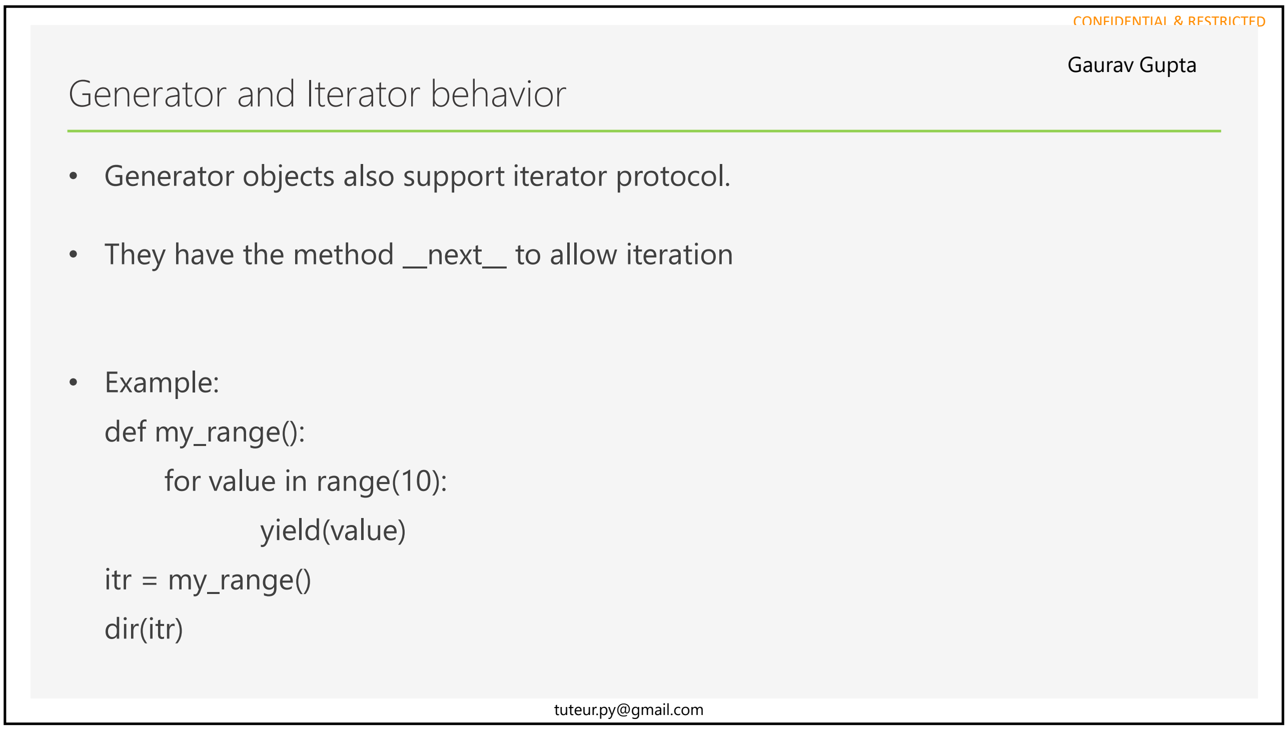


Gaurav Gupta

World without for loops

```
• l = [1, 2, 3]
  i = 0
  while i < len( l ):
      print( l[i] )
      i += 1
```

```
• It = iter( l )
  try:
      while True:
          print( next( it ) )
  except:
      pass
```



Gaurav Gupta

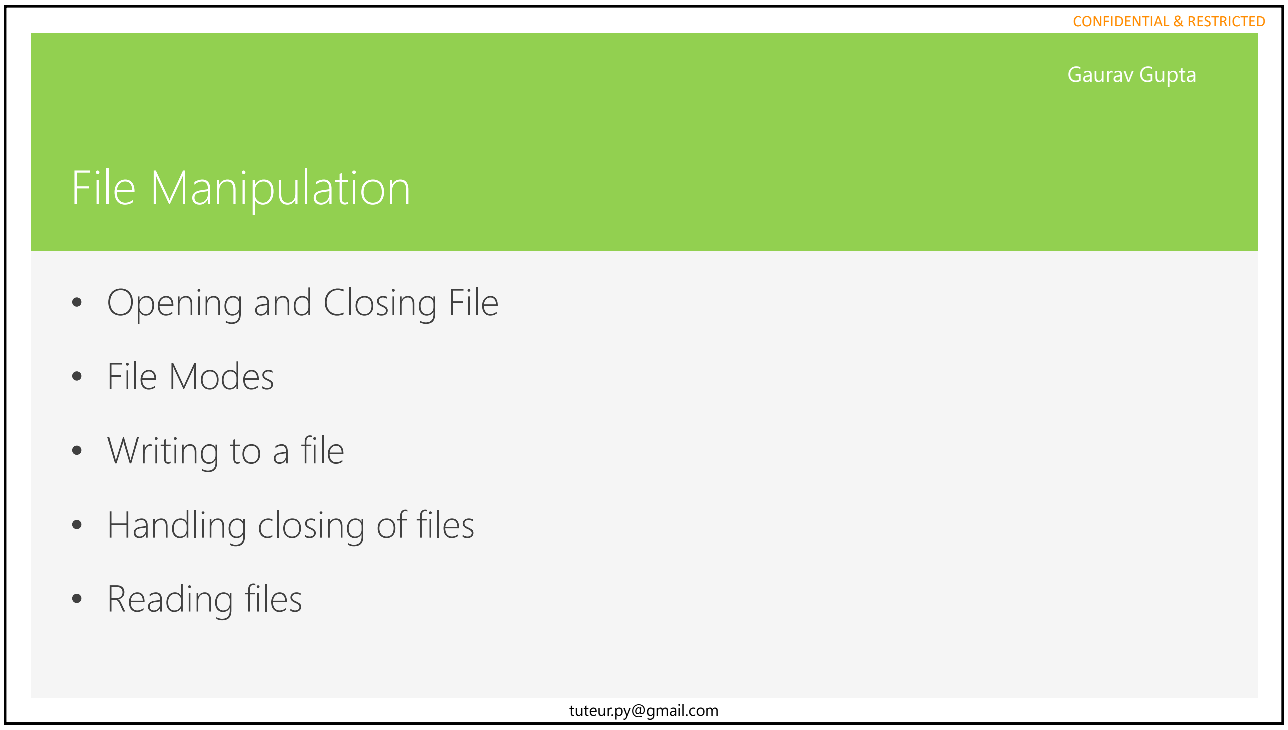
Generator and Iterator behavior

- Generator objects also support iterator protocol.
- They have the method __next__ to allow iteration
- Example:

```
def my_range():
        for value in range(10):
                yield(value)
itr = my_range()
dir(itr)
```

tuteur.py@gmail.com

107



Gaurav Gupta

File Manipulation

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

tuteur.py@gmail.com

108

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()**

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>)

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

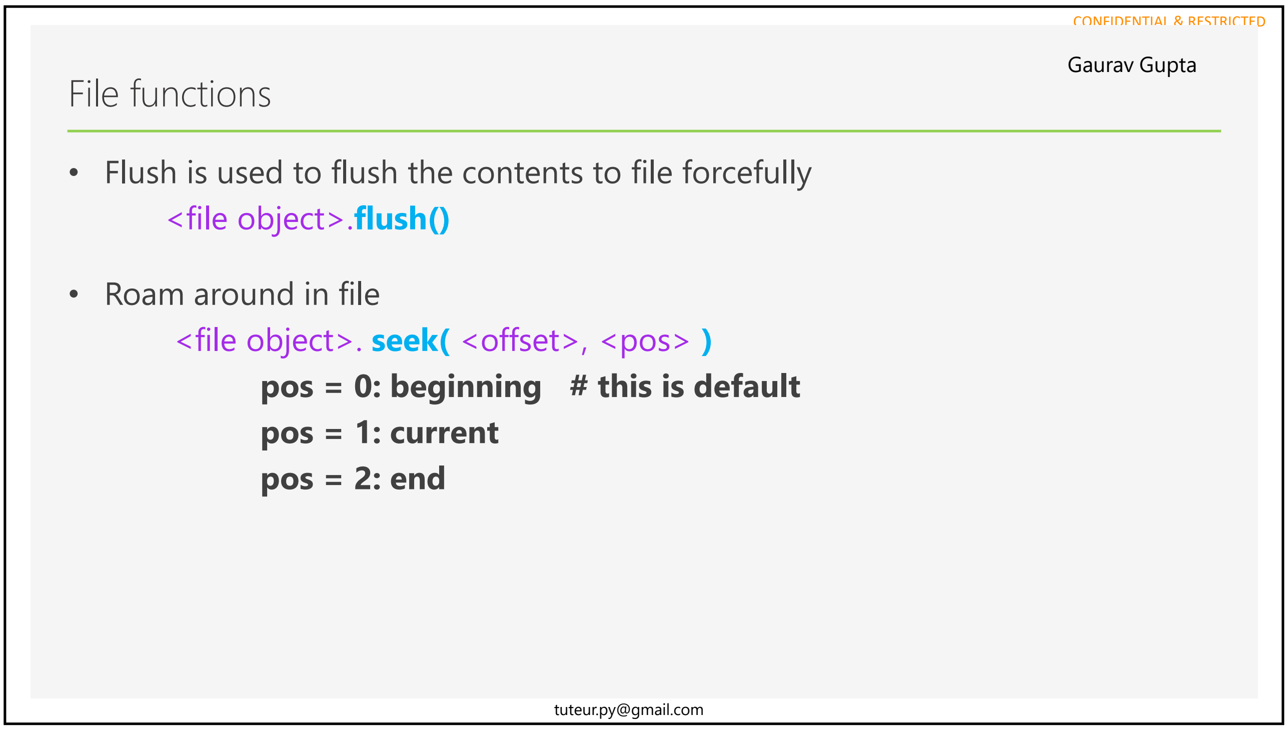
Gaurav Gupta

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.

tuteur.py@gmail.com



Gaurav Gupta

File functions

- Flush is used to flush the contents to file forcefully

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

- Roam around in file

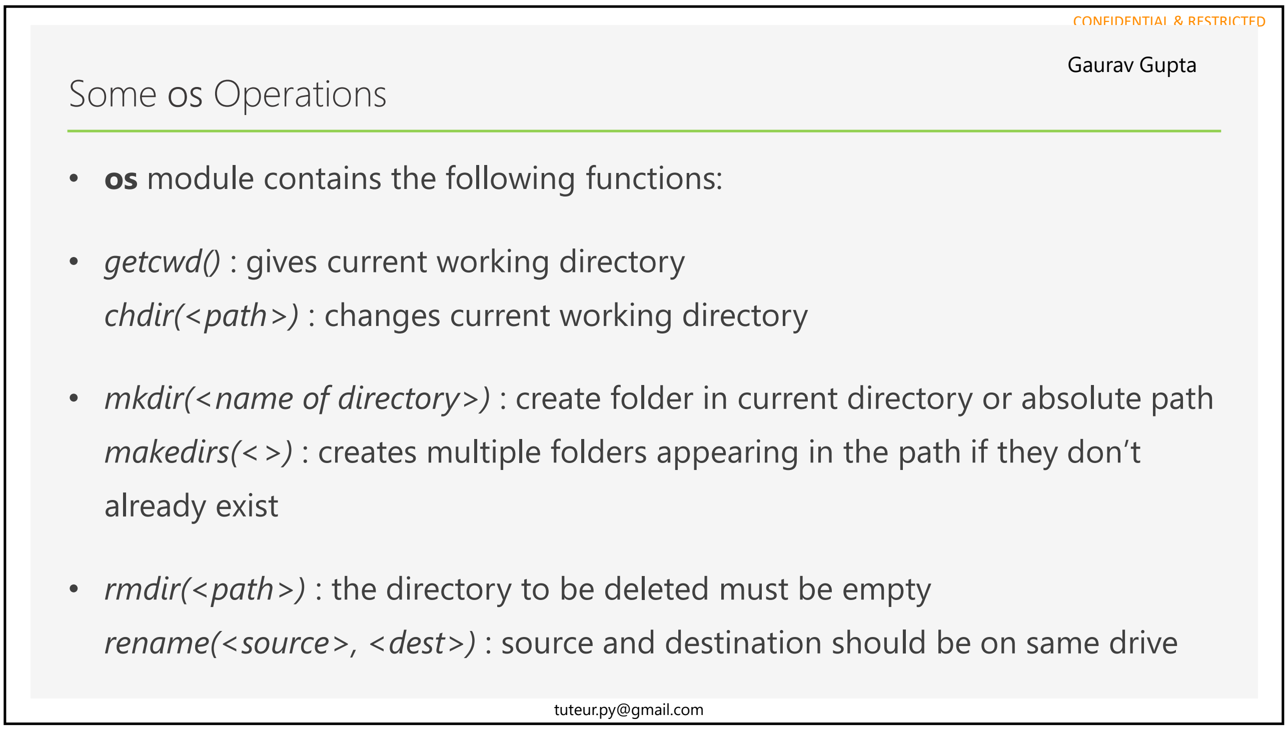
  <file object>. **seek(** <offset>, <pos> **)**

  pos = 0: beginning   # this is default

  pos = 1: current

  pos = 2: end

117



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

118

Gaurav Gupta

Exceptions

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

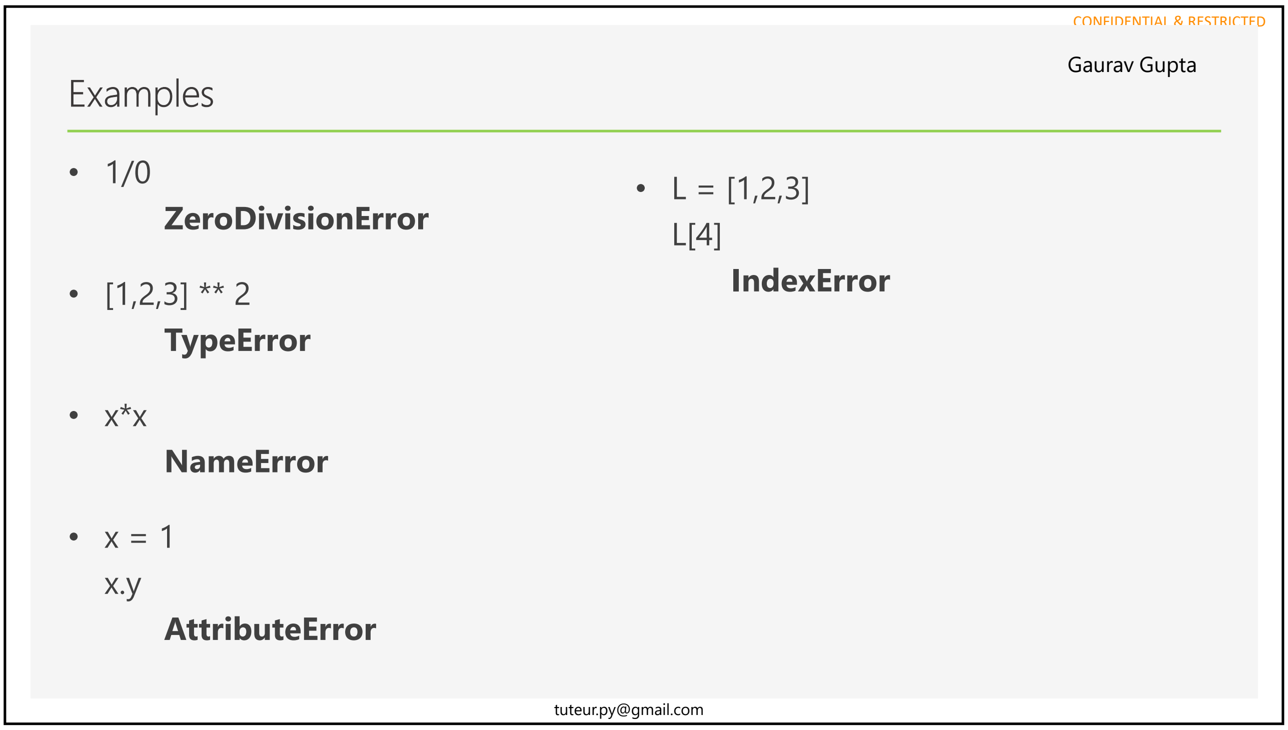
tuteur.py@gmail.com

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

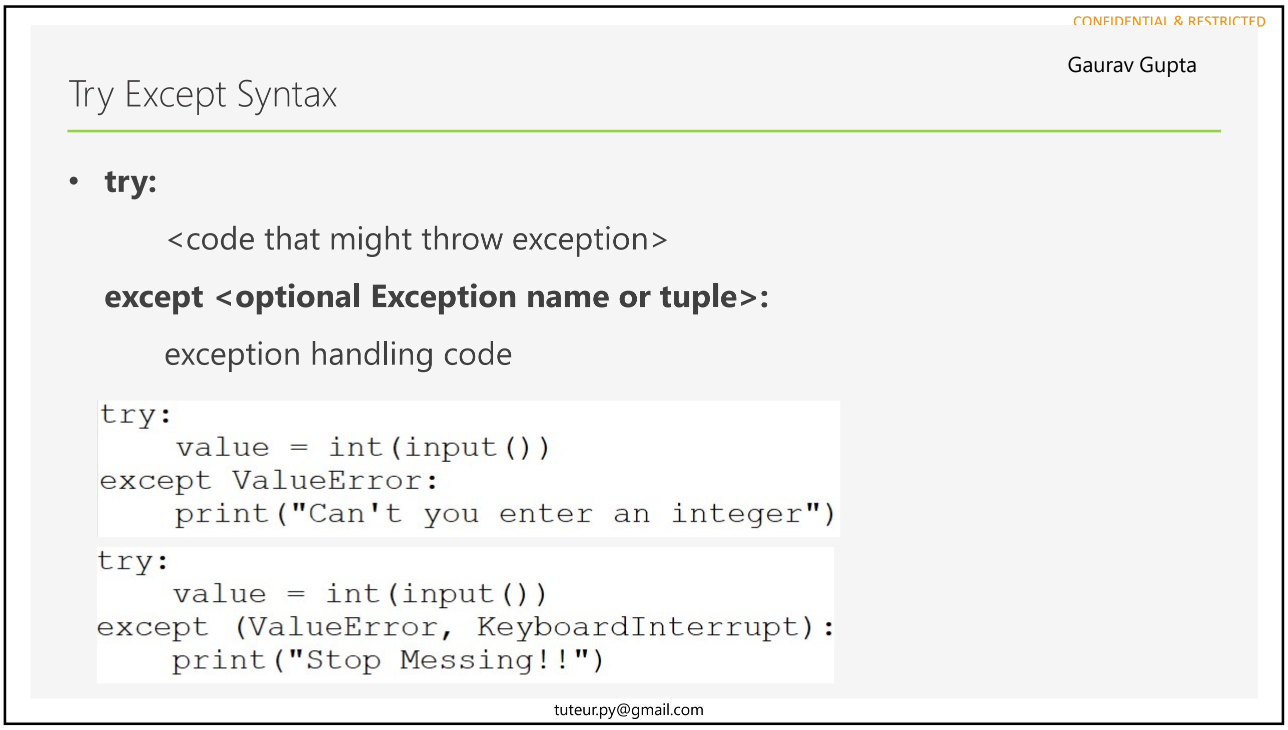
  AttributeError

- L = [1,2,3]
  L[4]

  IndexError

tuteur.py@gmail.com

121



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

122

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

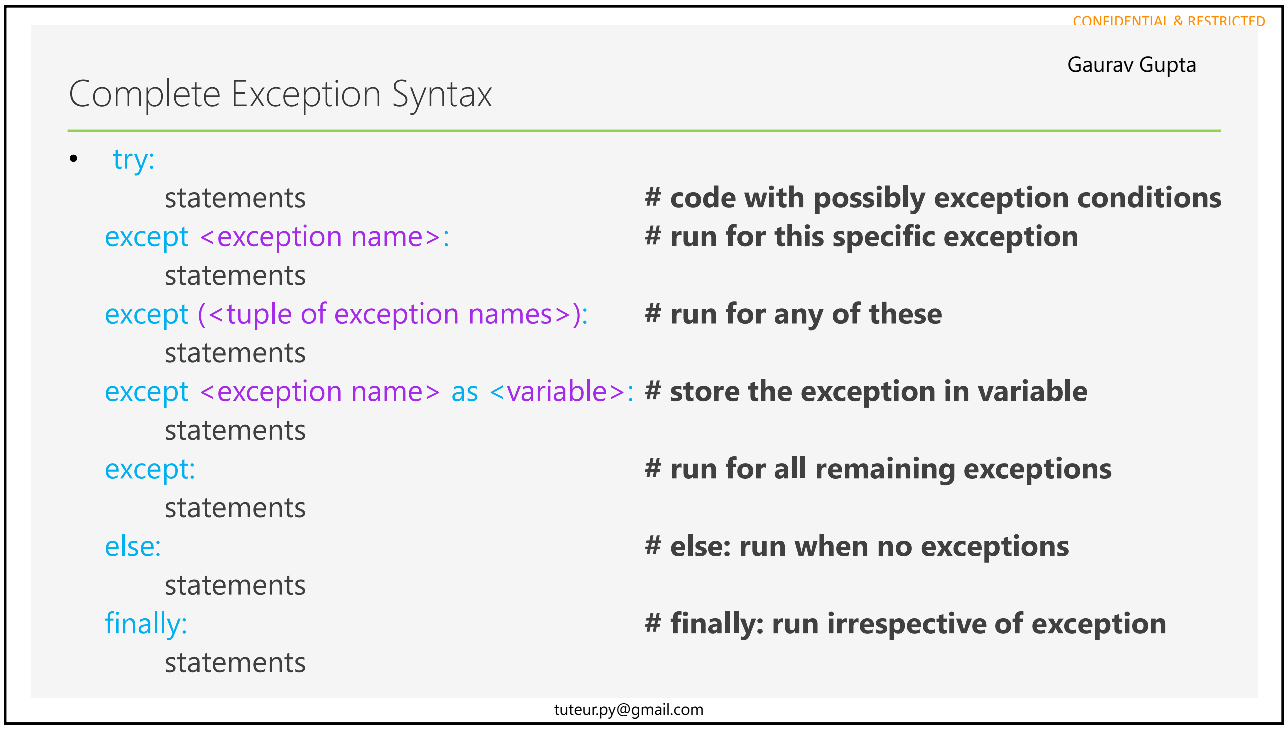
- Multiple Except Clauses

```
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

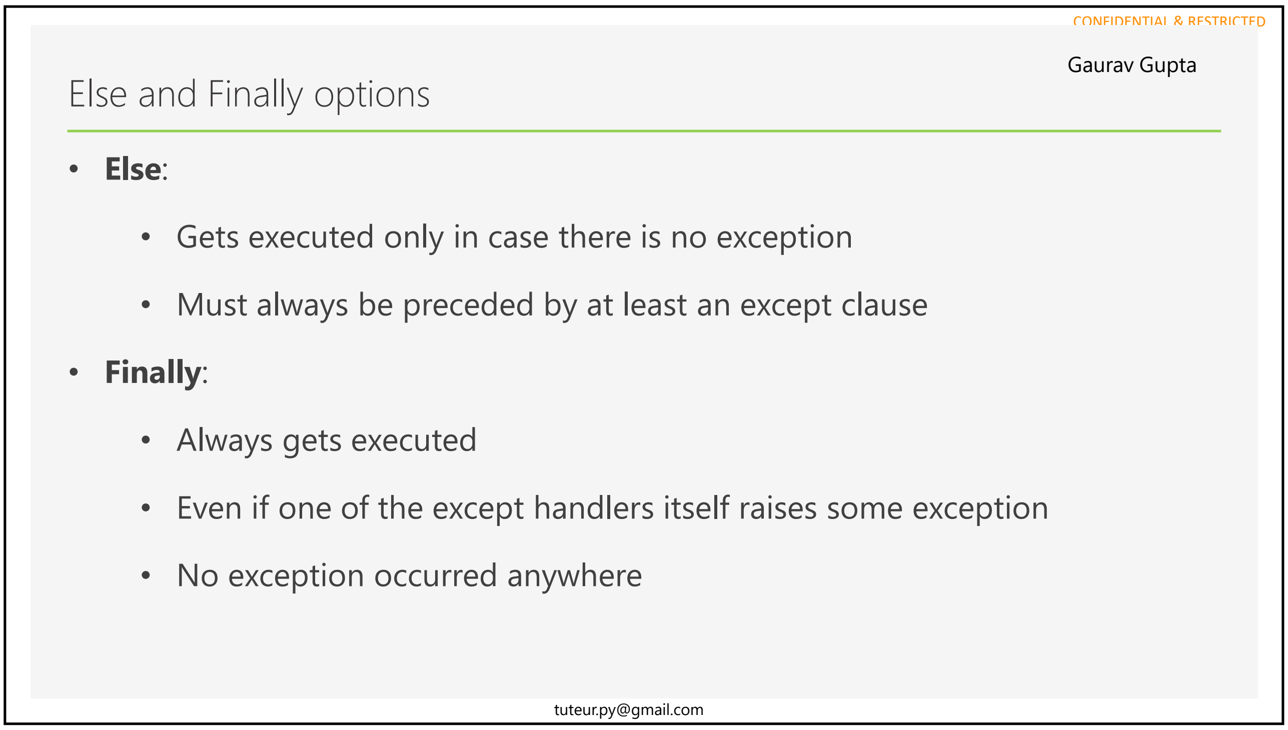


Gaurav Gupta

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

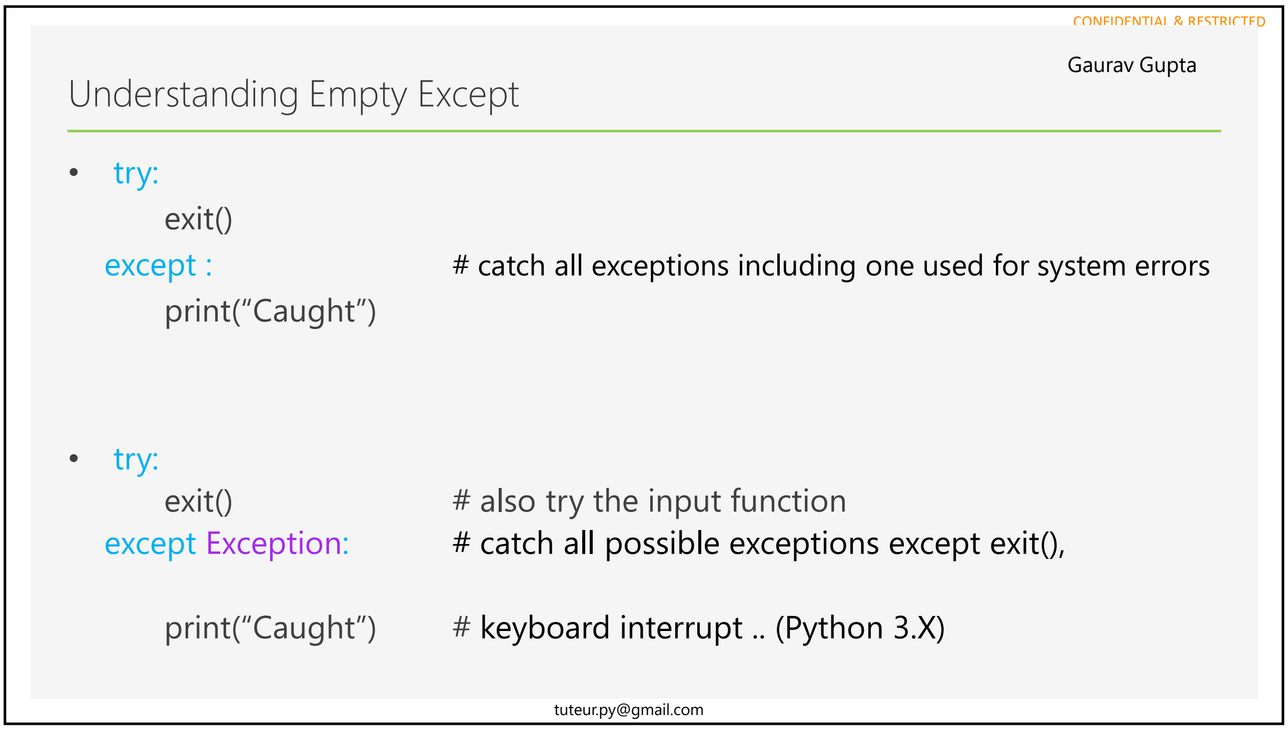


Gaurav Gupta

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



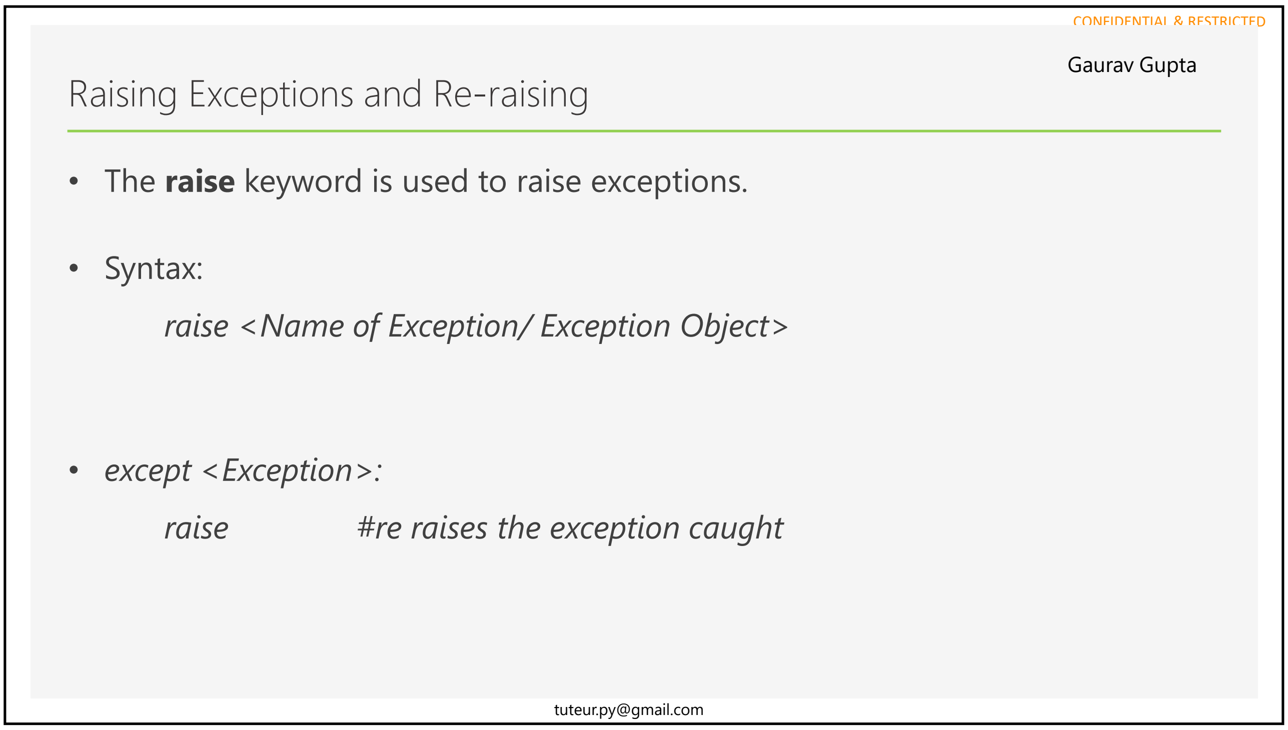
Gaurav Gupta

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

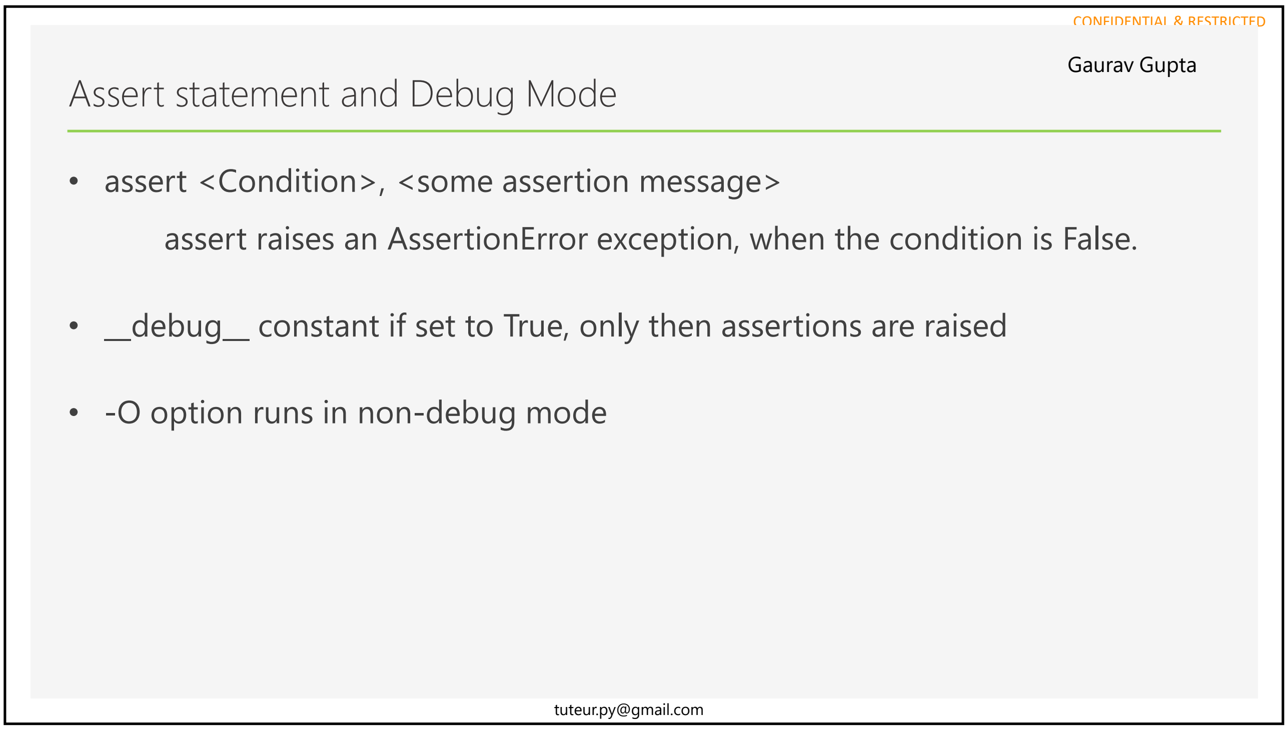
- The **raise** keyword is used to raise exceptions.
- Syntax:

  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>

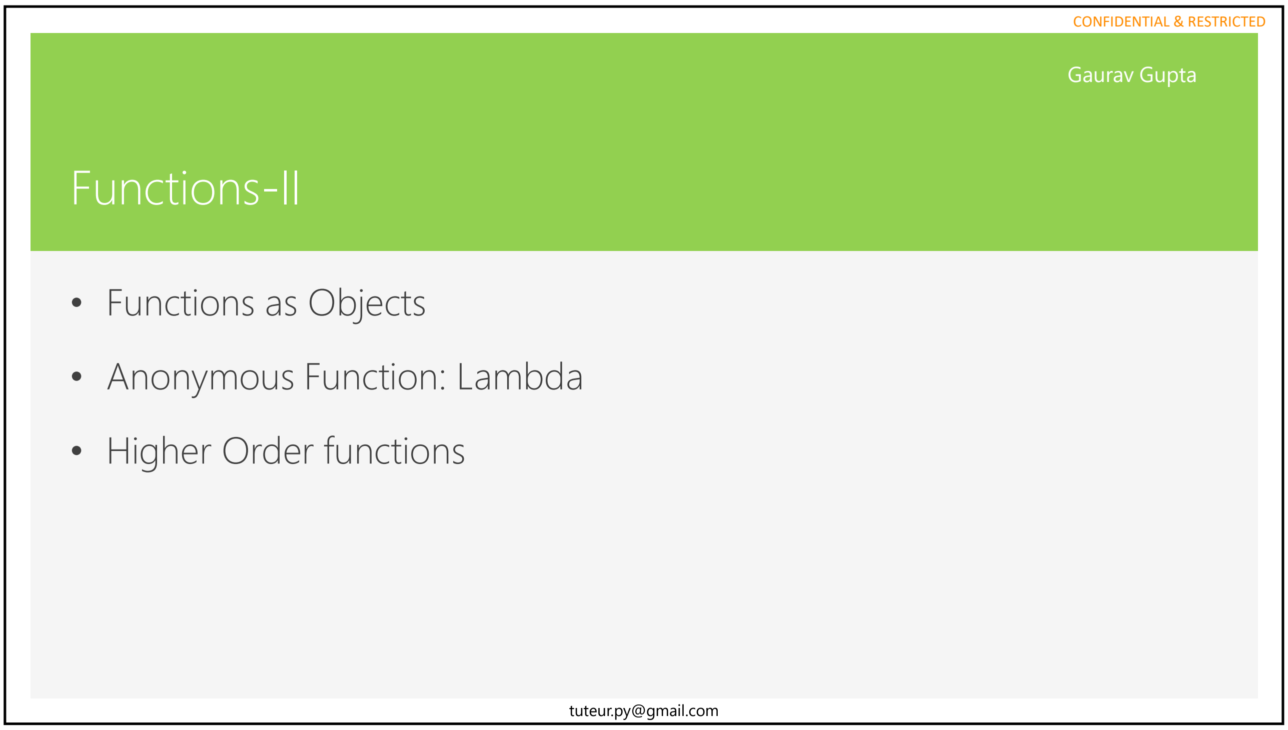
  assert raises an AssertionError exception, when the condition is False.

- __debug__ constant if set to True, only then assertions are raised

- -O option runs in non-debug mode

tuteur.py@gmail.com

129



Gaurav Gupta

Functions-II

- Functions as Objects
- Anonymous Function: Lambda
- Higher Order functions

tuteur.py@gmail.com

130

Gaurav Gupta

Before we Begin

- Introducing
  isinstance(<object>, <class-or-type-or-tuple containing types>) -> bool

- Return whether the **object** is an instance of a **class** or of a **subclass** or of the **type** as specified in the second argument.

- When using a tuple

  isinstance(x, (A, B, ...))   # is a shortcut for

  isinstance(x, A) **or** isinstance(x, B) **or** ...

tuteur.py@gmail.com

Gaurav Gupta

Functions are objects just like everything else

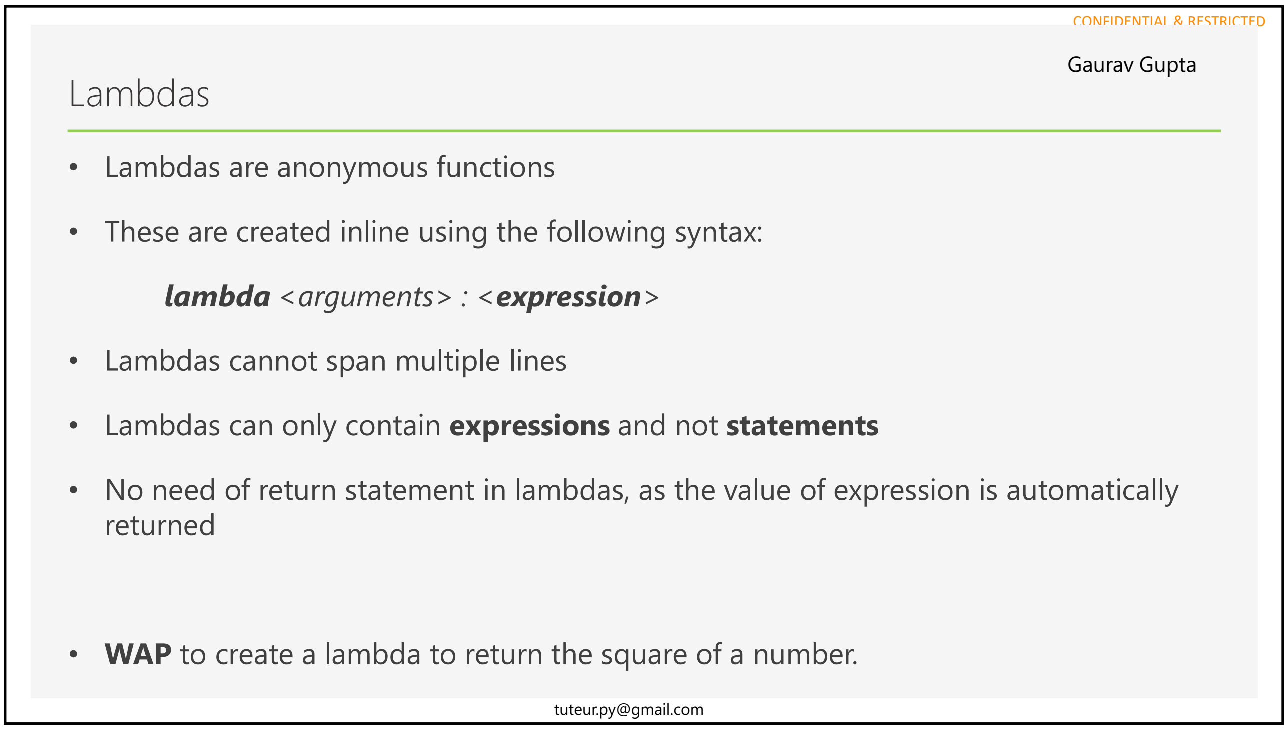
- Functions in python are **objects**.

- This means they can be **passed** to other functions and can be **stored** in a data structure like list, dict etc.

- Try to print the type of a function

- WAP to create a **calculator** using a **dictionary** of functions mapped to each operator

tuteur.py@gmail.com



Gaurav Gupta

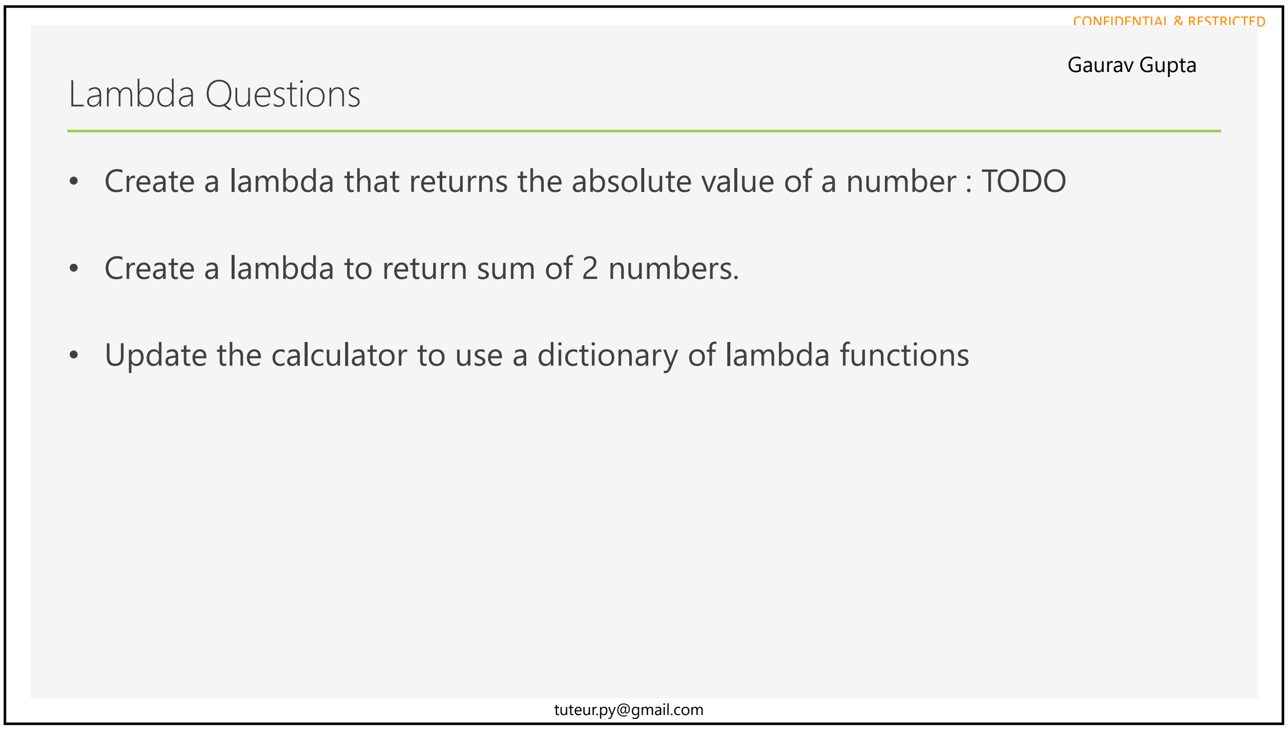
Lambdas

- Lambdas are anonymous functions
- These are created inline using the following syntax:

  lambda *<arguments> : <**expression**>*

- Lambdas cannot span multiple lines
- Lambdas can only contain **expressions** and not **statements**
- No need of return statement in lambdas, as the value of expression is automatically returned

- **WAP** to create a lambda to return the square of a number.



Gaurav Gupta

Lambda Questions

- Create a lambda that returns the absolute value of a number : TODO
- Create a lambda to return sum of 2 numbers.
- Update the calculator to use a dictionary of lambda functions

Gaurav Gupta

Higher Order Functions

- Functions that take functions as arguments or return functions are called higher order functions.
- **Map, reduce** and **filter** functions:

  map(<function to apply>, <list of inputs>)

  reduce(<function to apply>, <list of inputs>) # implement

  filter(<function to apply>, <list of inputs>) # implement

- *reduce* is available in **functools** module

tuteur.py@gmail.com

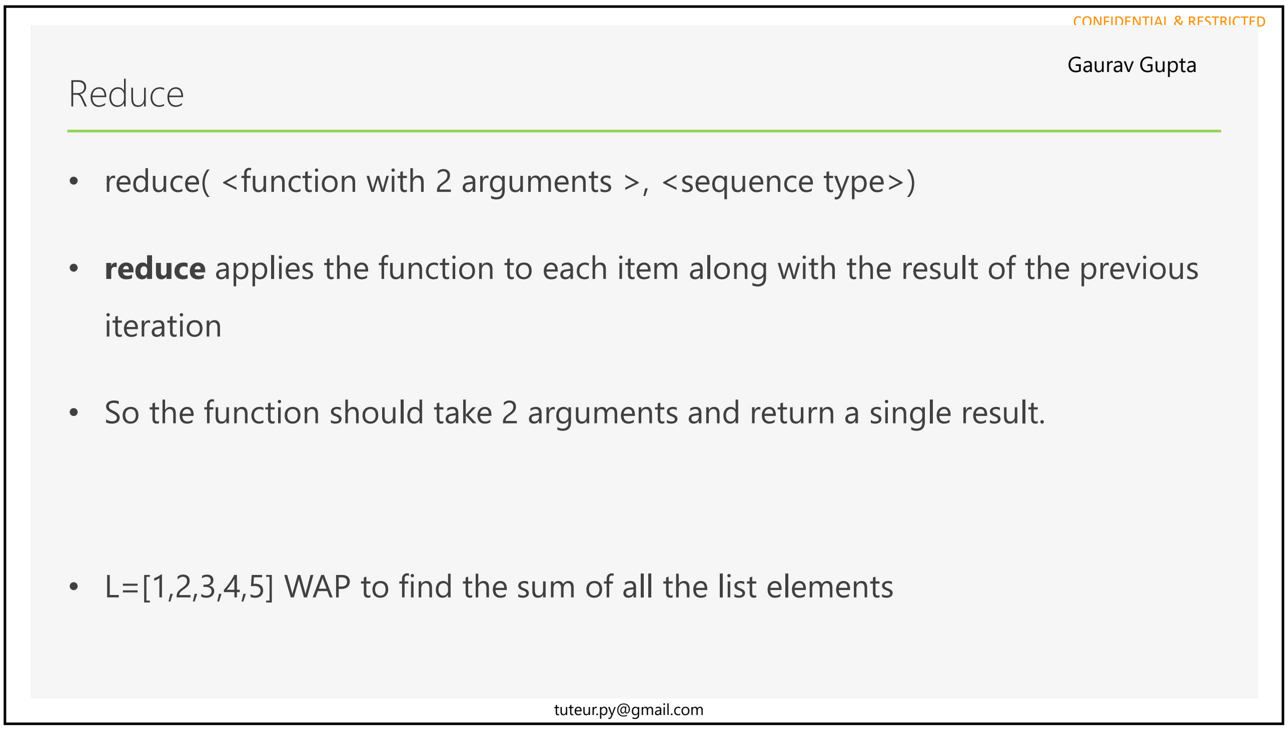
Gaurav Gupta

MAP

- Map applies the function to each item of the iterable and returns a sequence containing the result of corresponding values.

- L=[1,2,3,4,5] WAP to create a list of square of these numbers
- Replace all spaces with * in a string.

tuteur.py@gmail.com



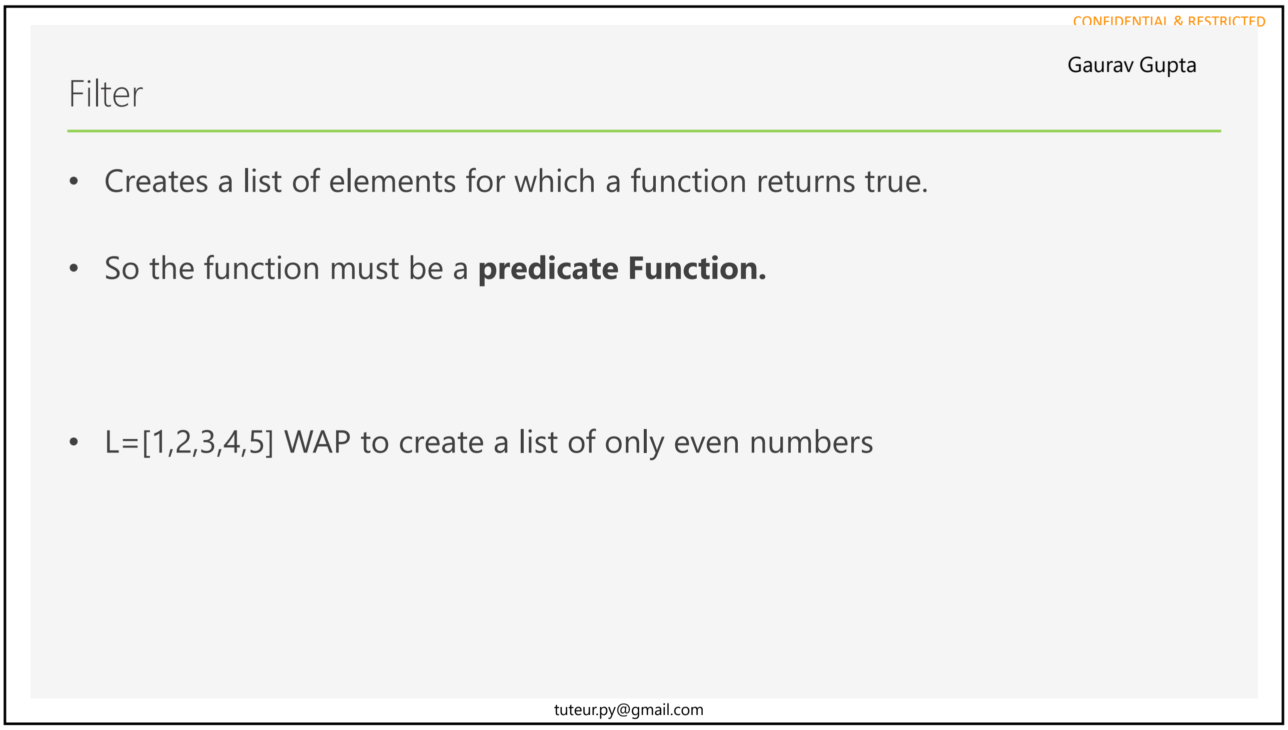
Gaurav Gupta

Reduce

- reduce(<function with 2 arguments >, <sequence type>)
- **reduce** applies the function to each item along with the result of the previous iteration
- So the function should take 2 arguments and return a single result.
- L=[1,2,3,4,5] WAP to find the sum of all the list elements

tuteur.py@gmail.com

137



Gaurav Gupta

Filter

- Creates a list of elements for which a function returns true.
- So the function must be a **predicate Function.**
- L=[1,2,3,4,5] WAP to create a list of only even numbers

tuteur.py@gmail.com

138

Gaurav Gupta

Predicate Function

- A function that takes an argument and returns the **true** or **false** (a Boolean value) as a result.
- The **lambda** passed to the **Filter** function used in the previous case is Even Numbers example is a Predicate function.

Gaurav Gupta

Sort method and lambdas

- Sorting a list of tuples containing name and age.

  [('Abhishek', '12'), ('Gaurav', 10), ('Rahul', '13'), ('Krishna', '11')]

- Sort complete syntax:

  <list object> . sort(key=<some function>, reverse=False)

  <some function> should be a function taking a single argument and returning a single value (*a good candidate for a lambda*).

Gaurav Gupta

Function and Scope

- The variable assignments done in a function create new objects that are local to the method
- Ex:

```
def method():
    a = 10 # local
    print(a)

method()
print(a) # gives error
```

```
a = 0 # global
def funct0():
    print(a)
def funct1():
    a = 100
    print(a)
```

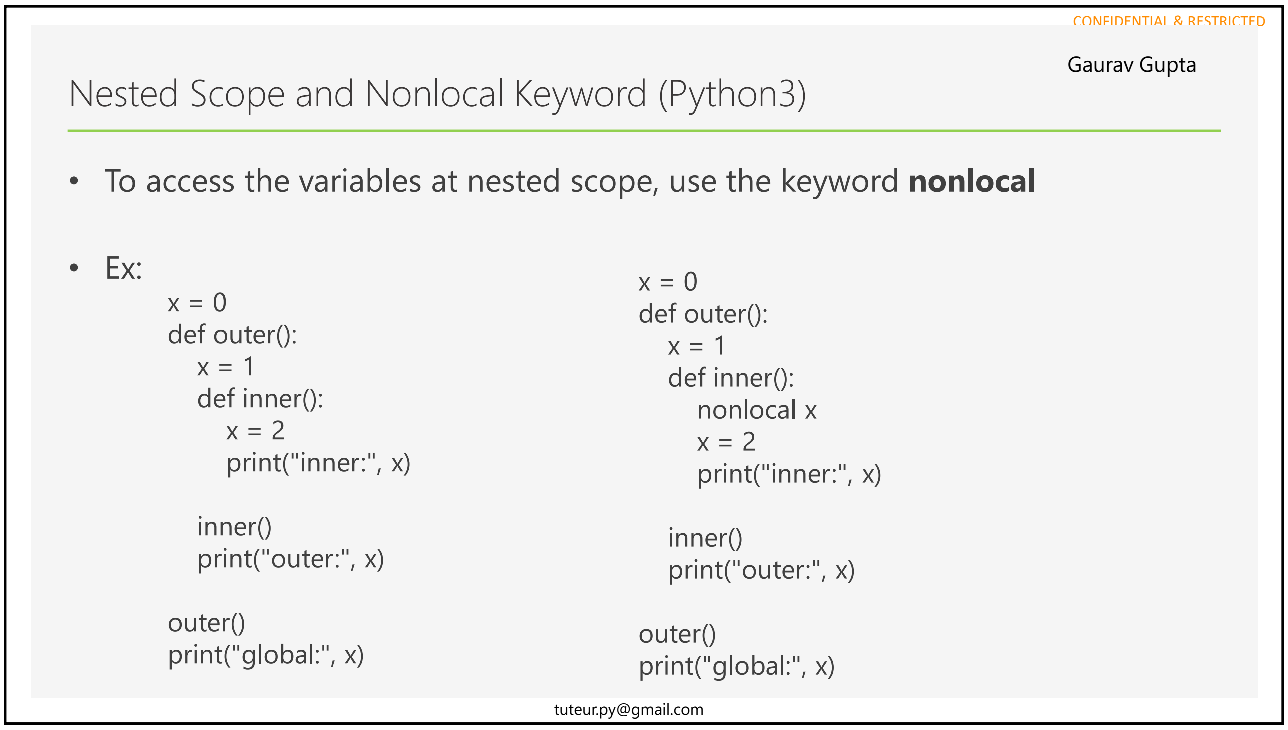
Gaurav Gupta

The Global Keyword

- To access the variables at global scope, use the keyword **global**
- Ex:

```
a = 0 # global variable
def funct():
    global a
    print(a)
    a = a+1
    print(a)

funct()
print(a)
```

```
# gives error; can't access local before declaring it
a = 0

def funct():
    print(a)
    a = 100
    print(a)

funct()
```



Gaurav Gupta

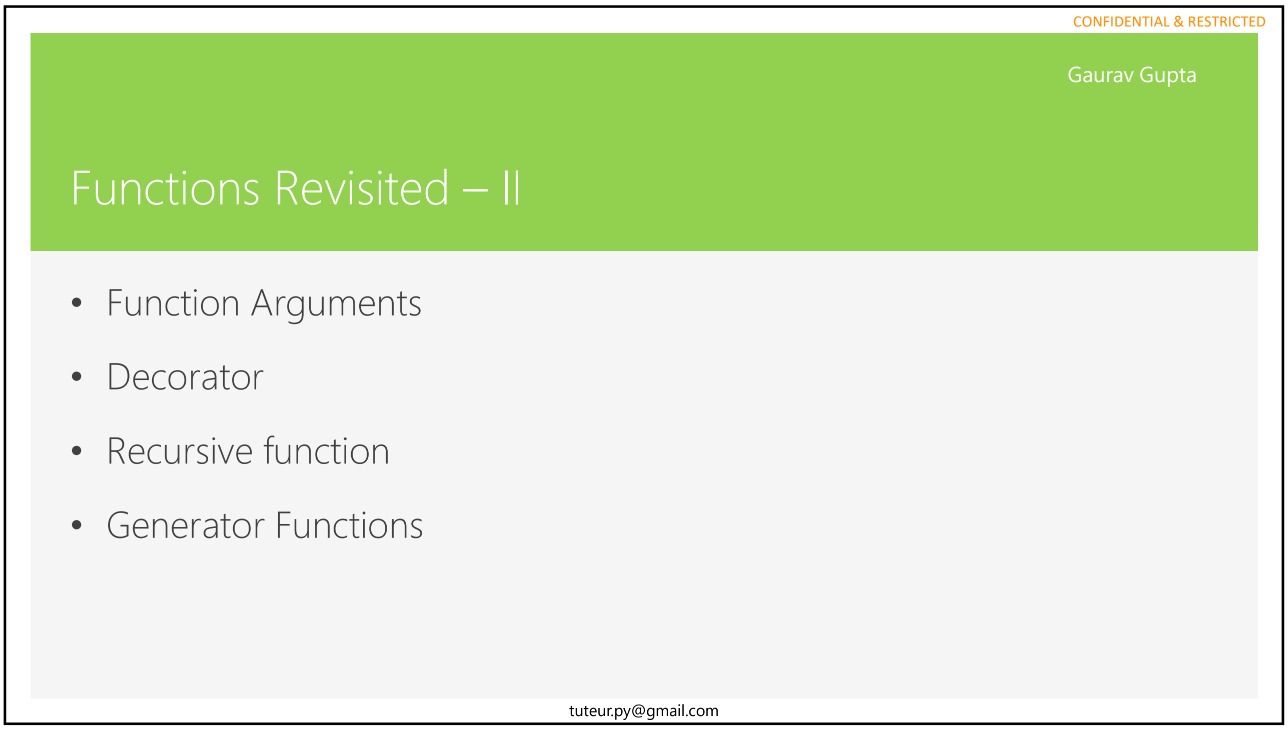
Nested Scope and Nonlocal Keyword (Python3)

- To access the variables at nested scope, use the keyword **nonlocal**
- Ex:

```
x = 0
def outer():
    x = 1
    def inner():
        x = 2
        print("inner:", x)

    inner()
    print("outer:", x)

outer()
print("global:", x)
```

```
x = 0
def outer():
    x = 1
    def inner():
        nonlocal x
        x = 2
        print("inner:", x)

    inner()
    print("outer:", x)

outer()
print("global:", x)
```

tuteur.py@gmail.com

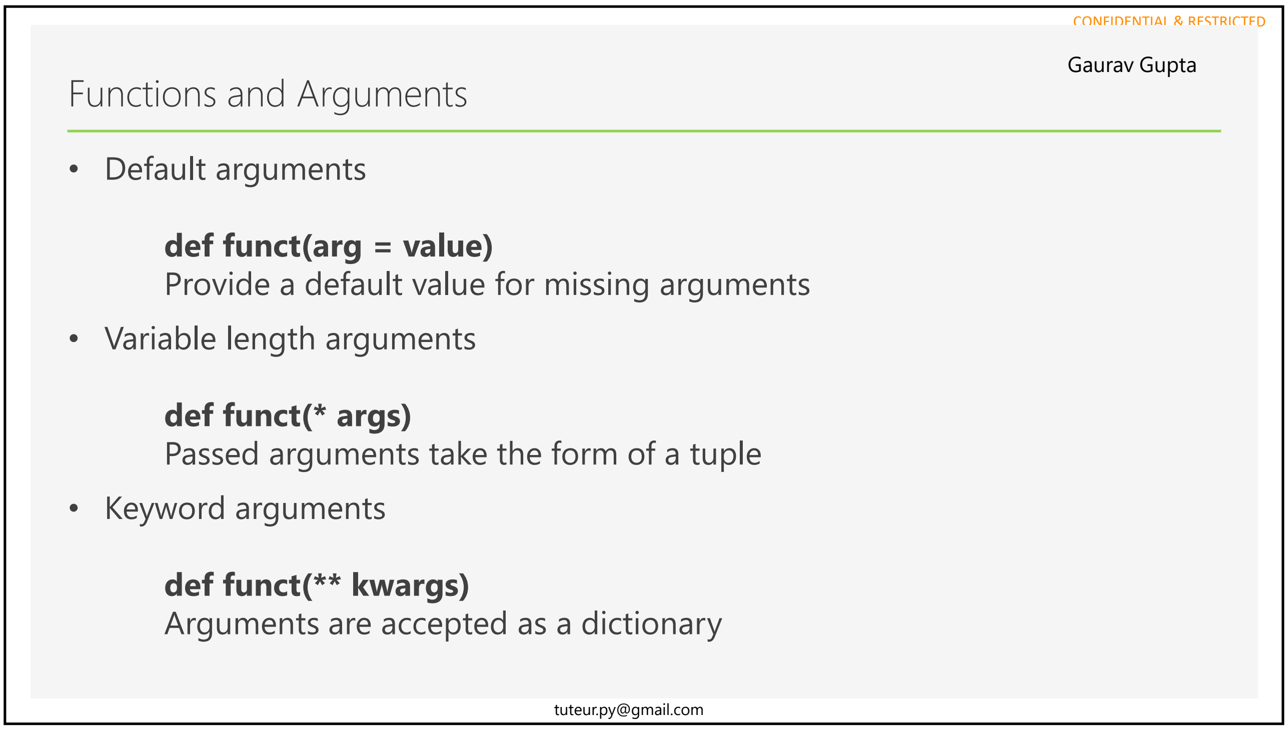


Gaurav Gupta

Functions Revisited – II

- Function Arguments
- Decorator
- Recursive function
- Generator Functions

tuteur.py@gmail.com



Gaurav Gupta

Functions and Arguments

- Default arguments

  def funct(arg = value)
  Provide a default value for missing arguments
- Variable length arguments

  def funct(* args)
  Passed arguments take the form of a tuple
- Keyword arguments

  def funct(kwargs)**
  Arguments are accepted as a dictionary

tuteur.py@gmail.com