Welcome to Python

Gaurav Gupta
tuteur.py@gmail.com

Gaurav Gupta

Setup and Workspace

- Installing Python
- Etherpad
- Testing Installation : The Interactive shell
- Tools for working environment
- Creating workspace: Directory structure
- Windows and Linux Command line
- Some shortcut keys

tuteur.py@gmail.com

Installing Python

Python available at the official website : https://www.python.org/

- Windows : Download the executable and run it.
- Linux : Run the command on Ubuntu shell

  sudo apt-get install python3

Gaurav Gupta

Oh!! Did I Introduce you to Etherpad

Etherpad is a shared notepad available at the following link.

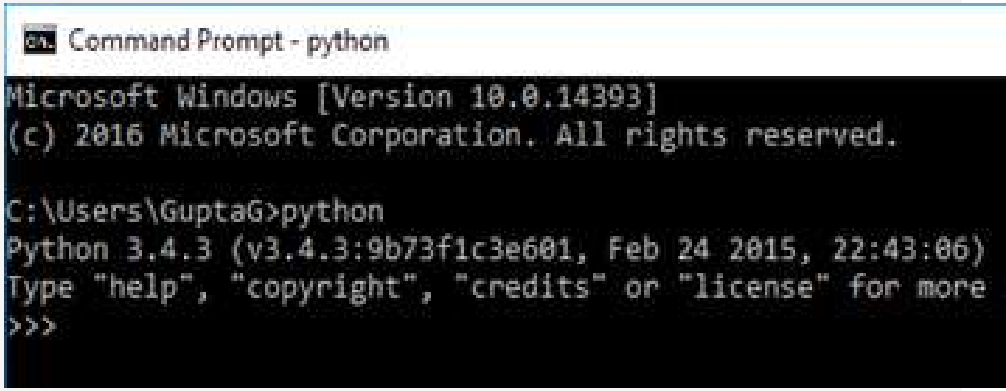
https://etherpad.net/p/py_learnbay

Consider it as your friend, you get to know why soon.

Gaurav Gupta

Testing Installation : The Interactive shell

- Windows : Press Windows Key and type cmd. On the Terminal type python



- Linux : Open a terminal (Ubuntu CTRL+ALT+T) and type python.

** if you get error like command not found, add python installation path

Gaurav Gupta

Tools for working environment

- Use an IDE

  Pycharm IDE with Python 3.x.x

  https://www.jetbrains.com/pycharm/download/

- Use any text editor and Command line (my preferred way)

  Write Scripts using a text editor : Notepad++, vi, vim, Sublime Text..

  Windows or Linux command line for executing.

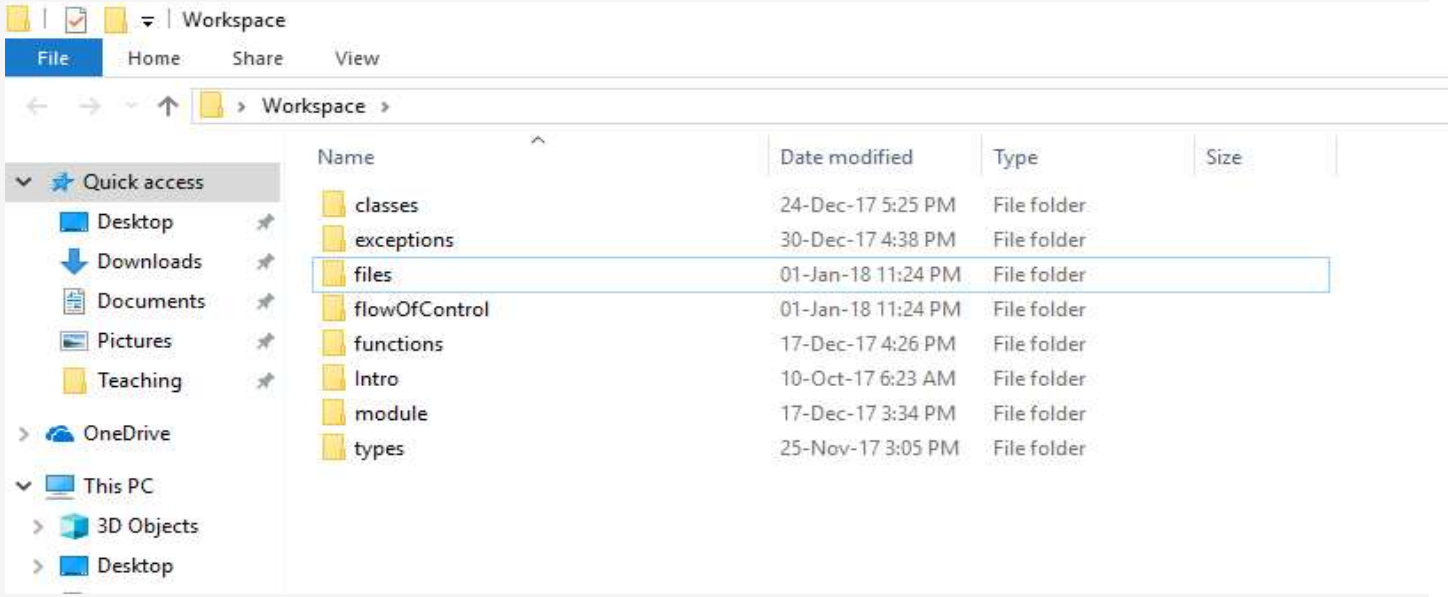
Gaurav Gupta

Creating workspace: Directory structure

- Create a folder ***workspace*** : all our scripts will be in this folder
- Maintain separate folders for each topic in ***workspace*** folder.
- Make sure to name the script files in following convention: f**N_topic**.py

Ex:

f1_ifStatement.py
f2_ifElse.py



Gaurav Gupta

Windows and Linux Command line

	Windows	Linux
Go to the folder	*cd <folder Name>* *Ex:* *cd Workspace*	*cd <folder Name>* *Ex:* *cd Workspace*
Go to the previous directory	*cd ..*	*cd ..*
List files in current directory	*dir*	*ls* *ls –la*

Use up and down arrow keys to view previous commands in cmd window

Gaurav Gupta

Notepad++ Shortcuts

Ctrl + a	To select everything in current file
Ctrl + s	Save current file
Ctrl + Tab	To switch files
Ctrl + n	To open new file
Ctrl + c	To copy selected text
Ctrl + v	To paste selected text

- Press *Shift* and Arrow keys to make selection of a part of text (you can use Ctrl key while selecting to make selection faster)

Gaurav Gupta

Windows shortcuts

Open command window in current folder

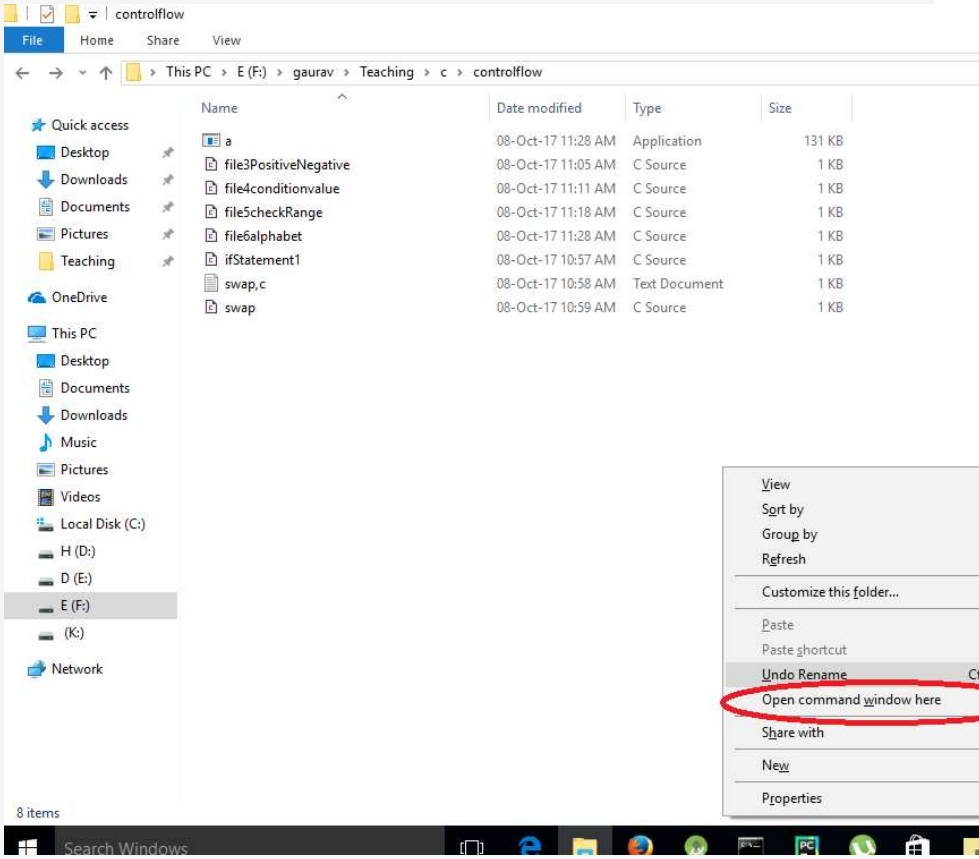
Press the Shift Key and right click

You will see the option :

Open Command Window Here

To Switch Tabs/Windows

Alt + Tab and Alt + Shift + Tab



tuteur.py@gmail.com

Gaurav Gupta

Python Kickstart

- Using Interpreter and a Script
- Intro to print function
- Dir and help functions

tuteur.py@gmail.com

Gaurav Gupta

Using Interpreter

- Open cmd window and type:

  1 + 2

- Create a python script and type the same thing there.

  Save at **f1.py**

- Now run from the command line as:

  python f1.py ***#before doing this just check version of python***

Gaurav Gupta

Intro to print function

- In a python script type:

  print(1 + 2)

  now save it and run again.

- Now try working with variables.
- Printing multiple values from single print function
- *And yes PRINT IS A* **FUNCTION**

tuteur.py@gmail.com

Gaurav Gupta

Creating a Variable: Dir And Help functions

- Create a variable in the current scope and check what all things are available there
- **Dir** gives the list of available attributes and objects in the current scope or of the object if passed and argument.
- **Help** method returns help information, depending on how it is invoked.
- Help can be called without argument, with the names of builtins, or with names specified as a string

tuteur.py@gmail.com

Gaurav Gupta

Python Syntax ,Keywords and Operators

- Tokens : building blocks
- Python Comments
- Print Method
- Input()
- Type() and basic types in python
- Conversion Between Types

tuteur.py@gmail.com

Gaurav Gupta

Tokens : building blocks

- Smallest individual components that make up a program.
- 4 Types :
  - Keywords
  - Identifiers
  - Operators
  - Literals

Gaurav Gupta

Keywords

- Special reserved words predefined or reserved by the language.

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

Gaurav Gupta

Identifiers

- **Identifiers** can be a combination of letters in lowercase (**a to z**) or uppercase (**A to Z**) or digits (**0 to 9**) or an underscore (_)
- Variable names, class names, function names and module names are all identifiers.
- Some special identifiers in Python :

  __*__ : Special Reserved system defined names

  __* : Used to define private class members

Gaurav Gupta

Operators

- + ,- ,*, /, >, <, =, <=, >=, ==, !=, >>, <<, &, |, ~, ^
- +=, -=, *=, /=, =
- (,),[,],{,}

tuteur.py@gmail.com

Gaurav Gupta

Literals

These are just constant values:

integer	:	1,-1,0….
Floating	:	-1.0, 0.0, 3.14
string	:	‘’, ‘ ’, ‘a’, ‘abcd’
Boolean	:	True, False

Gaurav Gupta

String Dilemma

- Single, Double or Triple Quotes??
- ‘Quoted String’ “Quoted String” ”””Quoted String””” ‘’’Quoted String’’’
- Single quote can be used in double quoted string and vice versa:

  ‘ single ‘ in single ‘ ; **“double “ in double”** : Wrong

  ‘ double “ in single’ ; **“single ‘ in double”** : Right

- **””” Multi Line string”””**

Gaurav Gupta

Comments

- **Single line** comments start with #.

```
# This is a single line comment in python
```

- **Multi line** comments can use the triple quote syntax.

```
"""
This is a multi line
comment in python.
"""
```

Gaurav Gupta

Print Function

- Print method prints to the standard output
- Syntax:

  *print(<var/const>, ..., **sep**= '<separator>', **end** = '<delimiter>', **file** = <file object>)*

  ***sep, file** and **end**, arguments are optional and should appear in the end.*

- *Escape Sequences : **\n** and **\t***

Gaurav Gupta

Type Method

- Syntax:

  type(<object argument>)

- Returns the type of the argument

- Argument might be variables, objects ….

- Some basic types are:

  int, float, string, bool, complex

Gaurav Gupta

Converting Between types

- int(<string>), int(<int>), int(<float>) # converts string containing digits to int
- str(<int/float/....>) # converts any type to its string representation

tuteur.py@gmail.com

Gaurav Gupta

Input()

- The input method returns the value entered by user as a string
- Also allows to specify a string argument for a message to displayed

```
1  x = input('Enter one Number')
2  x = int(x)
3  y = x*x
4  print("Square of " + str(x) + " is %d" % y )
```

tuteur.py@gmail.com

Gaurav Gupta

Conversion Between Types

- String to **Int** : int(<string variable/constant>)
- String to **float** : float(<string variable /constant >)
- Any Type to **String** : str(<variable /constant >)
- **bin()** method returns the binary representation of an **integer**

Gaurav Gupta

Data Types and Operations

- Numeric types
- Boolean types
- Strings
- None types

tuteur.py@gmail.com

Gaurav Gupta

Numeric 2+2.5 = 4.5

- int, float, complex types
- Operations
  Relational : >, >=, <, <=, ==, !=
  Arithmetic : +, -, *, **, /, //, %
  Bit Operation: |, ^, &, <<, >>, ~
- ****** - power; -4**2 and (-4)**2 WAP to input X and Y and find x^y
- **//** - int division; -10//3 and 10//3
- **%** - modulus; 10%3, 10%-3

tuteur.py@gmail.com

Gaurav Gupta

Boolean

- Only **True** and **False** values
- **True** and **False** are singleton objects
- **True** and **False** map to integers **1** and **0** respectively
- Any number other than **0** is treated as **True.**

- Test the outputs of the following commands on the prompt or in a script:

```
print(bool(0));        print(bool(10));     print(bool(-1))
print(int(True));      print(int(False))
```

Gaurav Gupta

Str '2'+'2.5' = '22.5'

- Strings are **immutable sequence** of characters
- Ex:

  ' simple string'

  "double quotes"

  """ triple quotes"""

tuteur.py@gmail.com

Gaurav Gupta

None type

- **None** represents null or empty
- Often returned by some methods, to mark no return value.

tuteur.py@gmail.com

Gaurav Gupta

Ascii Values and ORD

- All characters are represented by a numeric value in ASCII encoding
- A – 65
- a – 97
- ord() function returns the ascii value of a character

tuteur.py@gmail.com

Gaurav Gupta

Importing

- Importing Syntax
- Random Module
- Simulating Dice Roll
- Practice

tuteur.py@gmail.com

Gaurav Gupta

Importing Modules : Import statement

- import <module name> **# import the entire module**

  import cmath

  cmath.sqrt(-1)

- from <module name> import * **# import all components from module**

  *from cmath import **

  sqrt(-1)

- from <module name> import <class/function> **# import selected component from module**

  from cmath import sqrt

  sqrt(-1)

Gaurav Gupta

Random Library

- import random module using:

  import random

- Random Integers :

  randrange(end) **0 <= N <= end – 1**

  randrange(100)

  randint(start, end) **start <= N <= end**

  randint(1,10)

  randrange(start, end, [step]) **one from start, start+step, start + step*2..**

  randrange(10,20,2)

Gaurav Gupta

Random Library

- Random Floats:

  random()

  Floating number [0.0, 1.0) or 0.0 <= N < 1.0

  uniform(start, end)

  start <= N <= end

  uniform(11,44.5)

tuteur.py@gmail.com

Practice

- Build a library my_lib.py add a few variables to test.
- Add functions to input data.
- Add the library to the python search path.

Gaurav Gupta

Some Pythonic Humor

- Will there ever be braces in python (__future__ braces)
- Writing hello word is that simple __hello__
- The Zen of Python (import this)
- antigravity

tuteur.py@gmail.com

Gaurav Gupta

Functions

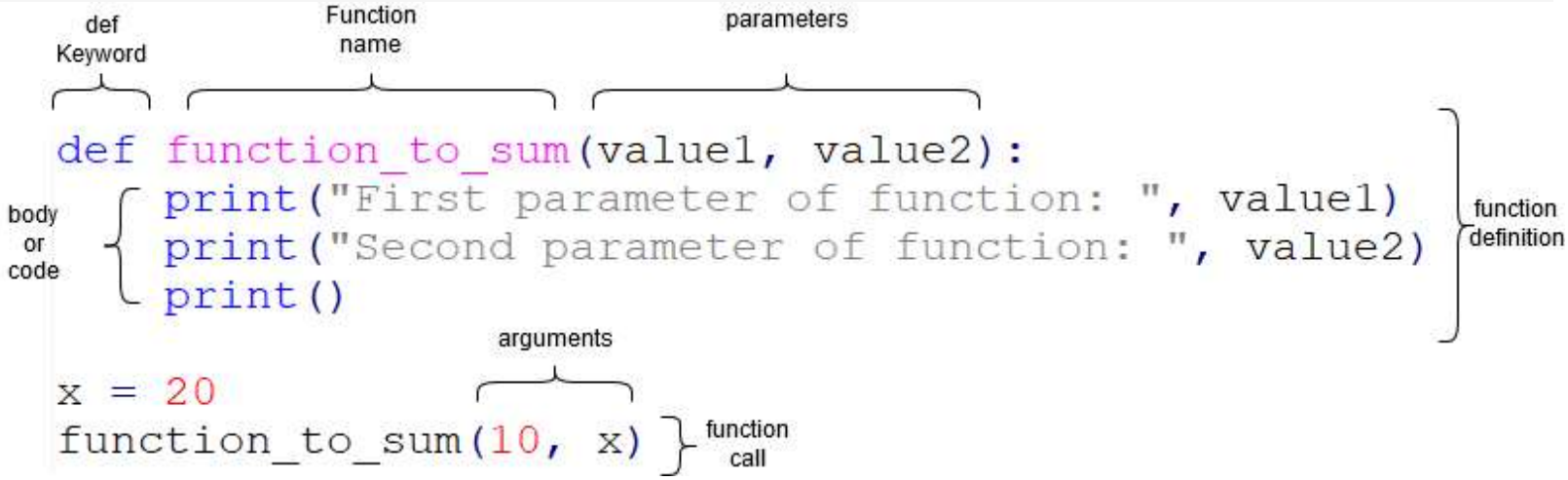
- Function definition and call
- Arguments
- Returning from function
- Arguments
- Creating a module

tuteur.py@gmail.com

Gaurav Gupta

Function Terminology

- **Parameter:** the variables specified in the bracket of a function definition
- **Return value:** the value or variable written after **return** keyword in a function
- **Definition** the code written along with the def statement.
- **Argument** the value passed to a function at *function call*.
- **Function Call** the name of the function along with the arguments if any.



Gaurav Gupta

Creating Functions

- Syntax:

```
def <function name>(arguments):
        """     optional doc string          """
        # body/logic/code of function
```

- ***Def*** keyword is used to start a function
- Function may or may not **return** a **value**; depends on the use of ***return*** keyword
- Function gets executed only when it is **called/invoked**

- WAF that **inputs** temperature in Celsius and **Prints** it in Fahrenheit

Gaurav Gupta

Function Arguments

- Remember the **randrange** function which takes the max value as argument.

  random.randrange(100) # generates number between 0 and 99

- Arguments are a way of passing or giving input values to a function

- WAF (Write a Function) that takes temperature in Celsius as **argument** and **Prints** the temperature in Fahrenheit.
- Update the above method to test the validity of the **type** of argument (it should be **float** or **int** only).

Gaurav Gupta

Returning values

- The **randrange** method returns or gives us the generated value, instead of printing it on the screen.

```
num = random.randrange(100)  # the result gets stored in num
```

- Python uses the **return statement** to returns results/values from function
- The function **terminates** once a return statement executes and control passes to the calling function.
- Multiple values can also be returned in form of tuples, dictionaries…

- WAF (Write a Function) that takes temperature in Celsius as **argument** and **returns** the temperature in Fahrenheit.

Gaurav Gupta

Default Arguments

- Some arguments may have a default value.
- i.e. If while calling the value for that argument is not given, then the default value specified in function definition is taken automatically.

Gaurav Gupta

Creating a Module

- Any script created in python is a module and can be imported in other scripts/modules in python.
- Python looks for modules in the current working directory apart from the pythons' default search locations.
- The variable sys.path lists all the locations which are searched.
- Use the environment variable **PYTHONPATH** to add paths to modules other than current working directory.

Gaurav Gupta

Back to Strings

- String Functions
- Indexing and Slicing
- String Formatting

tuteur.py@gmail.com

Gaurav Gupta

String Functions

- len() : len(<string object>) # return length of the string
- upper() : **<string object>.upper()** # returns in upper case
- lower()
- isdigit() isalpha() isspace() isalnum()
  islower() isupper()

tuteur.py@gmail.com

Gaurav Gupta

Slicing and Indexing

- Indexing:

  <string>[<integer index>]

- Slicing:

  <string>[start : end]

  <string>[start : end : step]

- Start and end decide the end and start point in string

* Indexes start from 0 and end at (length – 1) [Think how to get the length]

Gaurav Gupta

More Methods

- count() : **# counts occurrence of a string in other**
  <string object>.count(<search string>, [start, [end]])
- find() : **# finds index of first occurrence, else returns -1**
  <string object>.find(<search string>, [start, [end]])
- in : **# membership check; this is a keyword not a function**
  <string object> in <other string object>

Gaurav Gupta

Even more functions

- replace() : # replaces all occurrence of **old** with **new count** no of times

  <string object>.replace(old , new [, count])

- split() : # splits a *string object* in multiple strings, using the *split string*

  <string object>.split(<split string> = ' ')

- join() : # joins the *list of strings* using the *join string*

  <joining string>.join(<list of strings>)

Gaurav Gupta

Formatting strings

- " some format string goes in here" % (a tuple of values)
- %s = string
- %d = integer
- %f = float

Gaurav Gupta

Sequence Type List

- List Creation
- List Mutability
- Operations
- Slicing

tuteur.py@gmail.com

Gaurav Gupta

List

- **[1,2,3, True, ‘abcd’]**
- **Mutable Sequence** type with elements separated by a comma.

```
l1 = []

l2 = list()

l3 = [1,2,3]

l4 = list(l3)

l5 = list(‘string’)
```

Gaurav Gupta

List

- **Mutability**

  l [1] = 4

  l. append(5)

  l. insert(2,33)

  l. extend([10 ,20])

  len(l)

- **WAP** to input a sentence from user , and print one random word out of it.

tuteur.py@gmail.com

Gaurav Gupta

List Functions

- **In Place** operations

  l.sort()

  l.index()

  l.pop()

  l.remove()

- Indexing:
  l = [[10, 20], [True, False], [], 'abcd']
  l [0] [1]
  l [3] [3]

Gaurav Gupta

Sequence Type Tupe

- Tuple Creation
- Immutability
- Operations
- Slicing

tuteur.py@gmail.com

Gaurav Gupta

Tuple

(1 ,2.3 , True, 'ABCD')

- **Immutable** sequences. Represented by a **()**
- x = ()

  x = tuple()

  x = (1,2,3)

  x = 1,2,3

  x = 1,

  x = tuple([1,2,3])

Gaurav Gupta

Tuple

Modifications not allowed

```
x = (1, 2, 3)
x[1] = 3
```

Gaurav Gupta

Copying Lists

- Simple assignments don't create copy

```
l2 = l1 # both are same
```

- Copying requires special call to **list()** or **slicing**

```
l2 = list( l1 )
l2 = l1 [:]
l2 = l2 [::]
```

Gaurav Gupta

Common operations on Sequences

- **len()** : returns the number of elements
- Slicing.
- Membership check

  in , **not in** # returns Boolean **True** or **False**

- Finding minimum and maximum values:

  min, **max**

- Concatenation and Replication

  +, ***

Gaurav Gupta

Loops

- While Loop
- Break and continue
- List Comprehension

tuteur.py@gmail.com

Gaurav Gupta

While Loop

- Syntax :

```
while <condition>:
        statements1
else:           # optional
        statements2
```

- *Statements2* is executed when condition becomes false (but not in case of break)

- WAP to print first 10 natural numbers. Update the program to print their sum
- WAP to count vowels in a string input by user.
- WAP to print all multiples of **3** till **N** (input N from user).

Gaurav Gupta

Break and Continue

- **break** statement is used to terminate the current loop
- On execution, **continue** statement skips the statements below it in the current loop and forces next iteration of the loop.

- Update the **rolling dice** program to ask user to roll again or exit(break).
- Update the **rolling dice** program to also check for invalid inputs(continue)

tuteur.py@gmail.com

Gaurav Gupta

Iterating Sequences Python way

- Simple For loop
- Range based for loop

tuteur.py@gmail.com

Gaurav Gupta

For loop

- Use **for** loop:

  for *<variable>* in *<sequence type>*:

  # operations using **<variable>**

- Printing a List

  Print Square of elements

  Print length of words in sentence

  Sum elements in a list

  Input a sequence of number separated by spaces and convert it into a list of numbers

tuteur.py@gmail.com

Gaurav Gupta

Range

- Represents **immutable sequence** of numbers.
- **range()** method returns a **range object** in python 3
  range(start [,end [, step size]])
- Employed in range based for loops
- Ex:

```
range(10)          # returns object with values 0 till 9
range(5,10)        # 5 till 9
range(20,100, 5)   # 20 till 95 with step size of 5
```

Gaurav Gupta

Practice

- Print Whole numbers till N
- Sum numbers till N
- Print Square of numbers till N
- WAP to print 5 random numbers
- WAP to put 5 random numbers in a list

Gaurav Gupta

List Comprehension : For loop

- Syntax:
  [expression(<*variable*>) **for** <*variable*> **in** <*sequence type*> *[if <condition>]*]
  condition is optional

- WAP to generate list of first 10 natural numbers (Generate a list of their squares also).
- WAP to count vowels using list comprehension
- WAP to find sum of the squares of first 10 even numbers
  4 + 9 + 16 + 25 ….

tuteur.py@gmail.com

Gaurav Gupta

Decision Statements

- Statement vs Expression
- Relational Operators
- Logical Operators
- If statement and its variants
- Nesting of statements

tuteur.py@gmail.com

Gaurav Gupta

Statement vs Expression

- **Expression** is something that evaluates to a value
- **Statement** is any line of code that can be executed by the python interpreter.
- Since expressions evaluate to value, so they can appear on the **rhs** of an **assignment** operator **(=)**.

tuteur.py@gmail.com

Gaurav Gupta

Relational Operators

- These operators return **True** or **False** depending on truth or false value of the relation

  Operators:

  >, <, >=, <=, ==, !=

Gaurav Gupta

Logical Operators

- These operators evaluate **Truth** and **False** values and return **True** or **False** depending logic of the operator

  3 logical Operators:

  and, or, not

- **and** and **or** are *binary* operator, whereas **not** is a *unary* operator

tuteur.py@gmail.com

Gaurav Gupta

Truth Table: and, or, not

X	Y	X and Y
False	False	False
False	True	False
True	False	False
True	True	True

X	Y	X or Y
False	False	False
False	True	True
True	False	True
True	True	True

X	not X
False	True
True	False

tuteur.py@gmail.com

Gaurav Gupta

Test

- x = 2
  y = x>1 and x < 100
  print(y)

- x = 2
  y = x>1 or x < 100
  print(y)

- x = 2
  y = x>1
  print(y)
  y = not y
  print(y)

- x = -100
  y = x>1 and x < 100
  print(y)

- x = -10
  y = x>1 or x < 100
  print(y)

tuteur.py@gmail.com

Gaurav Gupta

Simple If Statement

- if condition_1:
  statement_block_1 # notice the indentation (spacing) before the block

- The code referred to as statement_block_1 gets executed only if the condition evaluates to true else gets skipped.

- WAP to print absolute value of a number

Gaurav Gupta

Simple If-else Statement

- if condition_1:
  statement_block_1
  else:
  statement_block_2

- The code referred to as **statement_block_1** gets executed only ***if*** the condition evaluates to true ***else*** **statement_block_2** gets executed.
- WAP to input 2 number and print the larger one
- WAP to print whether number is even or odd
- WAP to check if a string is **palindrome** or not (**naman** is palindrome, **gaurav** is not)

Gaurav Gupta

if-elif-else Statement

- if *condition_1*:
  statement_block_1
  elif *condition_2*:
  statement_block_2
  …
  …
  else: # optional
  statement_block_n

- WAP to check if no is positive, negative or zero.
- WAP to create a 4 function calculator. (also update to use functions)

Gaurav Gupta

if-elif-else Statement

- WAP to input age and print the respective text depending on the age ranges as present in the table.

Age	Text To display
0-12	Child
13-17	Teen
18-50	Adult
51-100	Senior Citizen
age > 100	All the Best

Gaurav Gupta

Nested if-else statements

- if *condition_1*:
  if condition_2:
  block_1
  else:
  block_2
  elif …
  …
  …

- When a **if** block appears within another if block (can be inside **elif** or **else** or both), the inner block is said to be nested inside the outer block.

Gaurav Gupta

Test

- WAP to input 2 numbers. And do operation depending on the following:
    1. if any of the numbers is negative:
        a. if both are odd, add them
        b. otherwise, subtract them
    2. otherwise:
        a. if both are odd, multiply
        b. if one of them is odd, divide
        c. otherwise, find remainder

- WAP to input 2 numbers and check whether the first is divisible by the second and print true or false depending on the divisibility.

- WAP to print the value of the largest of 3 numbers taken as input from the user.

Gaurav Gupta

Mapping Type : Dict

- Dictionary
- Operations
- Programs

tuteur.py@gmail.com

Gaurav Gupta

Mapping : dict

- Mutable mapping type. Represented using {}

 # Creation

```
d = {}                          # empty dictionary
d = dict()                      # empty dictionary
d = dict(one=1, two=2, three=3)
d = {'one': 1, 'two': 2, 'three': 3}
d = dict([('two', 2), ('one', 1), ('three', 3)])  # list of tuples
```

 # Operations

d[<Key>] to access a value. Exception if key not found.

d[<Key>] = <Value> creates or overwrites **Value** for a **Key**

Gaurav Gupta

Dict : Operations

del *d[key]* # delete the entry for **Key**
pop(key [, default]) # deletes and returns value, **exception** if **key** not found and **Default** not provided
key **in** <*d*> # checks for membership of key in dictionary **d**
key **not in** <*d*>

Accessing elements
get(key, [default_value]**)** # returns key corresponding to the value. If key does not exist, returns None. If default value is specified, returns default value instead of None
items() # returns list of tuples of form **(key, value)**
keys() # returns list of keys
values() # returns list of values

tuteur.py@gmail.com

Gaurav Gupta

Question

Dictionary

_ Create a mapping of number to word from 0-9. **(0:'zero'……)**

_ Ask user for a single digit number and print the corresponding word format

_ Print all keys of a dictionary

_ Print all Values of a dictionary

_ Print all Key and Values of a dictionary

tuteur.py@gmail.com

Questions

- WAP to input a string from user and count occurrence of each alphabet in the string (Hint: use dictionaries). Upper and lower case alphabets are the same

  ex: sunny DaY

  s:1 u:1 n:2 y:2 d:1 a:1

Gaurav Gupta

Classes

- Class Syntax
- Writing simple class
- Creating objects
- Class vs Object

tuteur.py@gmail.com

Gaurav Gupta

Class Syntax

```
class <YourClassName>:
        # methods of class after one level of indentation
```

- Class names are **identifiers**.
- Class is a way of binding data and operations.
- This however looks different in python.

tuteur.py@gmail.com

Gaurav Gupta

What to do once you have a class

- Classes are generally meant for object/ instance creation.
- Instances are created in python, using class name and function call operator.
- While creating objects, there may be some arguments, just like functions.
- Syntax of creating an Object:

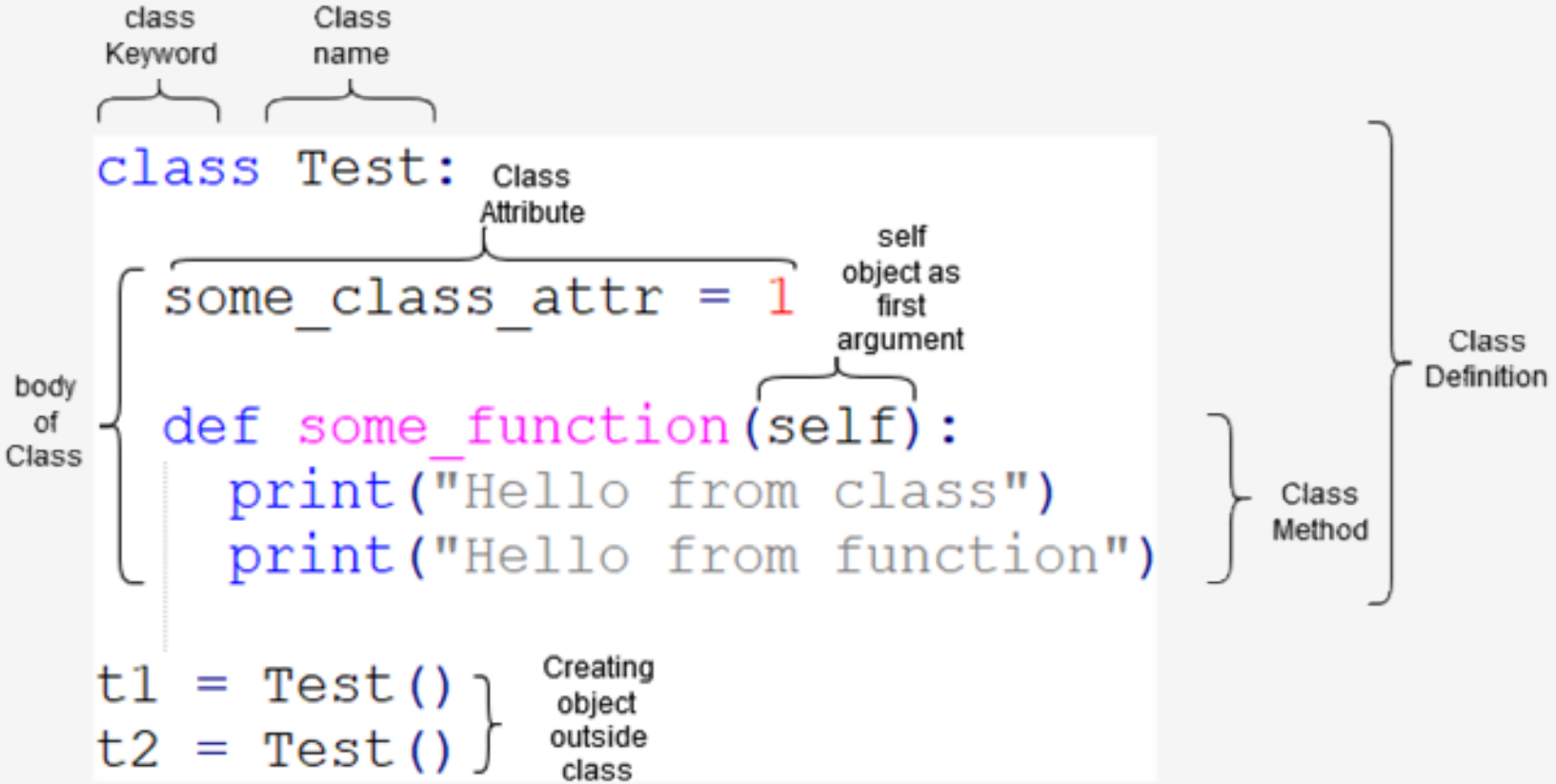
  <object_name> = <class_name>(zero or more arguments)

- **Example:**

```
s = str()          # creates without argument
s = str(100)       # give one argument: another list object
```

tuteur.py@gmail.com

Gaurav Gupta



tuteur.py@gmail.com

Gaurav Gupta

Adding Attributes

- Attributes refer to the data available or attached to an instance/object of a class.
- The attributes of an object are accessed using the **dot (.)** notation in python
- *Create a class **Contact**, with attributes : **name, phone and email**.*

```
p = Person()
p.name          # access the attribute name in p
```

Gaurav Gupta

Constructor and Destructor

- Constructor and Destructor are special methods in OOP, used for managing objects creation and destruction.
- Constructor defines some block of code that should get executed when any new instance of a class is created or whenever an object is instantiated.
- Destructor is the opposite of constructor and executes when object is destroyed.

tuteur.py@gmail.com

Gaurav Gupta

Constructor and __init__ method

- In python work of constructor is done by special **__init__** method.
- It takes a **self** argument, apart from other arguments, which is a reference to the newly created object.

```
class <class_name>:
        def __init__(self, <other arguments if needed>):
                # code for construction
```

* **Update** the person class with the __init__ method.

****Self** is just a notational convention, you can use any other name, but better to stick to self

Gaurav Gupta

Destructor and __del__()

- The code for **Destructor** in python goes into the **__del__(self)** method.
- So the **__del__** method is invoked only when the object is garbage collected.
- Since python uses reference counting mechanism to keep track of objects, your object may never be destroyed, till the program terminates.

* Add a destructor to the **Person** class

Gaurav Gupta

Operator Overloading in Python

- Operators are defined for types like integers, floats, lists …
- Ex: 1>2 ; 1+2 ;

```
I = [1,2,3,4]
print(I)
```

- But for custom classes, these operations have to be defined.

tuteur.py@gmail.com

Gaurav Gupta

Operator Overloading with ComplexClass

- Implement a class **ComplexNumber** that contains following attributes and methods:

  re : attribute for real part

  im : attribute for imaginary part

- Define a method **show(),** that displays the attributes of the class object

- Also define a method **add()**, that takes another **Complex** Object and returns a **new Complex Object** containing the **sum** of two objects.

Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)

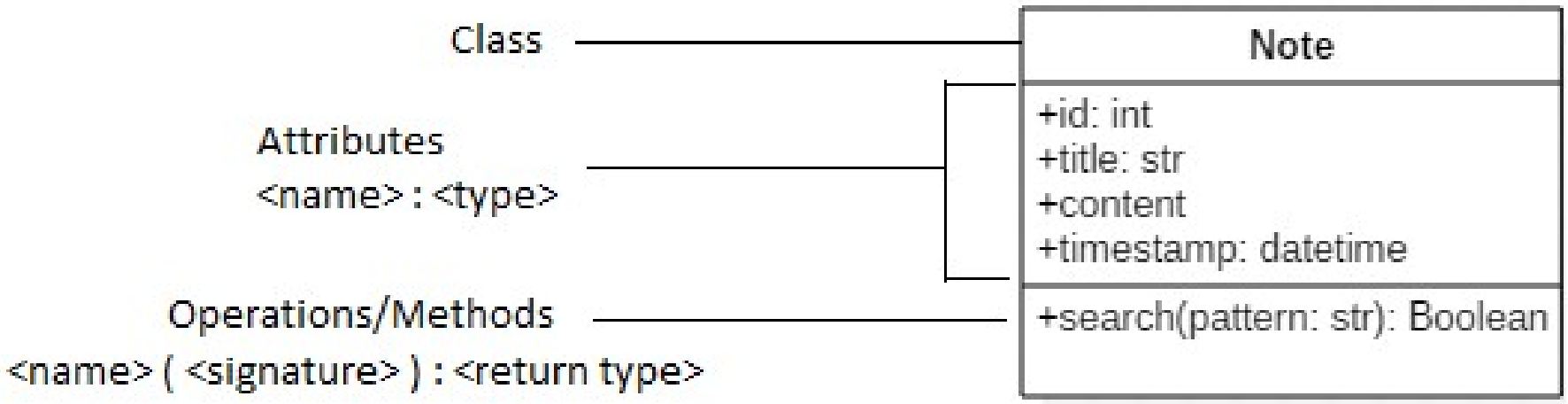
Gaurav Gupta

Class vs Instance Attributes

- Attributes can be bound to either **class** or its **instance.**
- **Class** and its **instance** are both separate namespaces
- **Class attributes** can be created directly inside the class like method, or can be assigned later.
- **Instance attributes**, are attached to the object.
- When looking a variable using object first it is searched in the objects namespace, if not found, then in the class namespace.

Gaurav Gupta

UML Notation



tuteur.py@gmail.com

Gaurav Gupta

Built-In Class Attributes

- __dict__ :Dictionary containing the class's namespace.
- __doc__ :Class documentation string or None if undefined.
- __name__:Class name.
- __module__:Module name in which the class is defined. This attribute is set to "__main__" in interactive mode.
- __bases__ :A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

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

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)
```

Gaurav Gupta

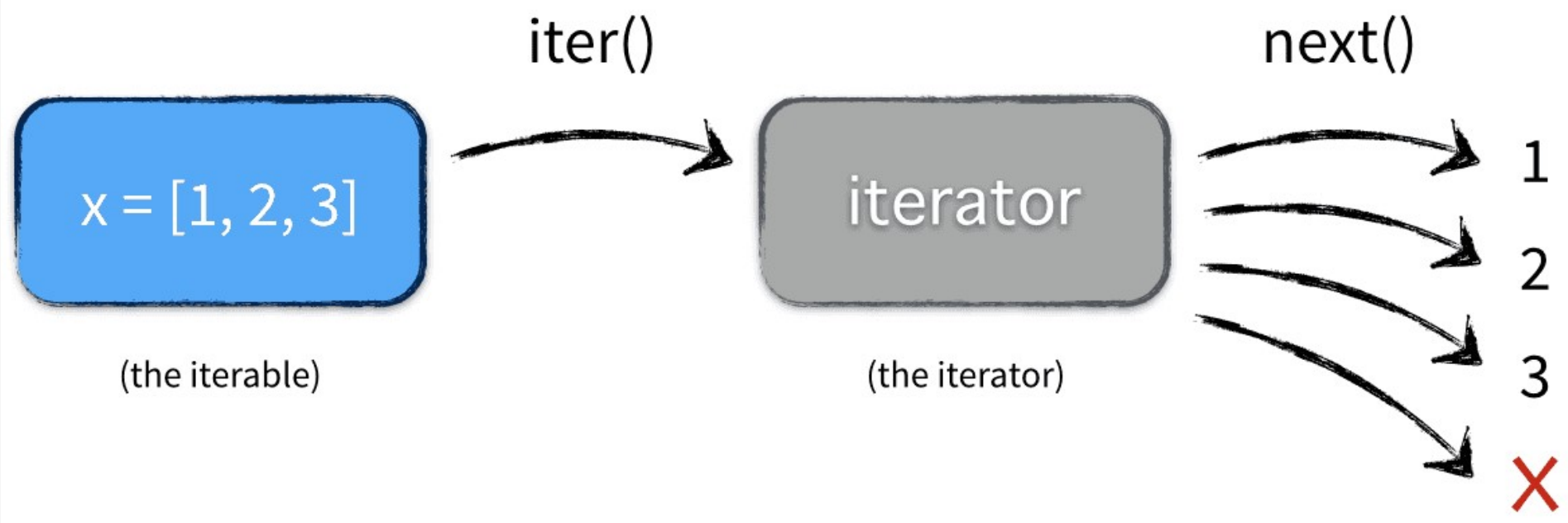
Iterator requirements

- 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)
```

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)
```

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

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()

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.

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

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

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

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.

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
```

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

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)
  ```

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

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

Gaurav Gupta

Functions-II

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

tuteur.py@gmail.com

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** ...

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

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 if 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

tuteur.py@gmail.com

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

- Available in **functools** module

Gaurav Gupta

MAP

- Map applies the function to each item of the iterable and returns a list 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.

Gaurav Gupta

Reduce

- reduce(<function with 2 arguments >, <sequence type>)
- Map 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

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

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)
```

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

Gaurav Gupta

Decorator

- Decorators are function wrappers or simply **functions** taking **functions** as **arugments** and **returning functions**.
- Python provides a special syntax for using decorators, using the @syntax

```
@<name of decorator>
def <function name>(arguments):          # normal definition of the function
        # code for the function
```

Gaurav Gupta

Seems like decorator

```
def decorator(func):                    # decorator
        print("Decorator")
        return func

def funct():                            # function we will be decorating
        print("Function")

f = decorator(funct)
print("After decorating")
f()
```

Gaurav Gupta

Actual Decorator

```
def decorator(func):                    # pass function to decorator
    def wrapper():
        print("Decorator")
        return func()                   # return whatever the function returns
    return wrapper                      # return new function from decorator

def funct():                            # function we will be decorating
    print("Function")

f = decorator(funct)
print("After decorating")
f()
```

tuteur.py@gmail.com

Gaurav Gupta

Decorator syntax

- *def <decorator function name>(wrapped function arguments):*
  *def <wrapper name>(*args, **kwargs):*
  # some operation involving function argument
  *return <wrapped function>(*args, **kwargs)*
  return <wrapper name>

- Decorate a function to print execution time of a function
- Write a decorator to call a function **n** times

tuteur.py@gmail.com

Gaurav Gupta

Recursive function

- A function that calls itself is a recursive function

- Printing first 10 natural numbers

- Finding sum of first **N** natural numbers

tuteur.py@gmail.com

Gaurav Gupta

Generator and yield

- Generator is a method containing a **yield** statement.
- Generators can be used in **for** loops for iteration.
- Instead of a **return**, the **yield** stops the method when executed and returns the yield value.
- On next iteration, the next value is yielded on the basis of the function logic, continuing from the previous state

- WAG to replicate the range method.
- WAG to generate a string in reverse order.

tuteur.py@gmail.com

Gaurav Gupta

Classes and Inheritance

- Inheritance
- Is-A vs Has-A
- Python Inheritance
- Overriding Methods
- MRO
- Super

tuteur.py@gmail.com

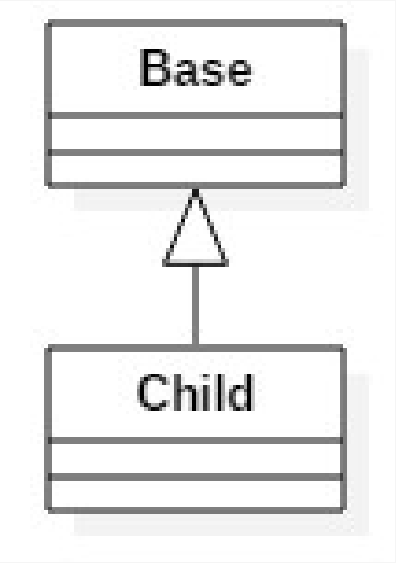
Gaurav Gupta

Inheritance

- Inheritance is the ability of a class to inherit the properties and operations of another class.
- Removes code redundancy or duplication
- Representing real world relationship between entities into the code.
- Allows code re-usage.

Inheritance Terminology

- **Base, Parent, Super Class** : The class from which other classes inherit
- **Derived, Child, Sub Class** : The class which is going to inherit from a Base class
- This relationship is also called as **Generalization**



Gaurav Gupta

Inheritance in Python

- In Python3 all classes inherit from the default ***object*** class.
- Syntax for inheritance in python:

```
class <Derived Class> (<Base Class>):
    # member definition
```

* Lets Start By creating Pizzas

tuteur.py@gmail.com

Gaurav Gupta

Overriding and Super

- **Overriding** : giving a custom implementation instead of using base class implementation of a method.

- **super** is used to invoke the base class implementation of an Overridden method
- Base class methods can also be invoked using Base class name and the method name.
- But in Python3 we have the **super** method available.

tuteur.py@gmail.com

Gaurav Gupta

Is-A vs Has-A

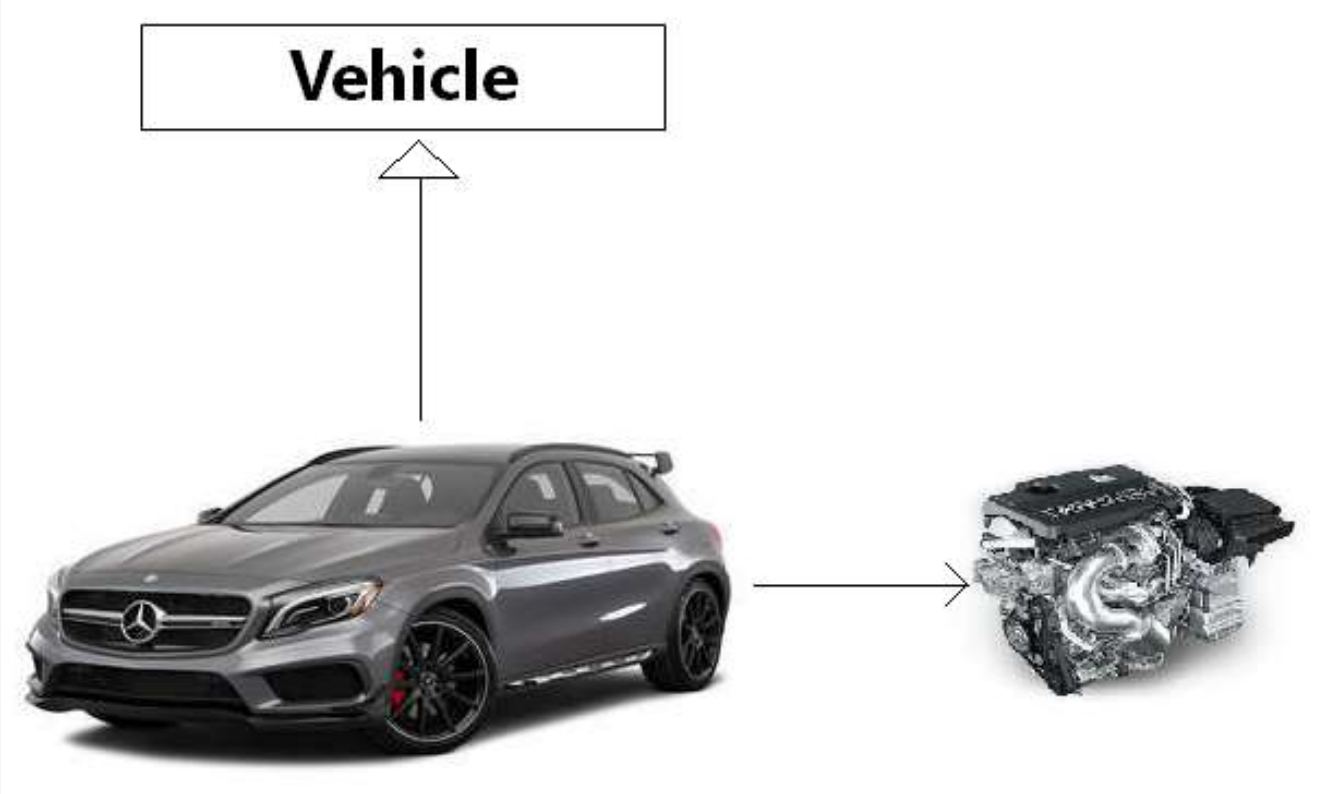
- **Is-A :** Is-A relationship means the concept of Inheritance.
- **Has-A :** Represents the concept of association, containership and aggregation.
- Car **Is-A** Vehicle

  Car **Has-A** Engine

  But if we say *Car Is-A Engine*... Doesn't sound good.

Gaurav Gupta

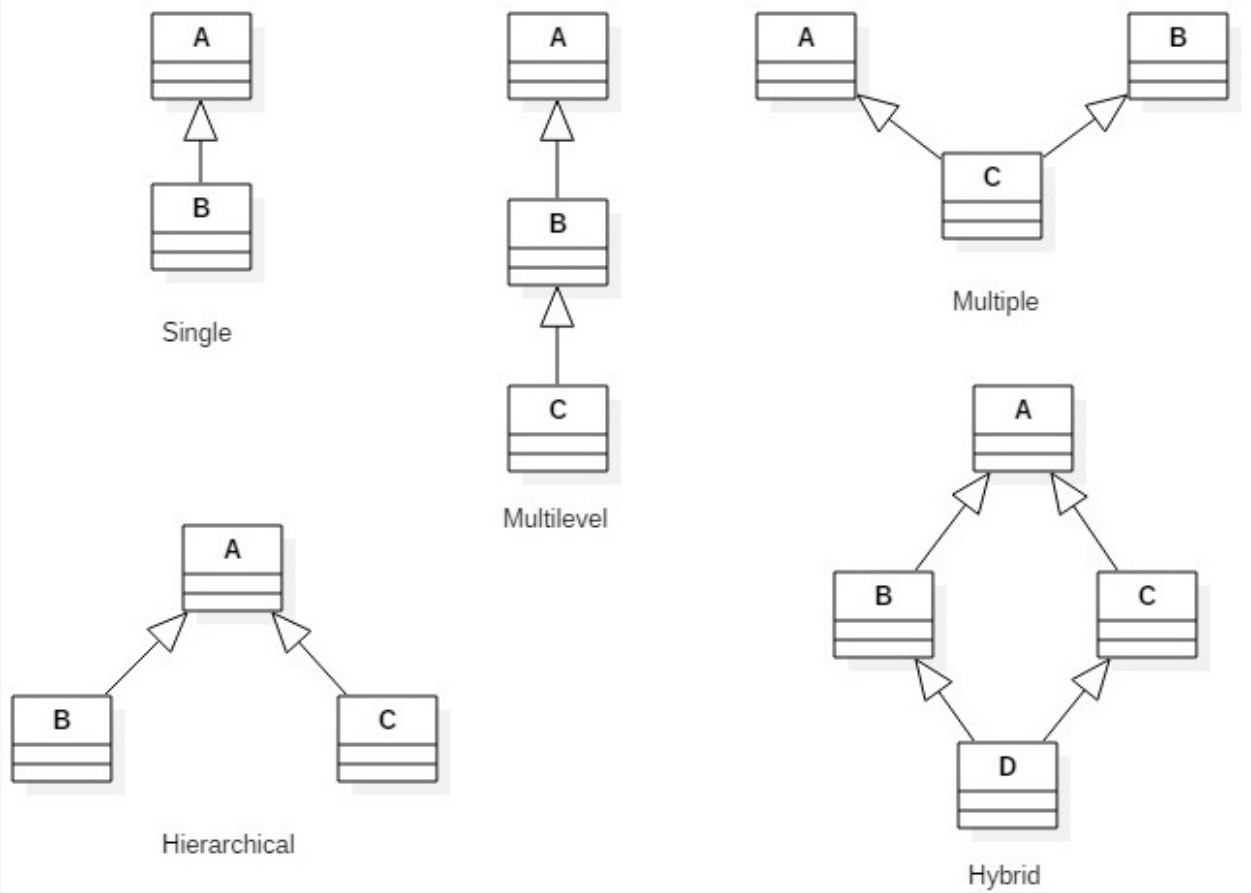
Car and Engine



tuteur.py@gmail.com

Gaurav Gupta

Inheritance Types



tuteur.py@gmail.com

Gaurav Gupta

Class Method and Static Method

```
class Demo:
    def norm_function(self, value):
        print("Method", self.__class__, value)
    @classmethod
    def class_function(clas, value):
        print("Method", clas.__name__, value)
    @staticmethod
    def static_function(value):
        print("Method", value)
```

Gaurav Gupta

Custom Exceptions

- All user defined Exceptions should inherit from Exception Class
- *class MyException(Exception):*

  pass

- When creating an exception hierarchy, the except clauses should appear in the order from the **derived to base** class.

tuteur.py@gmail.com

Gaurav Gupta

Exception Hierarchy

- **BaseException** : Parent of all exception classes in Python
- **Exception**: Inherits from BaseException. Parent of all exceptions except some system exceptions (SystemExit, KeyboardInterrupt…)
- All Exception classes should inherit from *Exception* and not *BaseException*

tuteur.py@gmail.com