URBAN EDGE

# Python programming notes by Harry

## What is Programming?

Just like we use Hindi or English to communicate with each other, we use a programming language like Python to communicate with the computer.
Programming is a way to instruct the computer to perform various tasks.

## What is Python?

Python is a simple and easy to understand language which feels like reading simple english. This Pseudo Code nature of python makes it easy to learn and understandable by beginners

## Features of Python

Easy to understand = Less development time
Free and open source
High level language
Portable → Works on Linux / Windows / Mac.
+ Fun to work with

## Installation

Python can be easily installed from python.org
When you click on the download button, python can be installed right after you complete the setup by executing the file for your platform

URBAN EDGE

# Chapter 1 : Modules, Comments & pip

Lets write our very first python program. Create a file called hello.py and paste the below code in it

```
print ("Hello World")
```
→ print is a function (More later)

Execute this file (.py file) by typing python hello.py and you will see Hello World printed on the screen.

## Modules

A module is a file containing code written by somebody else (usually) which can be imported and used in our programs

## Pip

Pip is the package manager for python
You can use pip to install a module on your system

↳ pip install flask installs flask module.

## Types of modules

There are two types of modules in Python

1) Built in modules → Pre installed in python
2) External modules → Need to install using pip.

Some examples of built in modules are os, abc, etc.
Some examples of external modules are tensorflow, flask etc.

## Using Python as a Calculator

We can use python as a calculator by typing "python" + ↵ on the terminal

↳ This opens REPL or Read Evaluate Print Loop

## Comments

Comments are used to write something which the programmer does not want to execute.

↳ Can be used to mark author name, date etc.

## Types of Comments

There are two types of comments in Python

1. Single line Comments → Written using #
2. Multi line Comments → Written using ''' Comment '''

URBAN EDGE

# Chapter 2 - Variables and Datatypes

A variable is the name given to a memory location in a program. for example

```
a = 30
b = "Harry"
c = 71.22
```

→ Variables = Containers to store a value.
→ Keywords = Reserved words in Python
Identifiers = class/function/variable name

## Data Types

Primarily there are following data types in Python

1. Integers
2. Floating point numbers
3. Strings
4. Booleans
5. None

Python is a fantastic language that automatically identifies the type of data for us.

```
a = 71          ⇒ Identifies a as class<int>
b = 88.44       ⇒ Identifies b as class <float>
name = "Harry"  ⇒ Identifies name as class <str>
```

Rules for defining a Variable name → Also applies to other Identifiers

→ A variable name can contain alphabets, digits and underscores.
→ A variable name can only start with an alphabet and underscore.
→ A variable name cant start with a digit
→ No white space is allowed to be used inside a variable name.

URBAN EDGE

Examples of a few variable names are :-
harry, one8, seven_, _seven etc.

Operators in Python
Following are some common operators in Python:
1. Arithmetic operators ⇒ +, -, *, / etc.
2. Assignment operators ⇒ =, +=, -= etc.
3. Comparison operators ⇒ ==, >, >=, <, != etc.
4. Logical operators ⇒ and, or, not

type() function and Typecasting
type function is used to find the data type of a given variable in Python.

```
a = 31
type(a) => class <int>

b = "31"
type(b) => class <str>
```

A number can be converted into a string and vice versa (if possible)
There are many functions to convert one data type into another

```
str(31) => "31"      => Integer to String Conversion
int("32") => 32      => String to Integer Conversion
float(32) => 32.0    => Integer to Float Conversion

... and so on
```

Here "31" is a string literal and 31 a numeric literal.

URBAN EDGE

input() function
This function allows the user to take input from the keyboard as a string

a = input("Enter name") ⇒ If a is "harry", the user entered harry

It is important to note that the output of input is always a string (even if the number is entered) ⇒ If a is "34" user entered 34

URBAN EDGE

# Chapter 3 - Strings

String is a data type in Python.
String is a sequence of characters enclosed in quotes.
We can primarily, write a string in these three ways

1. Single quoted strings → a = 'Harry'
2. Double quoted strings → b = "Harry"
3. Triple quoted strings → c = '''Harry'''

## String Slicing

A String in Python can be sliced for getting a part of the string.
Consider the following string:

name = "H a r r y" ⇒ length = 5

| | H | a | r | r | y |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |
| | (-5) | (-4) | (-3) | (-2) | (-1) |

The index in a string starts from 0 to (length-1) in Python. In order to slice a string, we use the following syntax:

sl = name [ind_start : ind_end]
first index included — last index is not included

sl [0:3] returns "Har" → characters from 0 to 3
sl [1:3] returns "ar" → characters from 1 to 3

Negative Indices: Negative Indices can also be used as shown in the figure above. -1 corresponds to the (length-1) index, -2 to (length-2)

URBAN EDGE

Slicing with skip value
We can provide a skip value as a part of our slice like this :

```
word = "amazing"
word [1:6:2] → 'mzn'
```

Other advanced slicing techniques

```
word = "amazing"
word [:7] → word [0:7] → 'amazing'
word [0:] → word [0:7] → 'amazing'
```

String functions
Some of the mostly used functions to perform operations on or manipulate strings are :

1> len() function → This function returns the length of the string

len ("Harry") → returns 5

2> string.endswith ("xxy") → This function tells whether the variable string ends with the string "xxy" or not. if string is "Harry", it returns true for "xxy" since Harry ends with xxy

3> string.count ("c") → Counts the total number of occurence of any character

4> string.capitalize() → This function capitalizes the first character of a given string.

URBAN EDGE

5. String.find(word) - This function finds a word and returns the index of first occurence of that word in the string.

6. String.replace(oldword, newword) - This function replaces the oldword with newword in the entire string.

Escape Sequence Characters
Sequence of characters after backslash '\' → Escape Seq characters

Escape sequence character comprises of more than one characters but represents one character when used within the strings.

Examples \n, \t, \', \\ etc.
(\n → newline, \t → Tab, \' → Single quote, \\ → backslash.)

# Chapter 4 - Lists and Tuples

Python Lists are containers to store a set of values of any data type

```
friends = ["Apple", "Akash", "Rohan", 7, False]
```

"Apple" → str()
"Akash" → can store value of any datatype
7 → int()
False → bool()

## List Indexing

A List can be indexed just like a String

```
L1 = [7, 9, "Harry"]
```

```
L1[0]  => 7
L1[1]  => 9
L1[70] => Error
L1[0:2] => [7, 9]   => List Slicing
```

## List Methods

Consider the following list:

```
L1 = [1, 8, 7, 2, 21, 15]
```

1. L1.sort(): updates the list to [1, 2, 7, 8, 15, 21]
2. L1.reverse(): updates the list to [15, 21, 2, 7, 8, 1]
3. L1.append(8): adds 8 at the end of the list
4. L1.insert(3,8): This will add 8 at 3 index

URBAN EDGE

5> L1.pop(2): Will delete element at index 2 and return its value

6> L1.remove(21): Will remove 21 from the list.

## Tuples in Python

A tuple is an immutable data type in python.
↳ Cannot change

a = ( ) => Empty tuple
a = (1,) => Tuple with only one element needs a comma
a = (1, 7, 2) => Tuple with more than one element

Once defined a tuple's elements cant be altered or manipulated.

## Tuple methods

Consider the following tuple
a = (1, 7, 2)

1> a.count(1): a.count(1) will return number of times 1 occurs in a.

2> a.index(1): a.index(1) will return the index of first occurence of 1 in a.

# Chapter 5 : Dictionary & Sets

Dictionary is a collection of key-value pairs

Syntax :

```
a = { "key" : "value",
      "harry" : "code",
      "marks" : "100",
      "list" : [1,2,9] }
```

(keys → "key", "harry", "marks", "list"; values → "value", "code", "100", [1,2,9])

a["key"] => Prints "value"
a["list"] => Prints [1,2,9]

Properties of a Python Dictionaries

1. It is unordered
2. It is mutable
3. It is indexed
4. Cannot contain duplicate keys

Dictionary Methodes

Consider the following dictionary

```
a = { "name" : "Harry",
      "from" : "India",
      "marks" : [92,98,96] }
```

1. a.items() : returns a list of (key, value) tuples
2. a.keys() : returns a list containing dictionary's keys
3. a.update({"friend" : "Sam"}) : updates the dictionary with supplied key-value pairs

URBAN EDGE

4. a.get("name"): returns the value of the specified keys (and value is returned eg. "Harry" is returned here)

More methods are available on docs.python.org.

Sets in Python

Set is a collection of non repetitive elements

```
S = Set()                    ⇒ No repetition allowed!
S.add(1)
S.add(2)                     ⇒ or Set = {1, 2}
```

If you are a programming beginner without much knowledge of mathematical operations on sets, you can simply look at sets in python as data types containing unique values.

Properties of Sets

1. Sets are unordered ⇒ Element's order doesn't matter
2. Sets are unindexed ⇒ Cannot access elements by index
3. There is no way to change items in sets.
4. Sets cannot contain duplicate values

Operations on Sets

Consider the following Set:

S = {1, 8, 2, 3}

1. len(s): Returns 4, the length of the set
2. S.remove(8): Updates the set S and removes 8 from S.

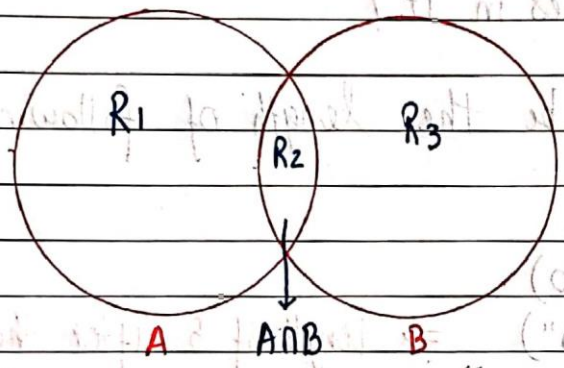
URBAN EDGE

3. S.pop() : Removes an arbitrary element from the set and returns the element removed

4. S.clear() : Empties the set S

5. S.union({8,11}) : Returns a new set with all items from both sets. => {1,8,2,3,11}

6. S.intersection({8,11}) : Returns a set which contains only items in both sets. => {8}




$R_2 \Rightarrow A \cap B$

$R_1 + R_2 + R_3 \Rightarrow A \cup B$

$R_1 + R_3 \Rightarrow A \Delta B$

$R_1 \Rightarrow A - B$

$R_3 \Rightarrow B - A$

# Chapter 6 - Conditional Expressions

Sometimes we want to play pubG on our phone if the day is Sunday.
Sometimes we order Icecream online if the day is Sunny.
Sometimes we go hiking if our parents allow.

All these are decisions which depends on a condition being met.
In Python programming too, we must be able to execute instructions on a condition(s) being met. This is what conditionals are for!

## If else and elif in Python

If else and elif statements are a multiway decision taken by our program due to certain conditions in our code.

Syntax:

```
if (condition1):                    => if condition 1 is true
    print ("yes")
elif (condition2):                  => if condition 2 is true
    print ("No")
else:                               => otherwise
    print ("May be")
```

Indentation ←

Code example:

```
a = 22
if(a>9):
    print("Greater")
else:
    print("lesser")
```

Quick Quiz: Write a program to print yes when the age entered by the user is greater than or equal to 18.

## Relational Operators

Relational operators are used to evaluate conditions inside the if statements. Some examples of relational operators are:

- == → equals
- >= → greater than/equal to
- <=, etc.

## Logical operators

In python logical operators operate on conditional statements. Example:

- and → true if both operands are true else false
- or → true if at least one operand is true else false
- not → inverts true to false & false to true

## elif clause

elif in python means [else if]. An if statement can be chained together with a lot of these elif statements followed by an else statement

```
if (Condition1):
    # code
elif (condition 2):
    # code
elif (condition 3):
    # code
:
:
else:
    # code
```

⇒ This ladder will stop once a condition in an if or elif is met.

Important notes:

1. There can be any number of elif statements.
2. last else is executed only if all the conditions inside elifs fail.

URBAN EDGE

# Chapter 7 - Loops in Python

Sometimes we want to repeat a set of statements in our program. for instance: Print 1 to 1000

Loops make it easy for a programmer to tell the computer, which set of instructions to repeat and how!

## Types of loops in Python

Primarily there are two types of loops in Python

1. While loop
2. For loop

We will look into these one by one!

## While loop

The syntax of a while loop looks like this:

```
While Condition:
    #Body of the loop
```

⇒ The block keeps executing until the condition is true

In while loops, the condition is checked first. If it evaluates to true, the Body of the loop is executed, otherwise not!
If the loop is entered, the process of [Condition check & execution] is continued until the condition becomes false.

Quick Quiz: Write a program to print 1 to 50 using a while loop.

URBAN EDGE

## An Example

```
i = 0
while i < 5:
    print("Harry")
    i = i + 1
```

⇒ Prints "Harry" - 5 times!

Note: If the condition never becomes False, the loop keeps getting executed.

Quick Quiz: Write a program to print the content of a list using while loops.

## For loop

A for loop is used to iterate through a sequence like list, tuple or string [Iterables]

The syntax of a for loop looks like this:

```
l = [1, 7, 8]
for item in l:
    print(item)
```

→ print 1, 7 and 8

## Range function in Python

The range function in python is used to generate a sequence of numbers.
We can also specify the start, stop and step-size as follows:

```
range(start, stop, step_size)
```

↳ step size is usually not used with range()

URBAN EDGE

## An Example demonstrating range() function

```
for i in range(0, 7):          → range(7) can also be used
    print(i)                   → prints 0 to 6
```

## For loop with else

An optional else can be used with a for loop if the code is to be executed when the loop exhausts

Example :

```
l = [1, 7, 8]
for item in l:
    print(item)
else:
    print("Done")              → This is printed when the loop exhausts!
```

Output :

```
1
7
8
Done
```

## The break statement

'break' is used to come out of the loop when encountered
It instructs the program to - Exit the loop now

Example :

```
for i in range(0,80):
    print(i)                   → This will print 0, 1, 2 and 3
    if i==3:
        break
```

## The continue Statement

'continue' is used to stop the current iteration of the loop and continue with the next one
It instructs the program to "skip this iteration"

Example:

```
for i in range(4):
    print("printing")
    if i == 2:
        continue
    print(i)
```

⇒ if i is 2, the iteration is skipped

## pass statement

pass is a null statement in python
It instructs to "Do nothing"

Example:

```
l = [1, 7, 8]
for item in l:
    pass
```

→ Without pass, the program will throw an error

URBAN EDGE

# Chapter 8 - Functions & Recursions

A function is a group of statements performing a specific task.

When a program gets bigger in size and its complexity grows, it gets difficult for a programmer to keep track on which piece of code is doing what!

A function can be reused by the programmer in a given program any number of

Example and Syntax of a function
The syntax of a function looks as follows:

```
def func1():
    print("Hello")
```

This function can be called any number of times, anywhere in the program.

Function Call
Whenever we want to call a function, we put the name of the function followed by parenthesis as follows:

```
func1()
```

→ This is called function call

Function definition
The part containing the exact set of instructions which are executed during the function call.

URBAN EDGE

Quick Quiz : Write a program to greet a user with "Good day" using functions.

Types of functions in Python
There are two types of functions in Python:
1. Built in functions → Already present in Python
2. User defined functions → Defined by the user

Examples of built in function includes len(), print(), range() etc.

The func1() function we defined is an example of user defined function

Functions with arguments
A function can accept some values it can work with. We can put these values in the parenthesis
A function can also return values as shown below:

```
def greet(name):
    gr = "Hello " + name
    return gr

a = greet("Harry")
```

→ "Harry" is passed to greet in name

→ a will now contain "Hello Harry"

Default Parameter Value
We can have a value as default argument in a function.
If we specify name = 'stranger' in the line containing def, this value is used when no argument is passed.

For example:

```
def greet(name = "Stranger"):
        # function body
```

greet() → Name will be "Stranger" in function body (default)
greet("Harry") → Name will be "Harry" in function body (passed)

## Recursion

Recursion is a function which calls itself
It is used to directly use a mathematical formula as a function. For example:

factorial(n) = n × factorial(n-1)

This function can be defined as follows:

```
def factorial(n):
    if i==0 or i==1:  → Base condition which doesn't call the function any further
        return 1
    else:
        return n * factorial(n-1)  → Function calling itself
```

This works as follows:

Factorial(4)
↓ ↳ [Function called]
4 × Factorial(3)
4 × [3 ↓ × factorial(2)]
4 × 3 × [2 ↓ × factorial(1)]
4 × 3 × 2 × [1] [Function returned]

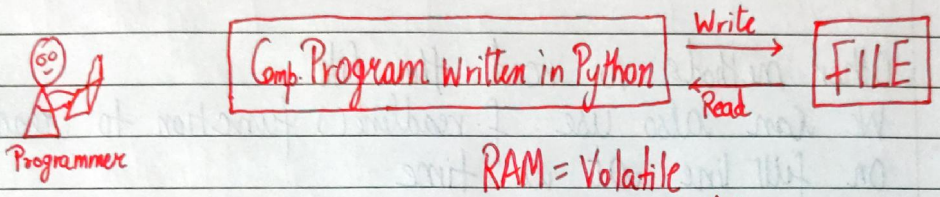
The programmer need to be extremely careful while working with recursion to ensure that the function doesnt Infinitely keep calling itself.
Recursion is sometimes the most direct way to code an algorithm

URBAN EDGE

# Chapter 9 - File I/O

The random Access memory is volatile and all its contents are lost once a program terminates
In order to persist the data forever, we use files

A file is data stored in a storage device. A Python program can talk to the file by reading content from it and writing content to it




RAM = Volatile
HDD = Non Volatile

## Types of Files

There are 2 types of files:

1. Text files (.txt, .c etc)
2. Binary files (.jpg, .dat, etc)

Python has a lot of functions for reading, updating and deleting files.

## Opening a file

Python has an open() function for opening files. It takes 2 parameters: filename and mode.

```
open("this.txt", "r")
```

- open is a built-in function
- "this.txt" → Filename
- "r" → mode of opening (read mode)

# Reading a file in python

```
f = open("this.txt", "r")  → open the file in r mode
text = f.read()            → Read its contents
print(text)                → Print its contents
f.close()                  → Close the file
```

We can also specify the number of characters in read() function : f.read(2)
↳ Reads first 2 characters

Other methods to read the file
We can also use f.readline() function to read on full line at a time

f.readline() → Reads one line from the file

## Modes of opening a file

r → open for reading
w → open for writing
a → open for appending
+ → open for updating

'rb' will open for read in binary mode
'rt' will open for read in text mode

## Writing files in Python

In order to write to a file, we first open it in write or append mode after which, we use the python's f.write() method to write to the file!

URBAN EDGE

```
f = open("this.txt", "w")
f.write("This is nice")
f.close()
```

→ Can be called multiple times

With statement

The best way to open and close the file automatically is the with statement

```
with open("this.txt") as f:
    f.read()
```

↳ Dont need to write f.close() as it is done automatically.

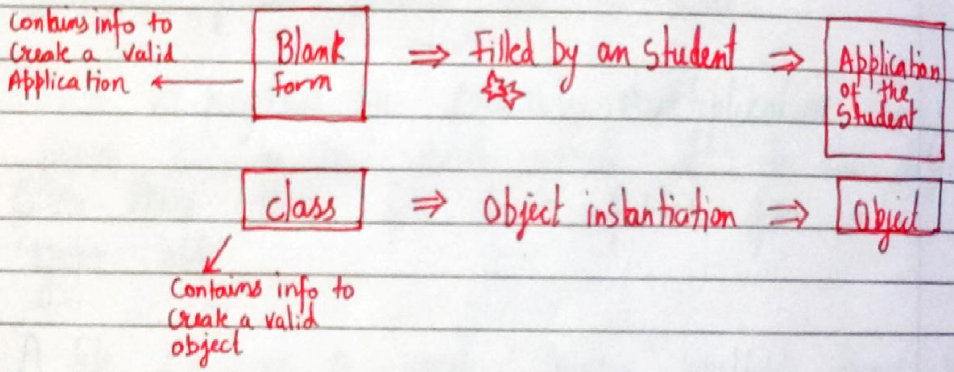
# Chapter 10 - Object Oriented Programming

Solving a problem by creating objects is one of the most popular approaches in programming. This is called Object oriented programming.

This concept focuses on using reusable code.
↳ Implements DRY principle

## Class

A class is a blueprint for creating objects.




The syntax of a class looks like this:

```
Class Employee:
    # methods & variables
```

[Classname is written in PascalCase]

## Object

An Object is an instantiation of a class. When class is defined, a template (info) is defined. Memory is allocated only after object instantiation.

Objects of a given class can invoke the methods available to it without revealing the implementation details to the user. → Abstraction & Encapsulation!

URBAN EDGE

## Modelling a problem in OOPs

We identify the following in our problem

Noun → Class → Employee
Adjective → Attributes → name, age, salary
Verbs → Methods → getSalary(), increment()

## Class Attributes

An attribute that belongs to the class rather than a particular object.

Example :

```
Class Employee :
    company = "Google"        → [Specific to each class]

harry = Employee()            → object instantiation
harry.company
Employee.company = "YouTube"  → changing class attribute
```

## Instance Attributes

An attribute that belongs to the Instance (object)
Assuming the class from the previous example :

```
harry.name = "Harry"
harry.salary = "30K"          ⇒ Adding instance attributes
```

Note : Instance attributes take preference over class attributes during assignment & retrieval

harry.attribute1 → ① Is attribute1 present in object ?
② Is attribute1 present in class ?

## self parameter

self refers to the instance of the class
It is automatically passed with a function call from an object

```
harry.getSalary()
```
→ here self is harry
→ equivalent to Employee.getSalary(harry)

The function getsalary is defined as:

```
class Employee:
    company = "Google"
    def getSalary(self):
        print("Salary is not there")
```

## Static method

Sometimes we need a function that doesnt use the self parameter. We can define a static method like this:

```
@staticmethod
def greet():
    print("Hello user")
```
→ decorator to mark greet as a static method

## __init__() Constructor

__init__() is a special method which is first run as soon as the object is created.
__init__() method is also known as constructor

It takes self argument and can also take further arguments

URBAN EDGE

For Example:

```
Class Employee:
    def __init__(self, name):
        self.name = name

    def getSalary(self):
        ...

harry = Employee("Harry")
```

↳ Object can be instantiated using constructor like this!

URBAN EDGE

# Chapter 11 - Inheritance & more on OOPs

Inheritance is a way of creating a new class from an existing class

Syntax:

```
class Employee:                    → Base Class
    # Code
    ...

class Programmer (Employee):       → Derived or child class
    # Code
```

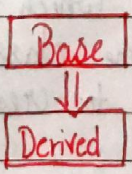
We can use the methods and attributes of Employee in Programmer object.
Also, we can overwrite or add new attributes and methods in Programmer class.

## Types of Inheritance

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance

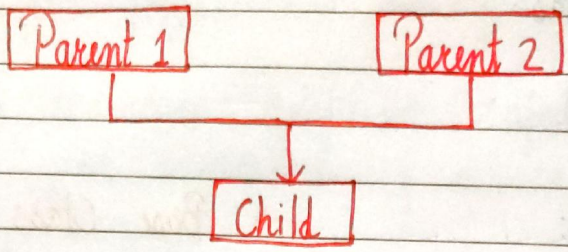
### Single Inheritance

Single inheritance occurs when child class inherits only a single parent class



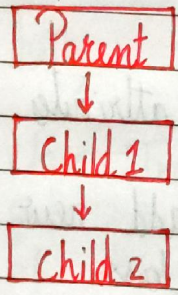
## Multiple Inheritance

Multiple inheritance occurs when the child class inherits from more than one parent class.




## Multilevel Inheritance

When a child class becomes a parent for another child class




## Super() method

Super method is used to access the methods of a super class in the derived class

```
super().__init__()
```

↳ calls constructor of the base class

## Class methods

A class method is a method which is bound to the class and not the object of the class.

@classmethod decorator is used to create a class method

Syntax to create a class method:

```
@classmethod
def (cls, p1, p2):
    ...
```

@property decorators
Consider the following class

```
class Employee:
    @property
    def name(self):
        return self.ename
```

if e = Employee() is an object of class employee, we can print(e.name) to print the ename/call name() function.

@.getters and @.setters
The method name with @property decorator is called getter method
We can define a function + @name.setter decorator like below:

```
@name.setter
def name(self, value):
    self.ename = value
```

Operator overloading in Python
Operators in python can be overloaded using dunder methods.
These methods are called when a given operator is used on the objects.

Operators in python can be overloaded using the following methods:

$p_1 + p_2 \rightarrow p_1.\_\_add\_\_(p_2)$

$p_1 - p_2 \rightarrow p_1.\_\_sub\_\_(p_2)$

$p_1 * p_2 \rightarrow p_1.\_\_mul\_\_(p_2)$

$p_1 / p_2 \rightarrow p_1.\_\_truediv\_\_(p_2)$

$p_1 // p_2 \rightarrow p_1.\_\_floordiv\_\_(p_2)$

Other dunder/magic methods in python

`__str__()` → used to set what gets displayed upon calling `str(obj)`

`__len__()` → used to set what gets displayed upon calling `__len__()` or `len(obj)`

URBAN EDGE

# Chapter 12 - Advanced Python 1

## Exception Handling in Python

There are many built-in exceptions which are raised in Python when something goes wrong.
Exceptions in Python can be handled using a try statement. The code that handles the exception is written in the except clause.

```
try:
    # Code                          → Code which might throw Exception
except Exception as e:
    print(e)
```

When the exception is handled, the code flow continues without program interruption.

We can also specify the exceptions to catch like below:

```
try:
    #Code
except ZeroDivisionError:
    # Code
except TypeError:
    # code
except:
    # Code          → All other exceptions are handled here.
```

## Raising Exceptions

We can raise custom exceptions using the raise keyword in python.

URBAN EDGE ___/___/___

try with else clause
Sometimes we want to run a piece of code when try was successful.

```
try:
    # Some code
except:
    # Some code
else:
    # Code        → This is executed only if the try was successful
```

try with finally
Python offers a finally clause which ensures execution of a piece of code irrespective of the exception.

```
try:
    # Some code
except:
    # Some code
finally:
    # Some code    → executed regardless of error!
```

if __name__ == '__main__' in Python
__name__ evaluates to the name of the module in Python from where the program is ran

If the module is being run directly from the command line, the __name__ is set to string "__main__"
Thus this behaviour is used to check whether the module is run directly or imported to another file.

## The global keyword

global keyword is used to modify the variable outside of the current scope.

## Enumerate function in Python

The enumerate function adds counter to an iterable and returns it

```
for i,item in list1:
    print(i,item)
```

↳ Prints the items of list 1 with index!

## list comprehensions

list comprehension is an elegant way to create lists based on existing lists

```
list1 = [1, 7, 12, 11, 22]

list2 = [i for item in list1 if item >8]
```

URBAN EDGE

# Chapter 13 - Advanced Python 2

## Virtual Environment

An environment which is same as the System interpretor but is isolated from the other python environments on the system.

## Installation

To use virtual environments, we write

```
pip install virtualenv → Install the package
```

We create a new environment using:

```
Virtualenv myprojectenv → Creates a new venv
```

The next step after creating the virtual environment is to activate it.
We can now use this virtual environment as a separate python installation.

## pip freeze Command

pip freeze returns all the packages installed in a given python environment along with the versions

```
"pip freeze > requirements.txt"
```

The above command creats a file named requirements.txt in the same directory containing the output of pip freeze.
We can distribute this file to other users and they can recreate the same environment using:

URBAN EDGE

```
pip install -r requirements.txt
```

Lambda functions
functions created using an expression using lambda keyword
Syntax:

```
lambda arguments : expressions
```

↳ Can be used as a normal function

Example:

```
Square = lambda x : x*x
Square(6)  → returns 36

Sum = lambda a,b,c : a+b+c
Sum(1,2,3) → returns 6
```

bin method (Strings)
Creates a string from iterable objects

```
l = ["apple", "mango", "banana"]
",and,".join(l)
```

The above line will return "apple,and,mango,and,banana"

format method (strings)
formats the values inside the string into a desired output

```
template.format(p1, p2 ...)
```

↳ arguments

Syntax for format looks like:

"{} is a good {}".format("Harry", "boy") — ①

"{1} is a good {0}".format("Harry", "boy") — ②

Output for ①
Harry is a good boy

Output for ②
boy is a good Harry

Map, Filter & Reduce
Map applies a function to all the items in an input_list

Syntax:
map(function, input_list)
↑ Can be lambda function

Filter creates a list of items for which the function returns true.

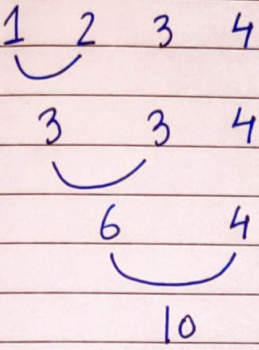
list(filter(function))
↳ Can be a lambda function

Reduce applies a rolling computation to sequential pair of elements

```
from functools import reduce
val = reduce(function, list1)
```
↳ Can be a lambda function

If the function computes sum of two numbers and the

list is [1, 2, 3, 4]




=> Sequential Computation